なぜ Zope 3？

Zope 3発展篇の発刊を記して

2010年7月14日

Plone研究会、Plone Users Group Japan合同月例会

山本 烈 (aka. retsu)

お待たせしました

2004 2005 2006 2007 2008 2009 2010

X3 3.0.0

3.3.1

3.4.0

1st Ed.

2nd Ed.

3rd Ed.

企画

入門篇

発展篇

何が書いてある？

入門篇

- コンポーネントアーキテクチャ
- インストール
- インタフェース
- コンテンツコンポーネント
- 永続性
- 簡単なビューとブラウザページ
- ブラウザフォーム
- 国際化
- サイトレイアウトのカスタマイズ
- アダプタ
- 自動化されたテスト作業
- 高度なビュー
- メタデータ
- コンテナ
- イベント

発展篇

- ソースとボキャブラリ
- サイト
- インデクシングと検索
- ブラウザセッション
- セキュリティ
- 認証とユーザ管理
- デバッグ
- パッケージングとデプロイメント
- APIリファレンス
- ZCMLリファレンス

誰が書いた？

Philipp von Weitershausen

フィリップ•フォン•ヴァイテルスハウゼン

Zope 3主要開発者の１人

- ソフトウェアと同時に本もリリース
- 絶妙なバージョンをベースに
- 極めて深淵な記述
- 豪華なレビューチーム

疑問

1. Zopeって、使われてるの？
2. Zopeって、何がすごいの？
3. Ploneと、どう関係あるの？
4. このZope 3本は、まだ役に立つの？

1. 測る

a. メーリングリスト

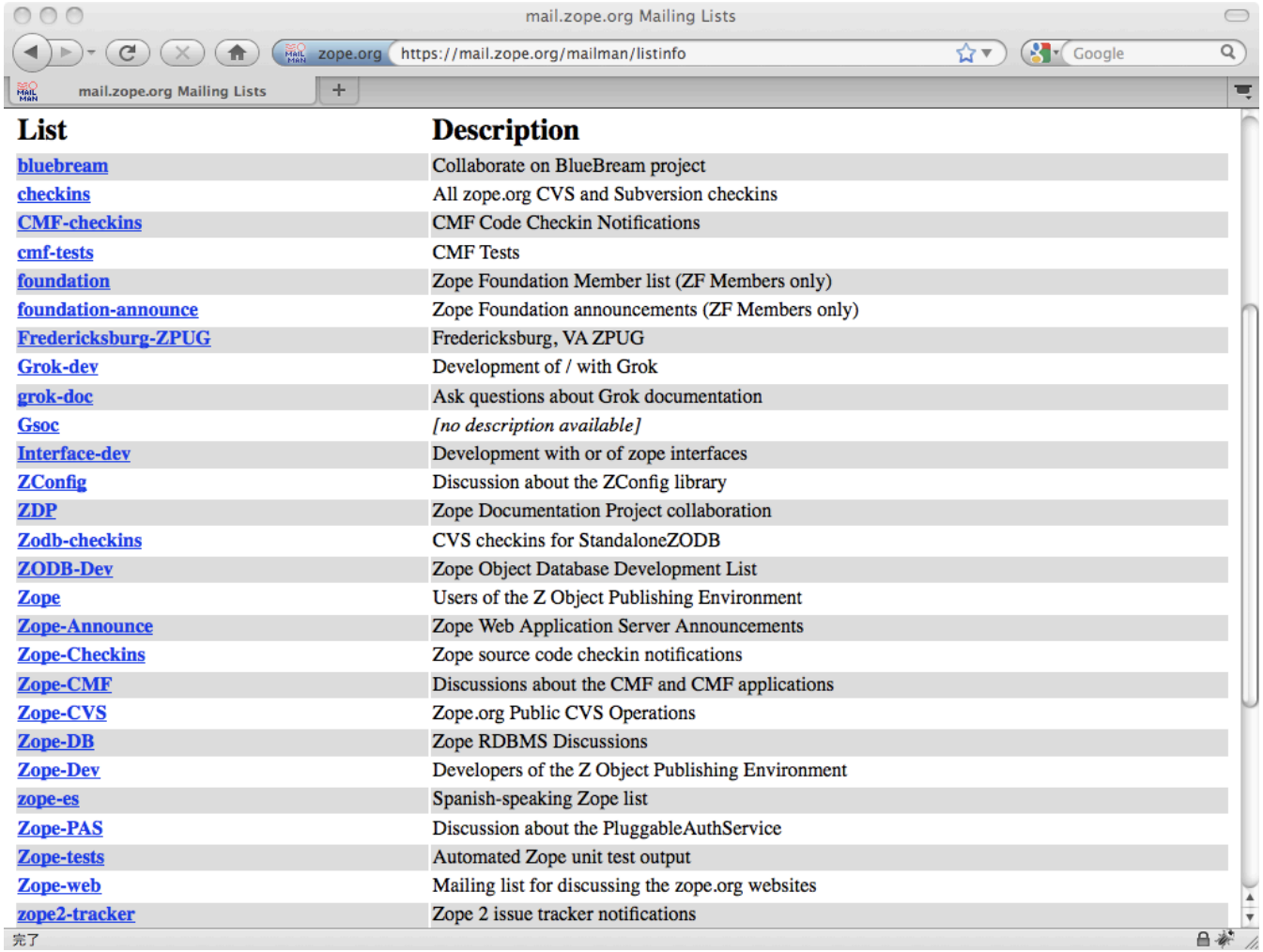
b. ソースコードリポジトリ

メーリングリスト上の活動

1. svn.zope.orgのsvn logを分析
2. 毎月のコミット数
3. コミットに含まれるパッチ数
4. コミッターの数（ユニークアカウント数）

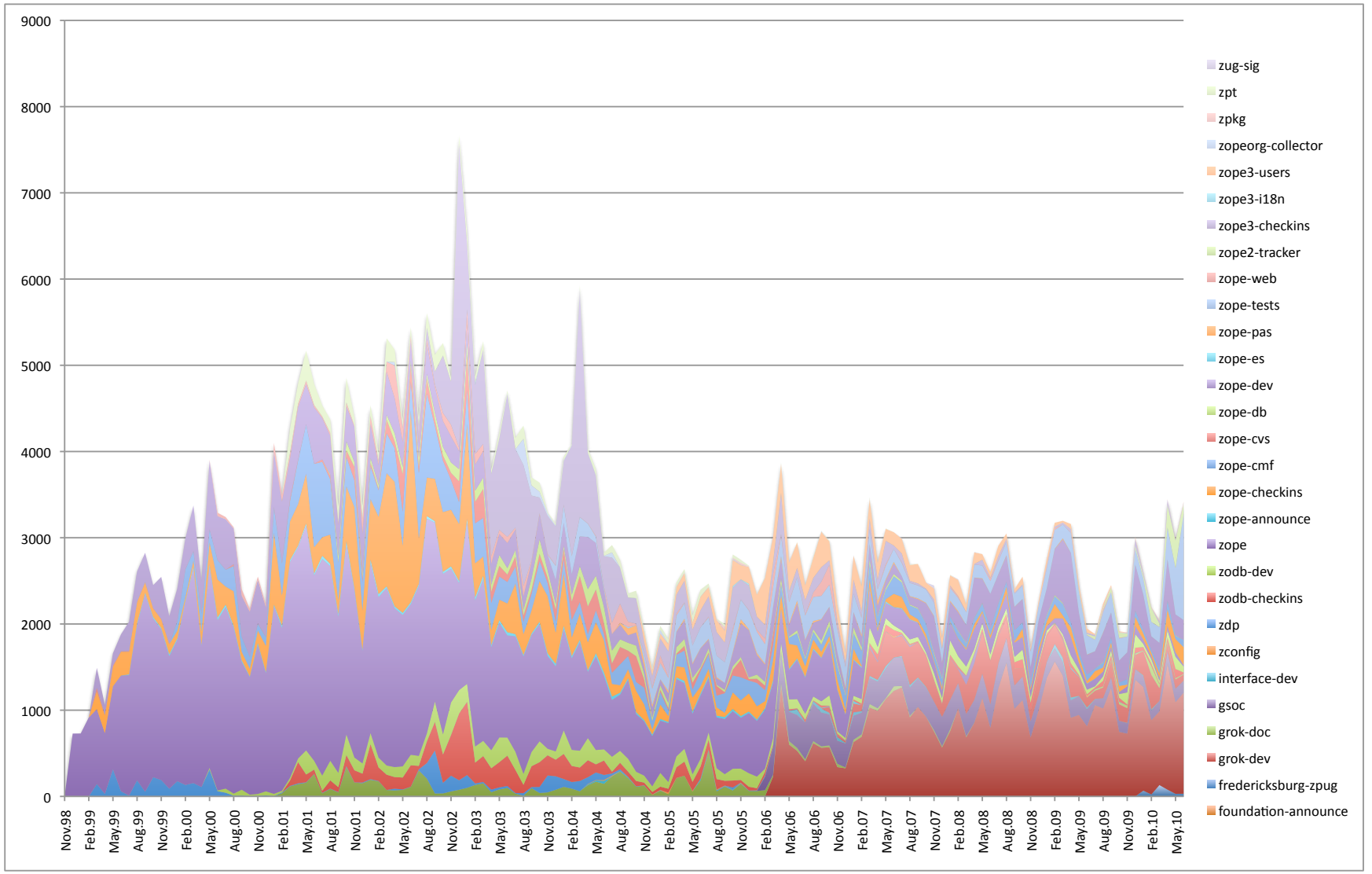
mail.zope.org

34のリスト、1998/11 – 2010/06、月単位



総投稿数

mail.zope.org全体、1998/11 - 2010/06、月単位

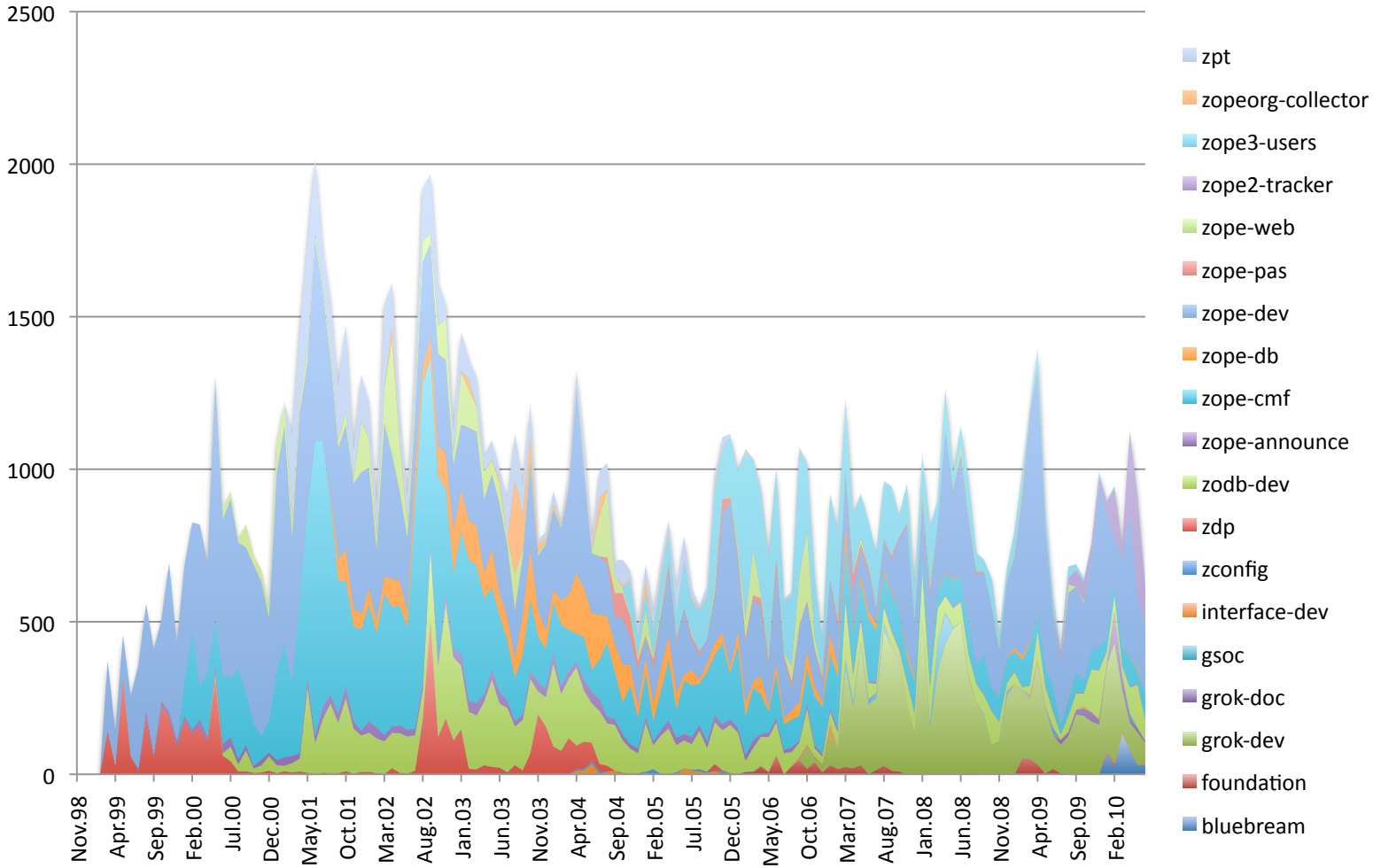


少し整理

1. zopeリスト（ユーザ系）を取り除く
2. コミット通知を取り除く
3. 自動テストの結果通知を取り除く
4. １００投稿以下のリストを取り除く

開発系議論の投稿数

mail.zope.org全体、1998/11 - 2010/06、月単位

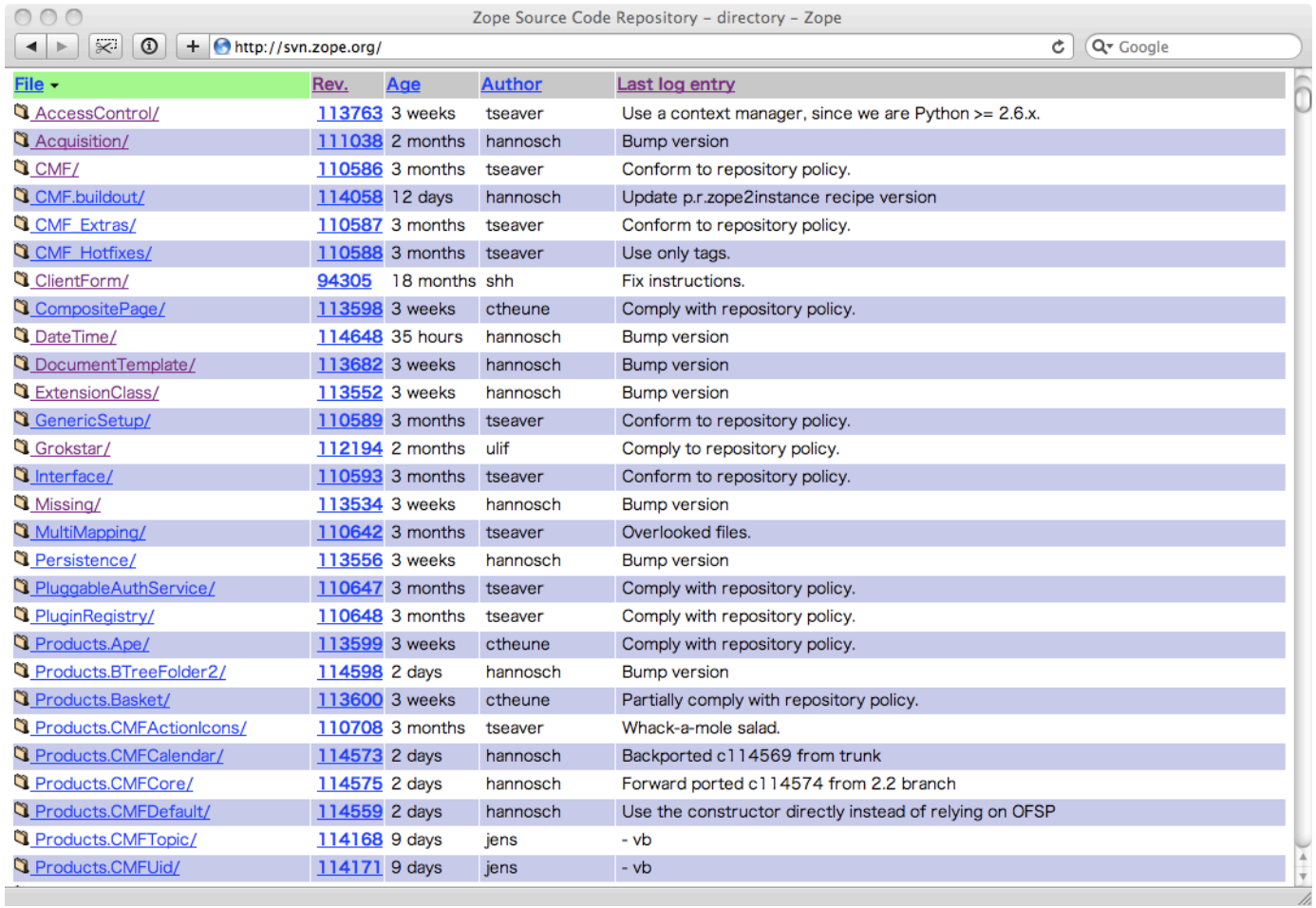


コードリポジトリ上の活動

1. svn.zope.orgのsvn logを分析
2. 毎月のコミット数
3. コミットに含まれるパッチ数
4. コミッターの数（ユニークアカウント数）

svn.zope.org

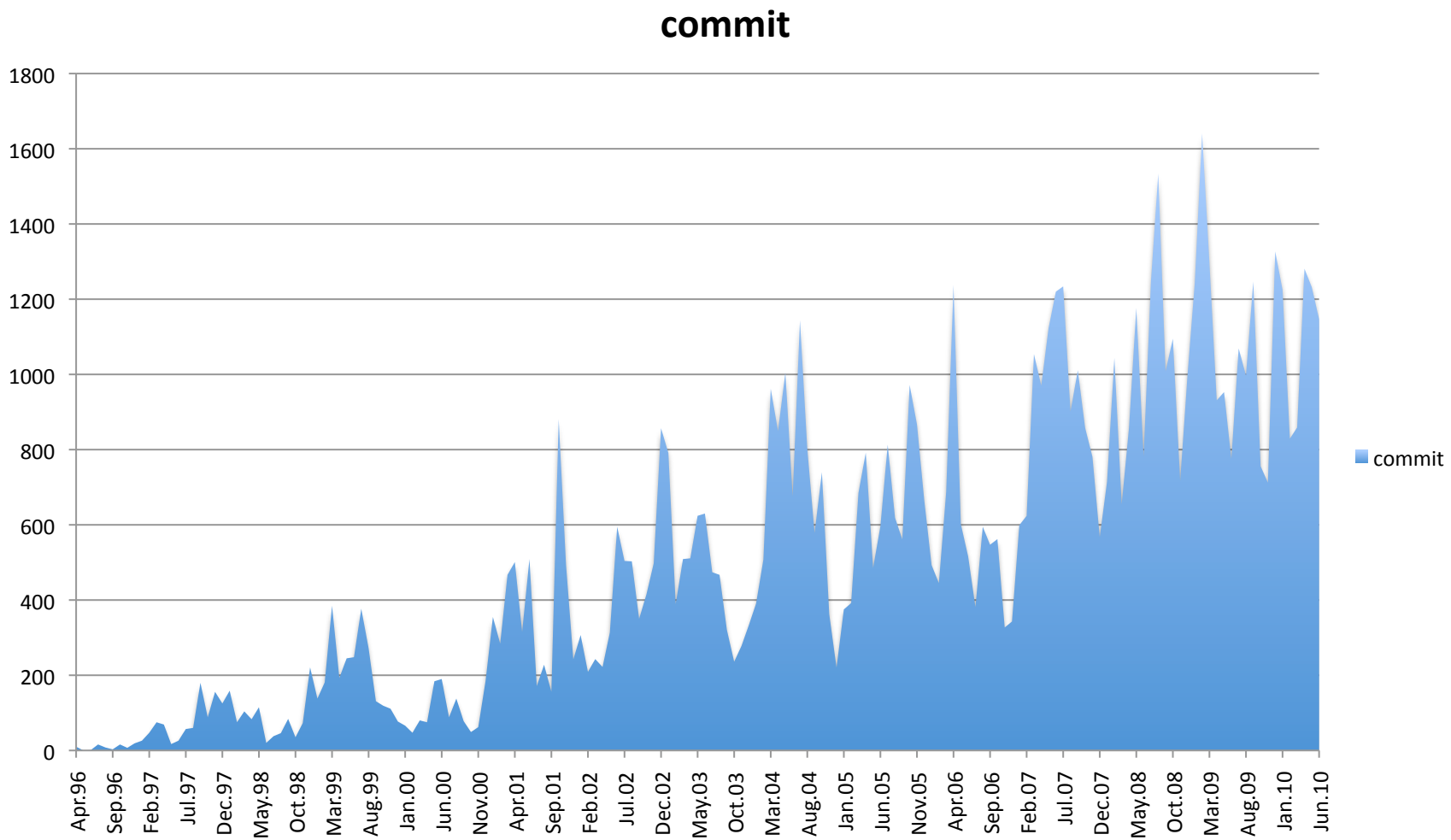
779 プロジェクト（2010年7月14日現在）



File	Rev.	Age	Author	Last log entry
AccessControl/	113763	3 weeks	tseaver	Use a context manager, since we are Python >= 2.6.x.
Acquisition/	111038	2 months	hannosch	Bump version
CMF/	110586	3 months	tseaver	Conform to repository policy.
CMF.buildout/	114058	12 days	hannosch	Update p.r.zope2instance recipe version
CMF_Extras/	110587	3 months	tseaver	Conform to repository policy.
CMF_Hotfixes/	110588	3 months	tseaver	Use only tags.
ClientForm/	94305	18 months	shh	Fix instructions.
CompositePage/	113598	3 weeks	ctheune	Comply with repository policy.
DateTime/	114648	35 hours	hannosch	Bump version
DocumentTemplate/	113682	3 weeks	hannosch	Bump version
ExtensionClass/	113552	3 weeks	hannosch	Bump version
GenericSetup/	110589	3 months	tseaver	Conform to repository policy.
Grokstar/	112194	2 months	ulif	Comply to repository policy.
Interface/	110593	3 months	tseaver	Conform to repository policy.
Missing/	113534	3 weeks	hannosch	Bump version
MultiMapping/	110642	3 months	tseaver	Overlooked files.
Persistence/	113556	3 weeks	hannosch	Bump version
PluggableAuthService/	110647	3 months	tseaver	Comply with repository policy.
PluginRegistry/	110648	3 months	tseaver	Comply with repository policy.
Products.Ape/	113599	3 weeks	ctheune	Comply with repository policy.
Products.BTreeFolder2/	114598	2 days	hannosch	Bump version
Products.Basket/	113600	3 weeks	ctheune	Partially comply with repository policy.
Products.CMFActionIcons/	110708	3 months	tseaver	Whack-a-mole salad.
Products.CMFCalendar/	114573	2 days	hannosch	Backported c114569 from trunk
Products.CMFCore/	114575	2 days	hannosch	Forward ported c114574 from 2.2 branch
Products.CMFDefault/	114559	2 days	hannosch	Use the constructor directly instead of relying on OFSP
Products.CMFTopic/	114168	9 days	jens	- vb
Products.CMFUid/	114171	9 days	jens	- vb

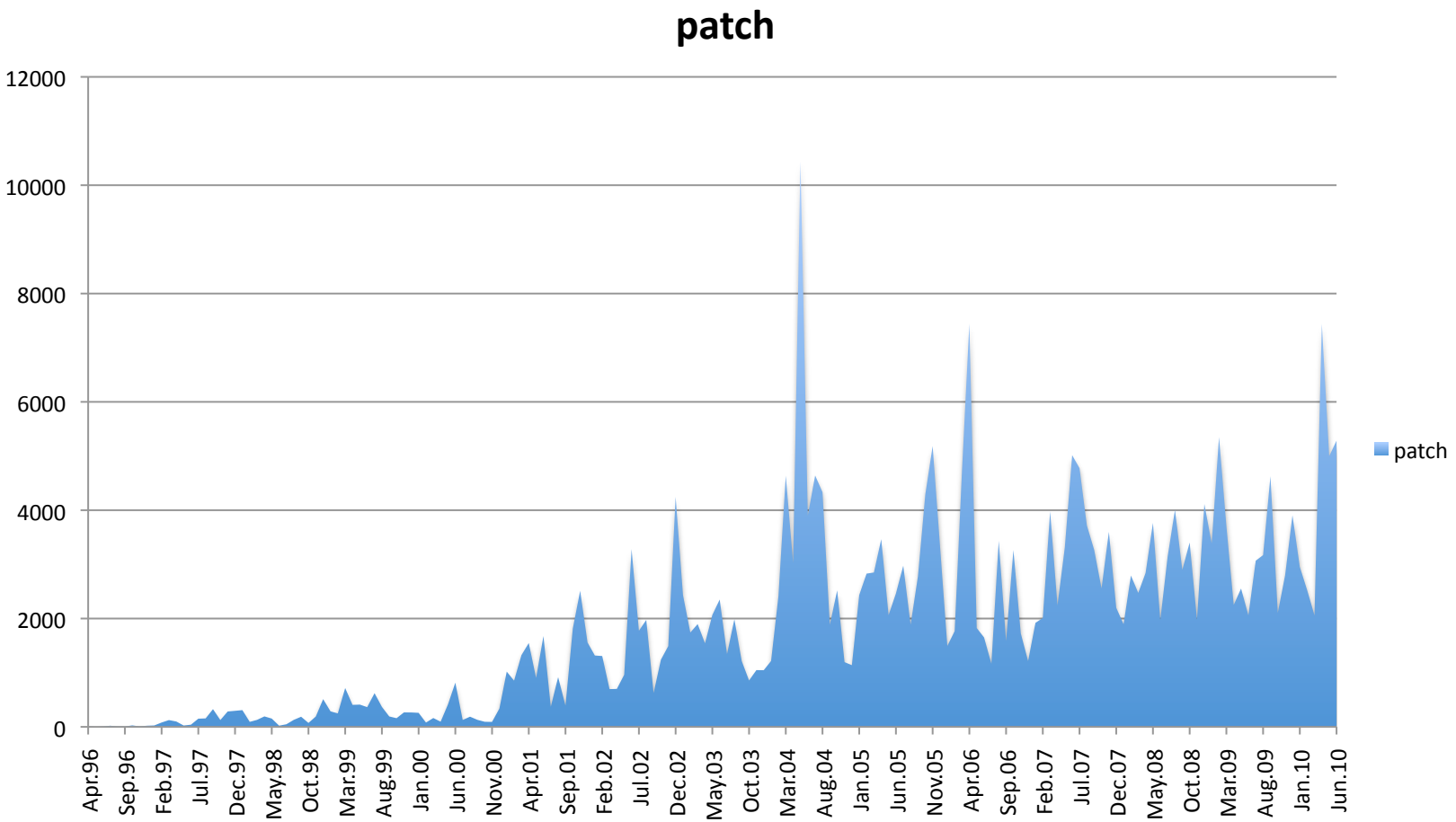
コミット数

svn.zope.org全体、1996/04 - 2010/06、月単位



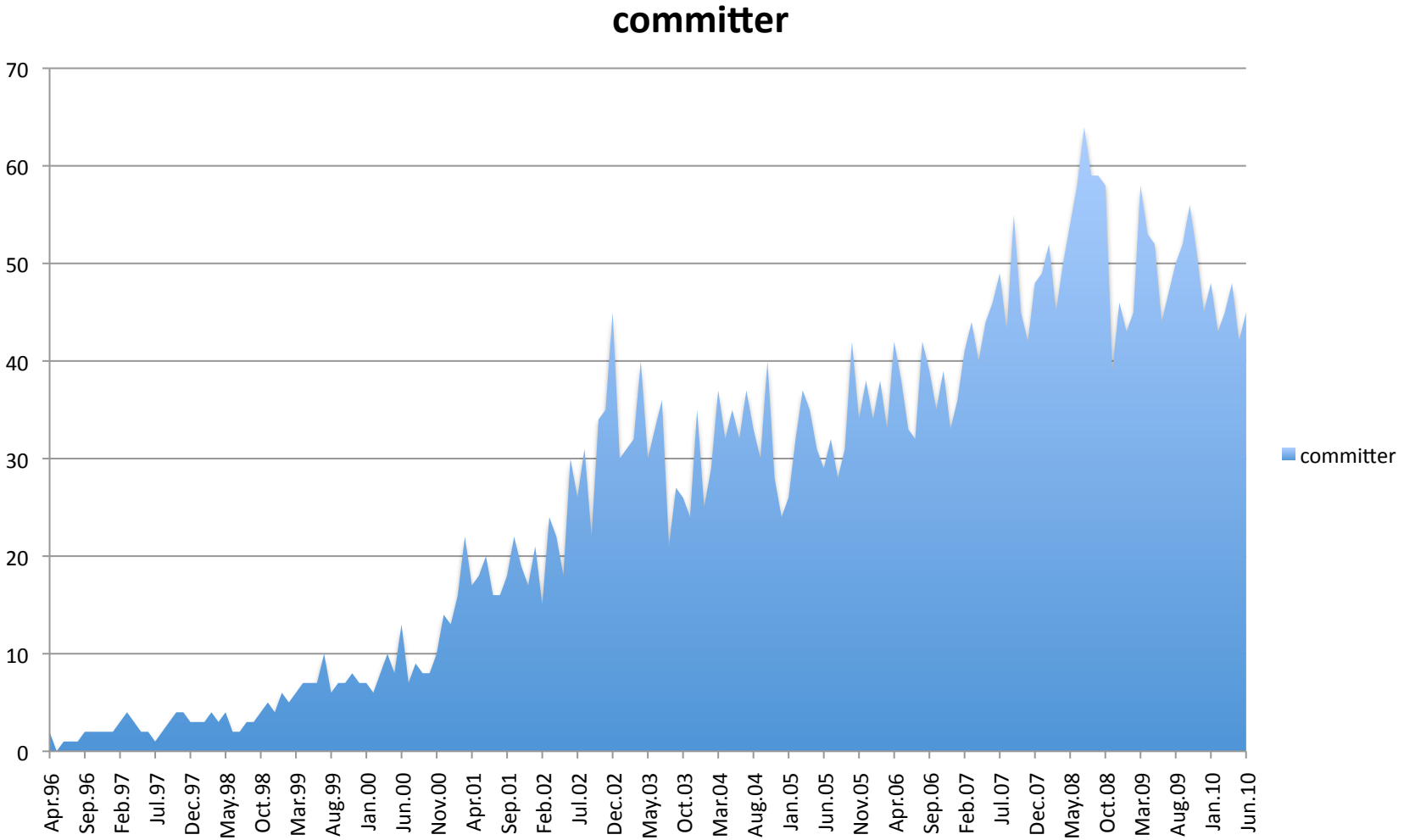
パッチ数

svn.zope.org全体、1996/04 - 2010/06、月単位



コミッター数

svn.zope.org全体、1996/04 - 2010/06、月単位



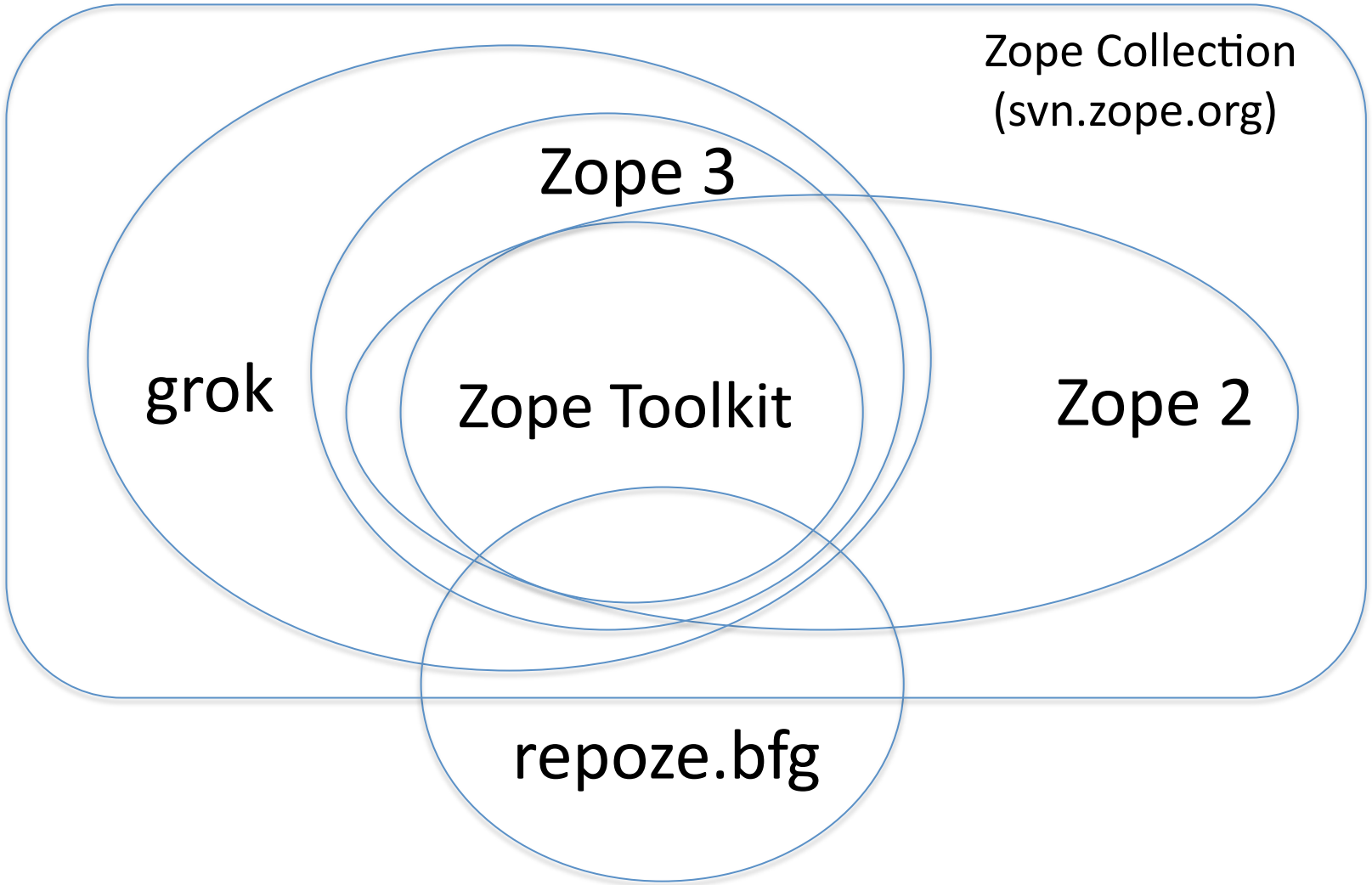
まとめ

1. 開発系のML流量はさほど変わらない
2. コードリポジトリへのコミット数、パッチ数、コミッター数は継続して増加
3. 毎日約20通の議論メール、30件のコミット、150件のコードへの操作（パッチ）は、十分に活発といえる

1. Zopeって、使われてるの？

a. あまりニュースを見ないけど、まだあるの？
   a. 広告宣伝費をまったくかけていない
   b. Zopeは開発者にとって魅力的だが、Web利用者は興味を持たない
b. 使っている人はいるの？
   a. たくさんいます
   b. 多くの人はZopeを使っていることに気がついていません
c. これから使って大丈夫？そのうち無くならない？
   a. もちろん大丈夫
   b. この１２年を経て、利用が拡大している

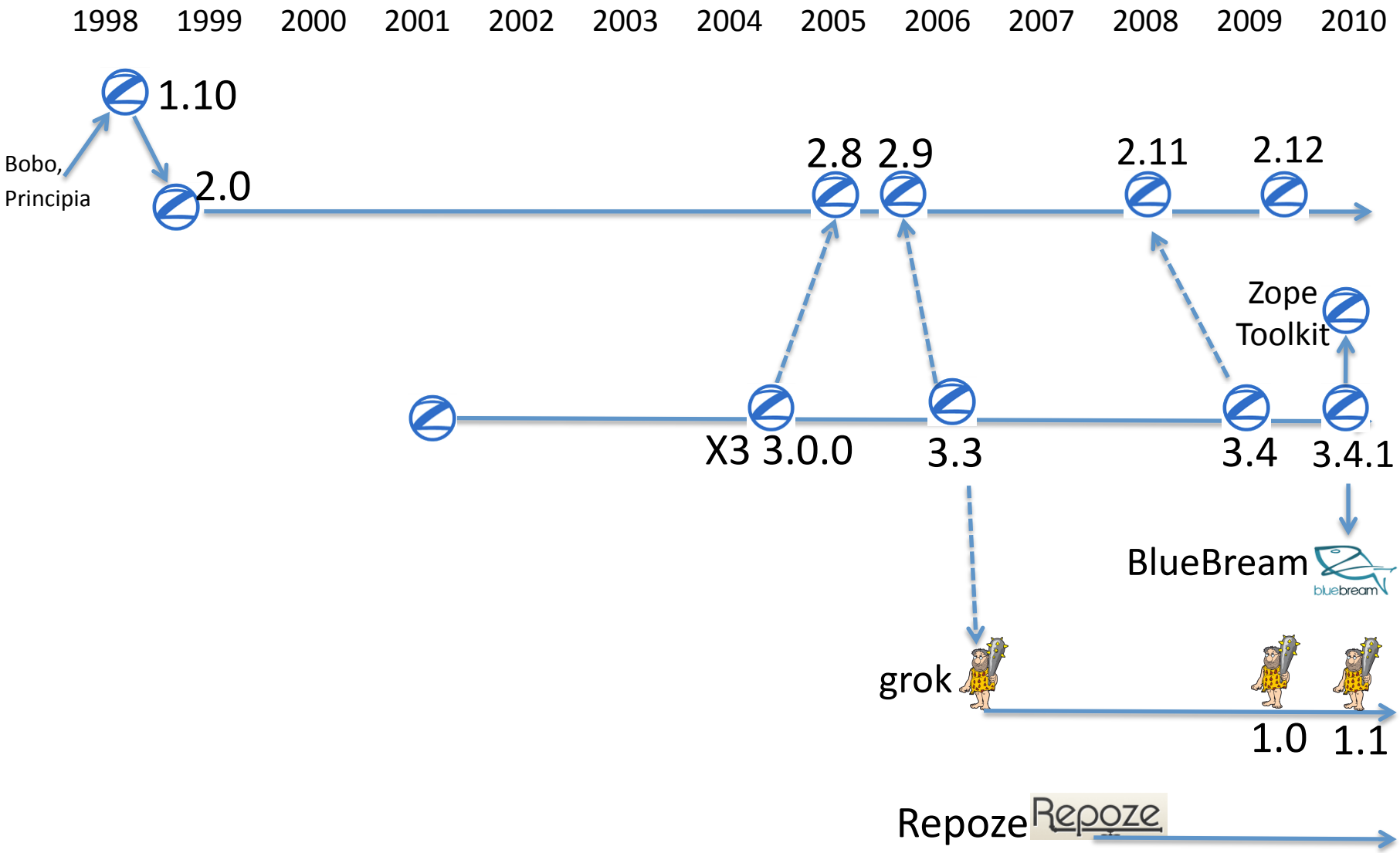
Zopeコード利用の拡大



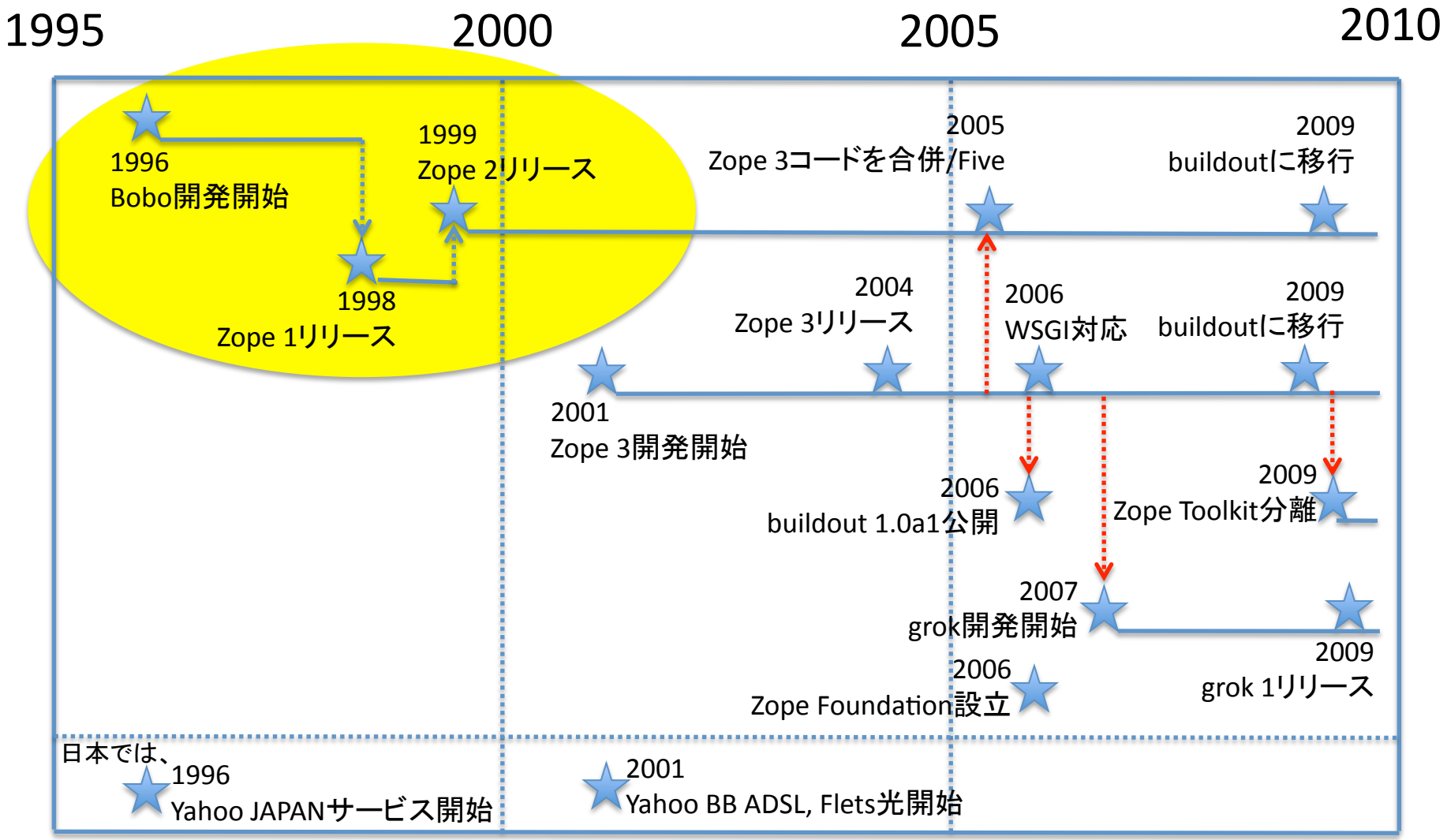
Zopeって、何がすごいの？

他と何が違うの？

Zopeの系譜


1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
1.10
Bobo,
Principia
2.0
2.8
2.9
2.11
2.12
Zope
Toolkit
X3
3.0.0
3.3
3.4
3.4.1
BlueBream
bluebream
grok
1.0
1.1
Repoze
Repoze

最初の波

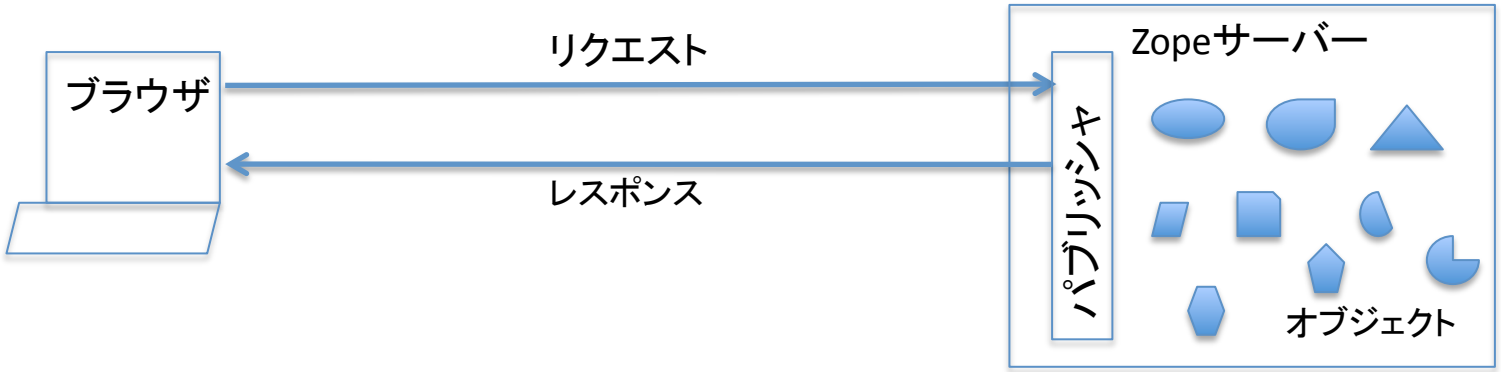

1995
2000
2005
2010
1996
Bobo開発開始
1998
Zope 1リリース
1999
Zope 2リリース
2005
Zope 3コードを合併/Five
2009
buildoutに移行
2004
Zope 3リリース
2006
WSGI対応
2009
buildoutに移行
2001
Zope 3開発開始
2006
buildout 1.0a1公開
2009
Zope Toolkit分離
2007
grok開発開始
2009
grok 1リリース
2006
Zope Foundation設立
日本では、
1996
Yahoo JAPANサービス開始
2001
Yahoo BB ADSL, Flets光開始

Zopeの革新

1998年 Zope登場

- オブジェクトパブリッシング
- トラバーシング
- オブジェクトデータベース
- オールインワン
- オープンソース化

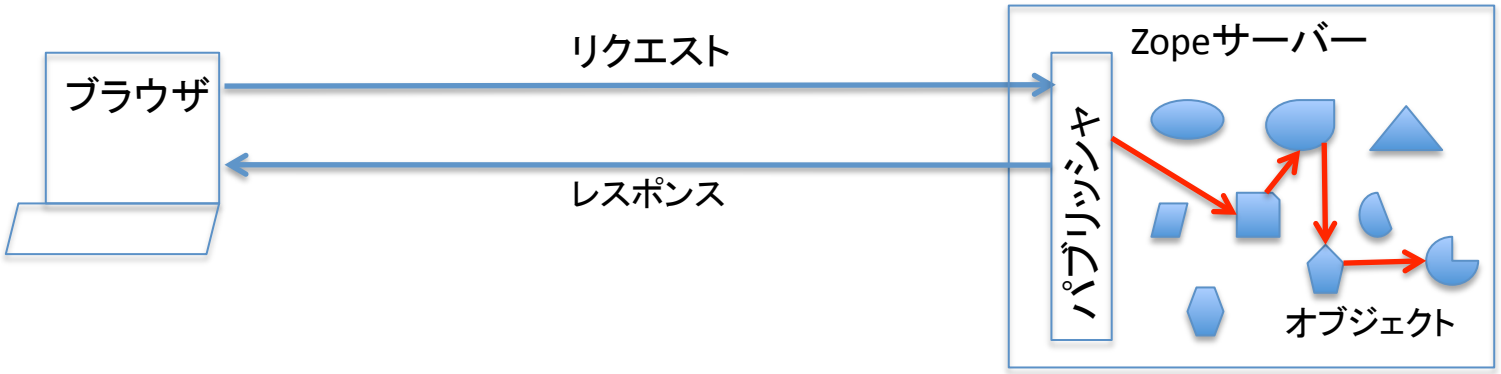
オブジェクトパブリッシング



これらのオブジェクトは、それぞれHTMLテキストや
画像データ、ブラウザで実行されるJavaScriptコード、
サーバーで実行されるPythonコードなどを含む

ブラウザからhttpリクエストが来たときに、
ファイルリソースを探してその中身を返すのではなく、
オブジェクトと呼ばれる抽象的な情報構造体を探して、
そのオブジェクトをコールして返されたものを返す。

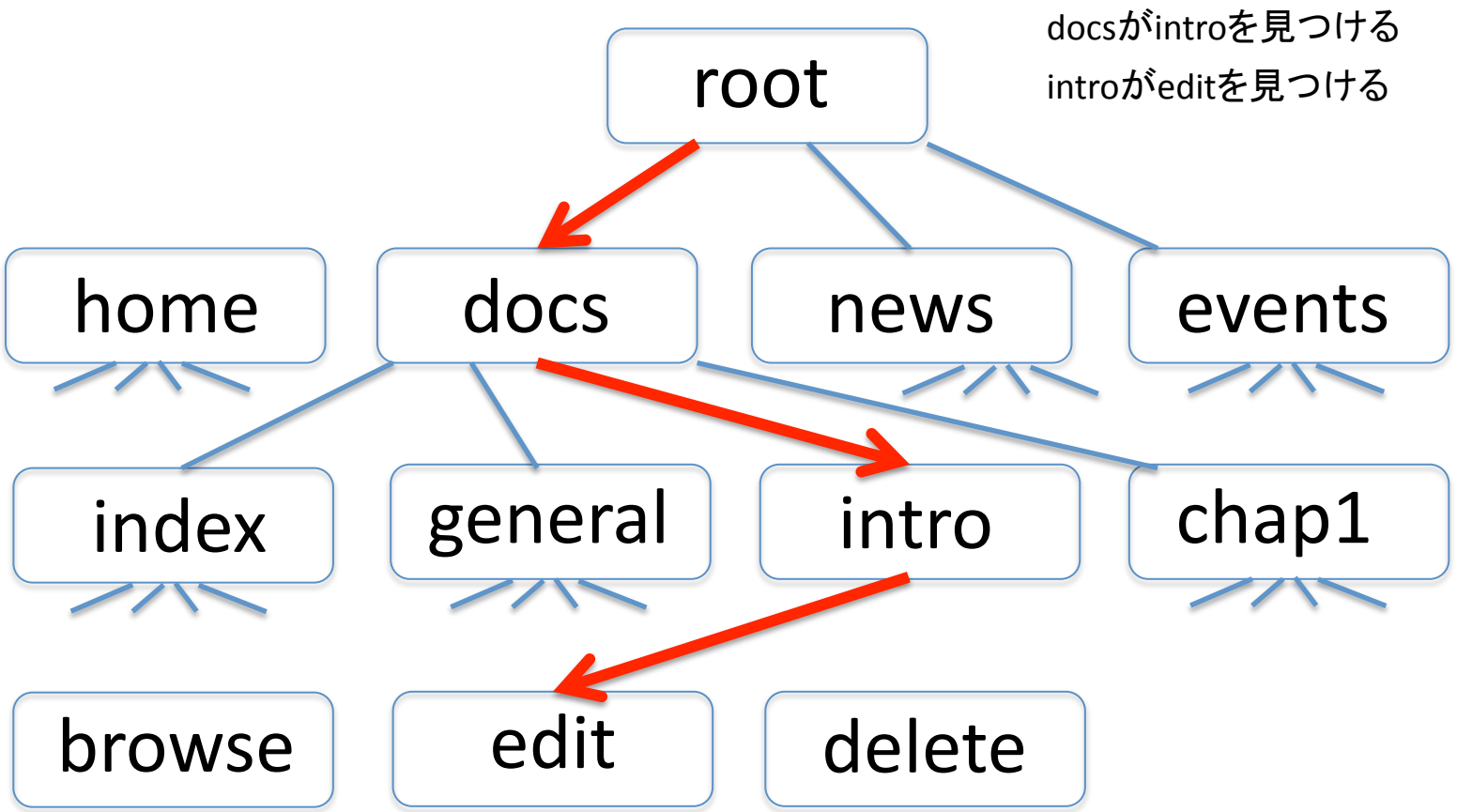
トラバーシング



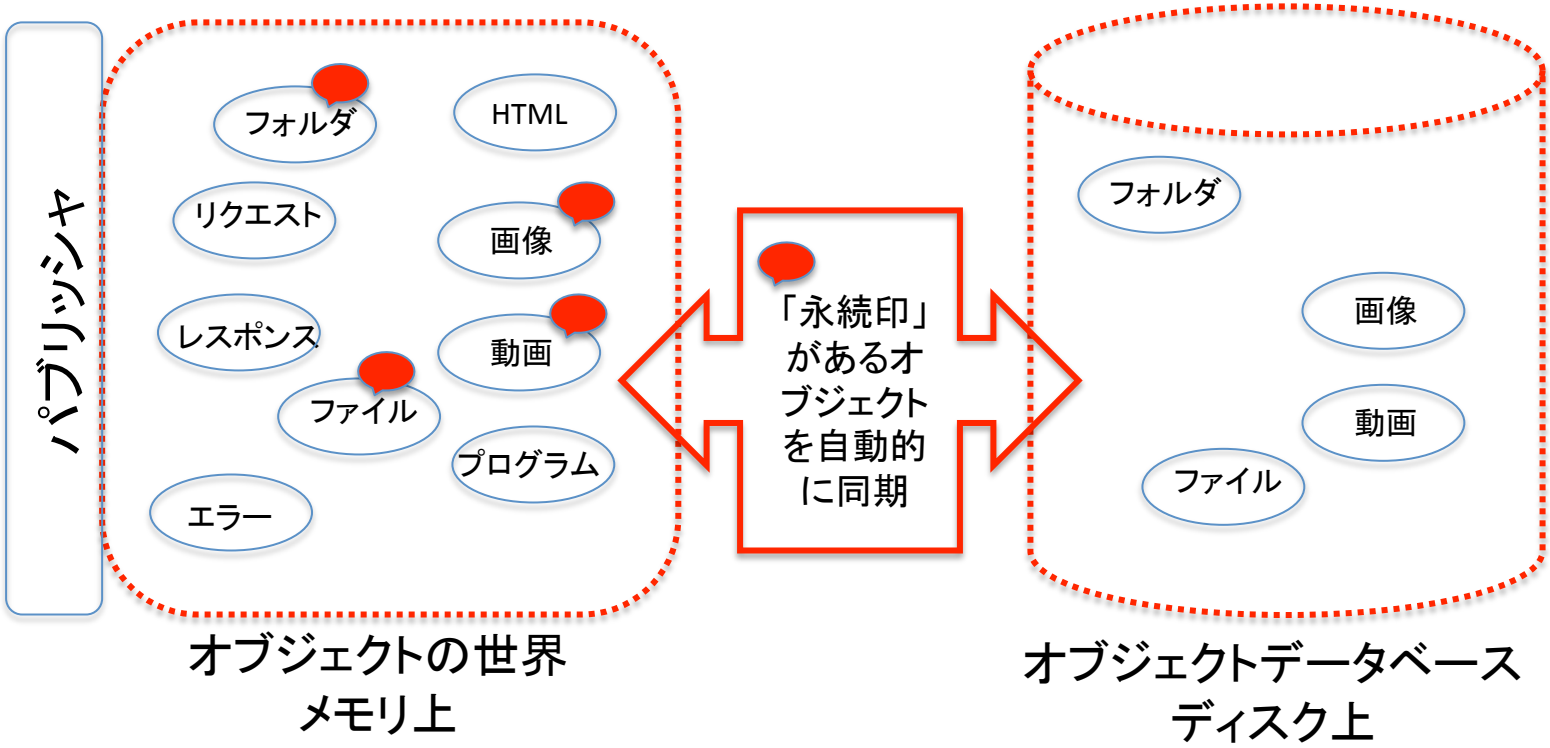
URLを受け取ると、
それを要素ごとに分解し、
それぞれの要素に対応するオブジェクトに次の要素に対応するオブジェクトを見つけてもらう。

トラバース

http://zope.jp/doc/zope/intro/edit

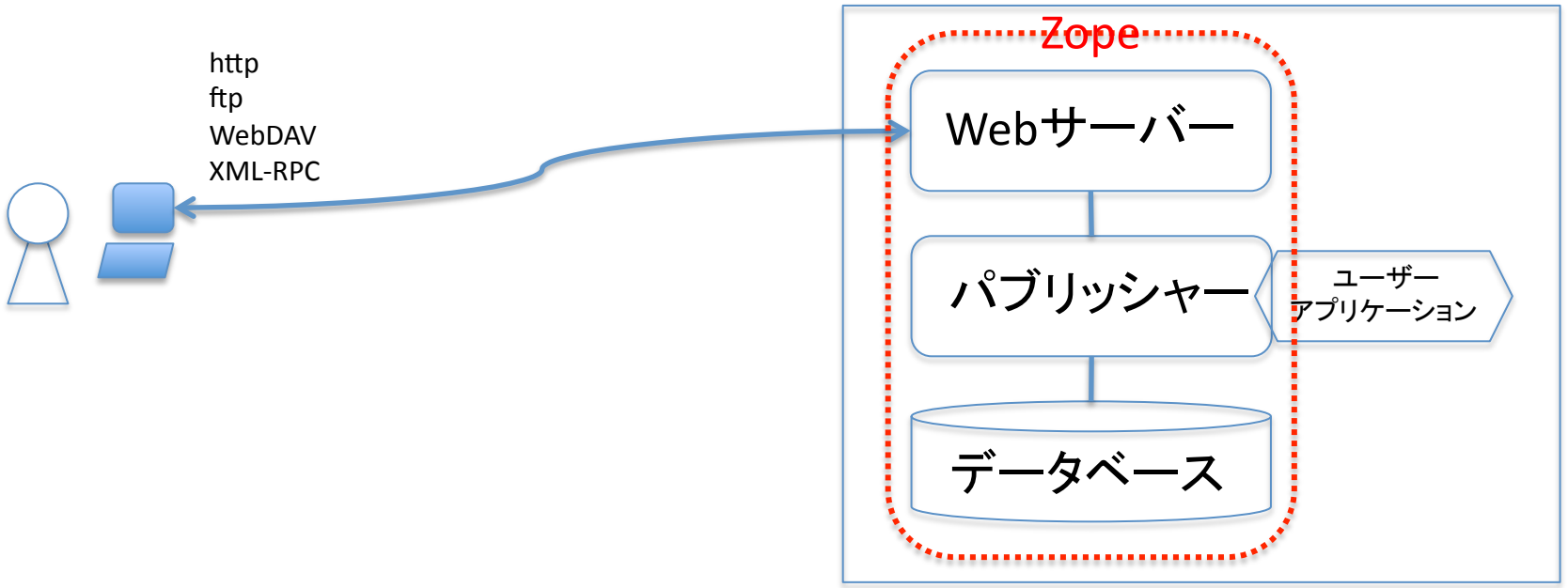


オブジェクトデータベース



永続クラスを継承することで、自動的にオブジェクトデータベース上に書き込まれるようになる。

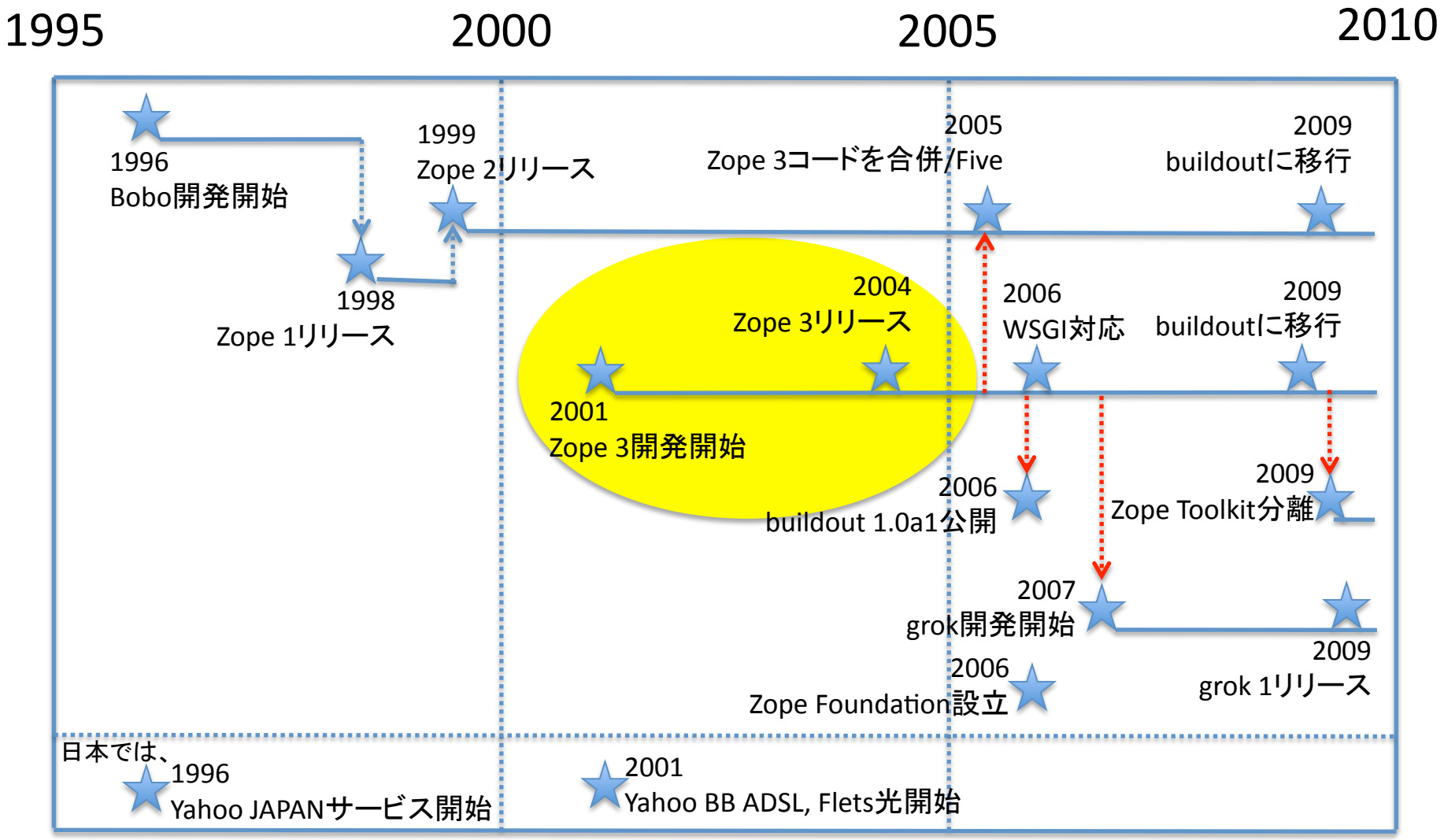
オールインワン



Webサーバー、パブリッシャー、データベースの
すべてが一緒に配布される。

これだけでWebアプリケーションが構築できる

第二の波


1995
2000
2005
2010
1996
Bobo開発開始
1998
Zope 1リリース
1999
Zope 2リリース
2005
Zope 3コードを合併/Five
2009
buildoutに移行
2004
Zope 3リリース
2001
Zope 3開発開始
2006
WSGI対応
2009
buildoutに移行
2006
buildout 1.0a1公開
2009
Zope Toolkit分離
2007
grok開発開始
2009
grok 1リリース
2006
Zope Foundation設立
日本では、
1996
Yahoo JAPANサービス開始
2001
Yahoo BB ADSL, Flets光開始

プログラムの再利用性

一度書いたら、何度も使う

コピペして、改造するのは再利用とは言えない。
　改造のために理解する手間、改造する手間、テストする手間がかかる
　不具合の時に元プログラムが悪いのか改造が悪いのか不明瞭になる

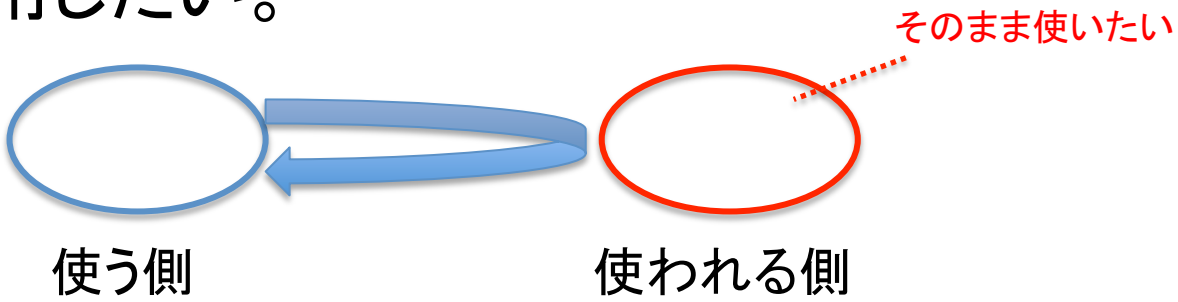
元のプログラムはいっさい変更しないで再利用する

コンポーネントアーキテクチャによる実装

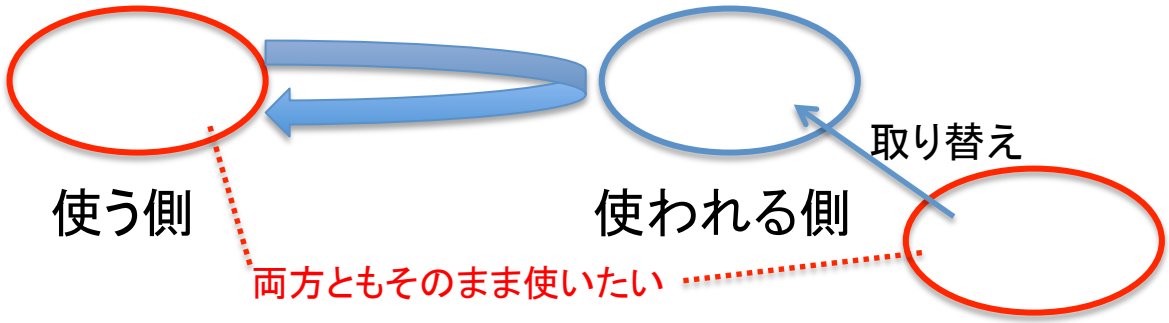
Zope 2を改造して実現するのは不可能

再利用って？

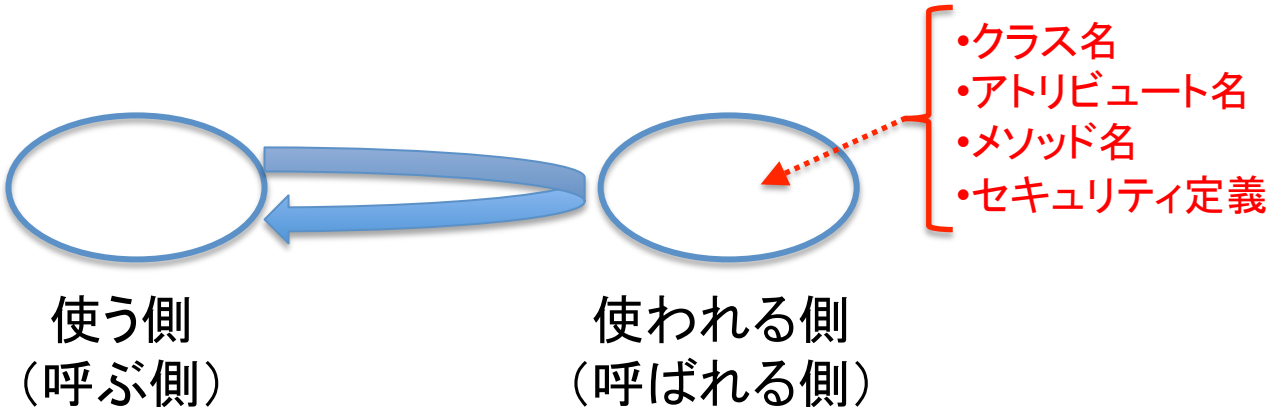
1. 新しいアプリケーションで既存プログラムを部品として再利用したい。



2. 既存アプリケーションで機能の一部分を別のプログラムで置換えたい。



再利用するのに何が問題？



コードの中に名前が入っているので、名前が違うとコードも変更しなくてはならない

コードの中にセキュリティ定義が入っているので、セキュリティが違うとコードも変更しなくてはならない。

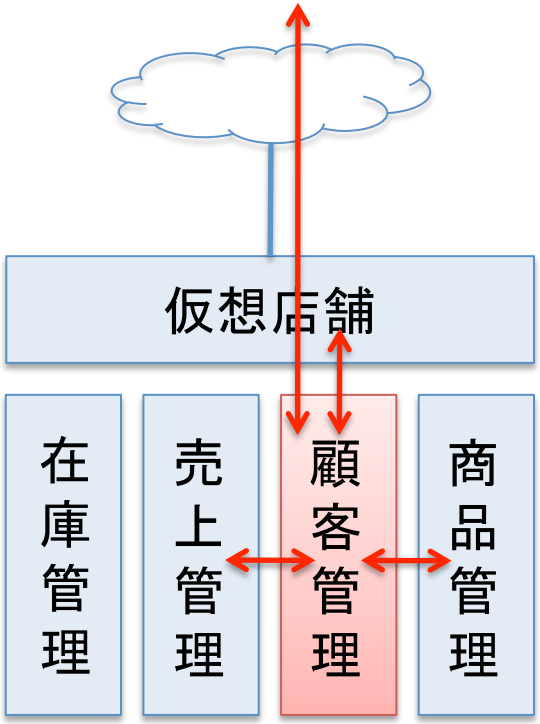
既存プログラム再利用の問題

他人が異なる目的で作った既存プログラムを利用するには、そのプログラムを改変する必要があった。

- APIの呼び方が違う
- 名前が違う
- 機能が少し違う
- セキュリティの設定が違う

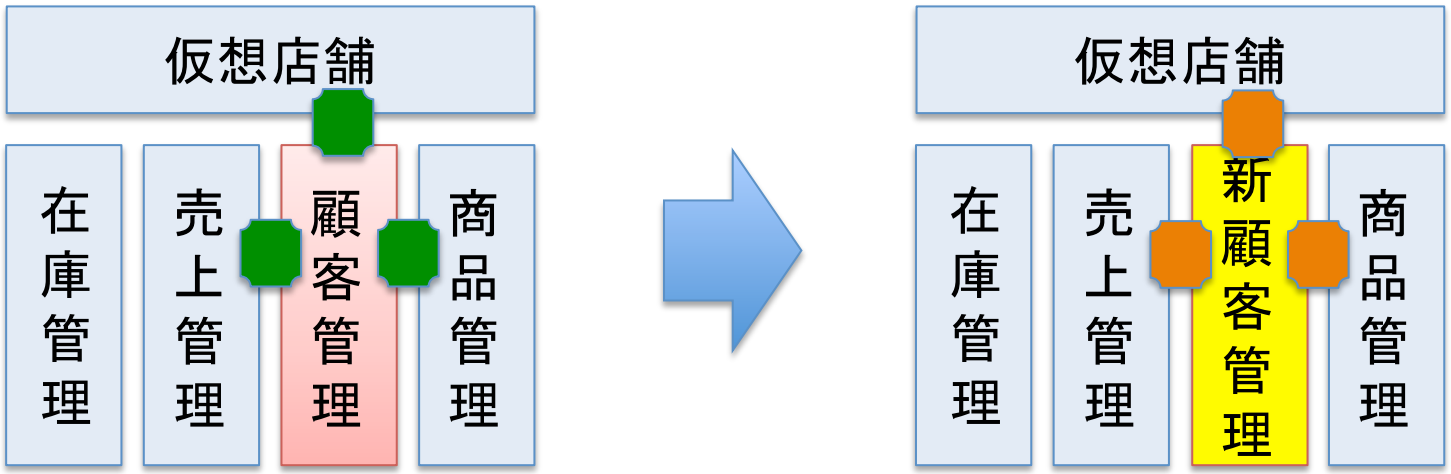
たとえば、

今ある仮想店舗の顧客管理の部分を、別の既存製品に取り換えたい場合：



- URLの要素名が違う
- APIの名前が違う
- モジュール名が違う
- 機能が少し違う
- パーミッション（セキュリティ設定）が違う

コンポーネントの交換可能性



青い部分も、黄色い部分も、いっさいコードを変更しないで交換可能か？

コンポーネント
アーキテクチャ

Componet Architecture

コンポーネントアーキテクチャ

既存プログラムを「部品」として再利用するための技術体系

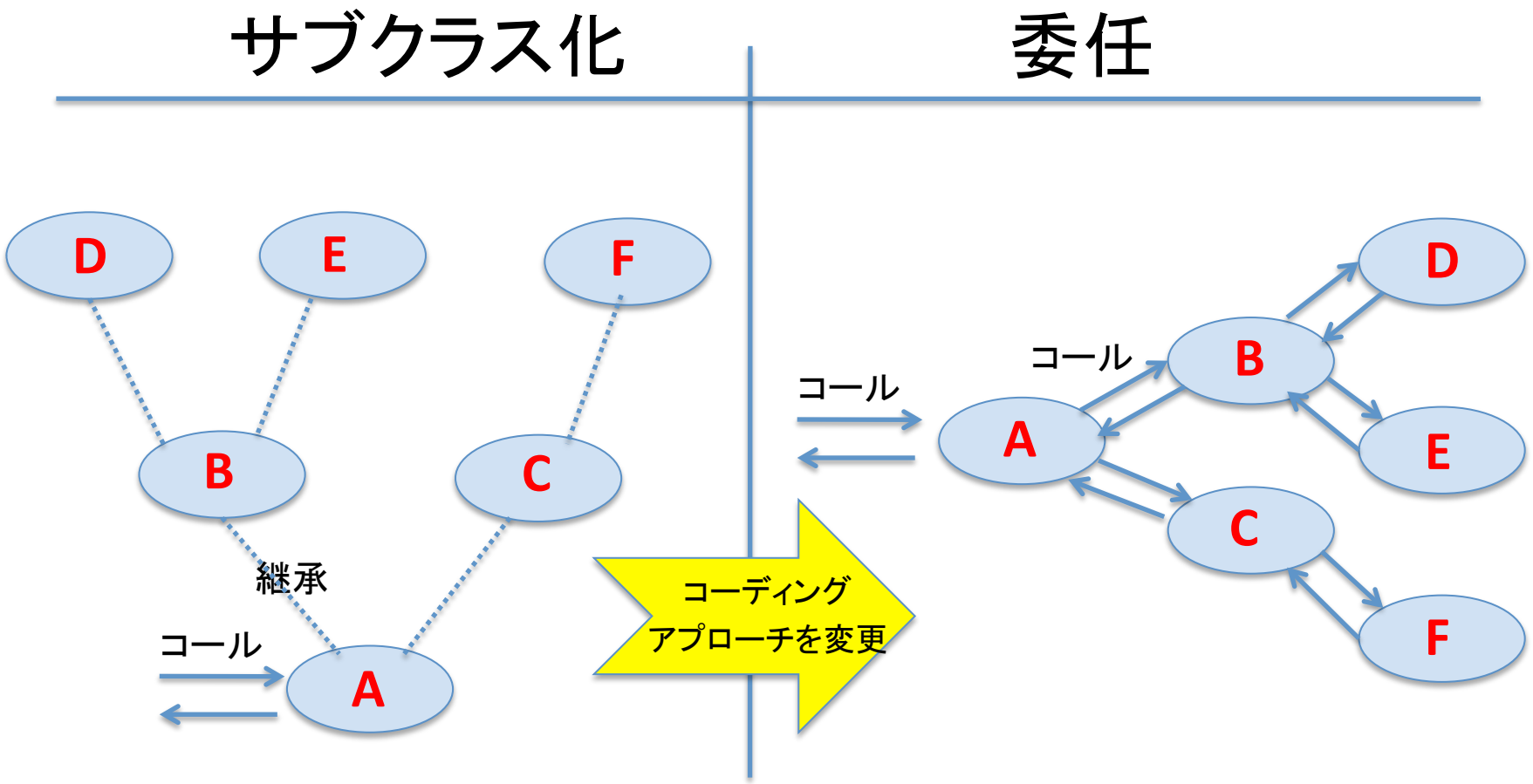
- サブクラス化から委任へ
- インタフェース
- コンポーネントレジストリ
- ZCML
- アダプタ

コンポーネントアーキテクチャ

プログラムの再利用化のための考え方の体系

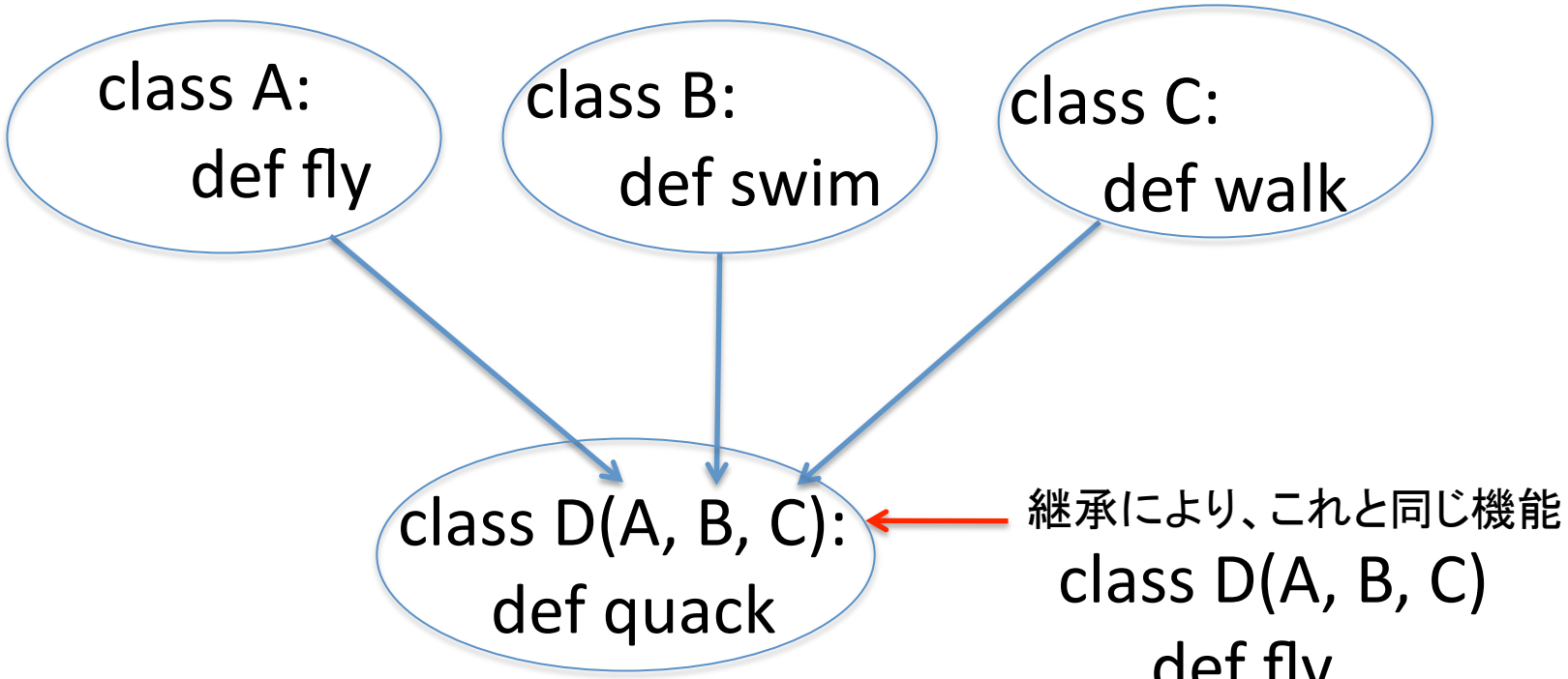
- サブクラス化よりも、委任を使う
- 再利用の単位として、コンポーネントというものを考える。
- コンポーネントのAPI定義を書いたインタフェースを作り、コンポーネントと結びつける。
- コンポーネントをインタフェースなどの特徴情報とともにコンポーネントレジストリに登録する。
- コンポーネントは、コンポーネントレジストリを通してインタフェースなどの特徴情報を基にして探し出される。
- 自分で書く部分と流用する既存コンポーネントの間で名前や機能が違う場合、アダプタで吸収する。
- セキュリティ定義をコンポーネントから取り除き、XML形式のZCMLで定義する。各パッケージのZCMLはアプリケーション側のZCMLでオーバーライドできる。

サブクラス化から委任へ

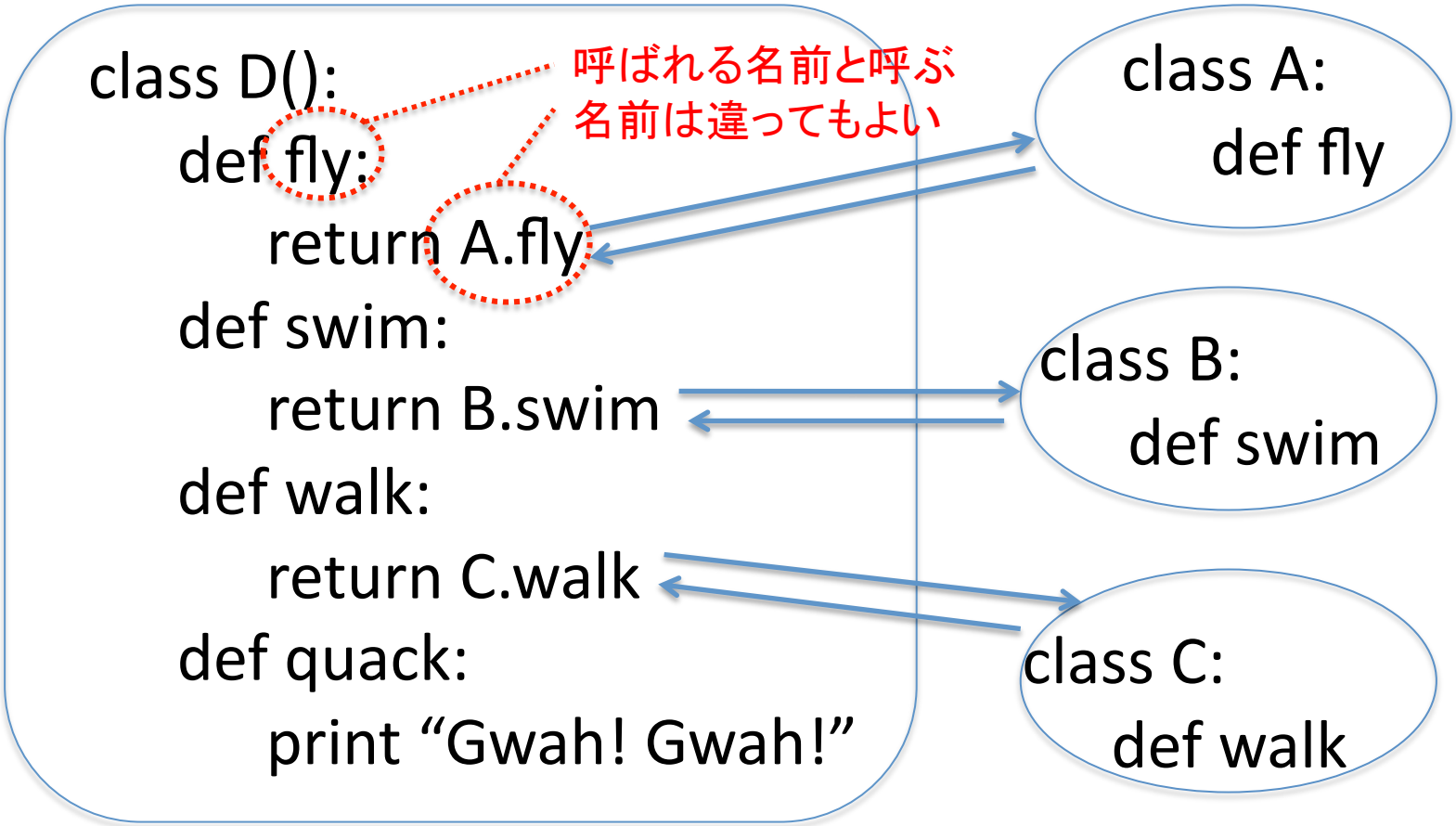


Aは、B, C, D, E, Fの機能を持つ

サブクラス化


class A:
def fly
class B:
def swim
class C:
def walk
class D(A, B, C):
def quack
継承により、これと同じ機能
class D(A, B, C)
def fly
def swim
def walk
def quack

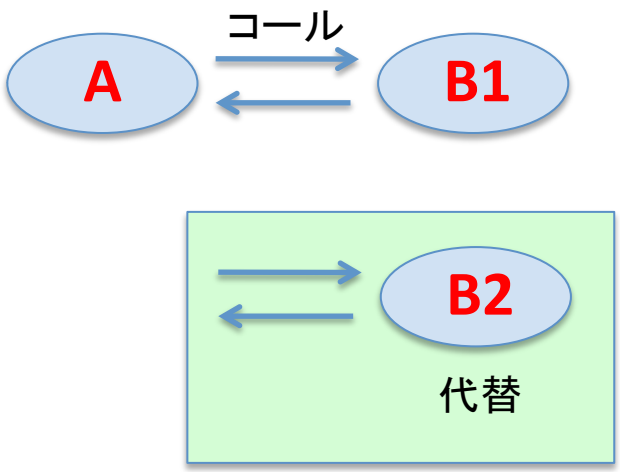
委任


class D():
def fly:
return A.fly
def swim:
return B.swim
def walk:
return C.walk
def quack:
print “Gwah! Gwah!”
呼ばれる名前と呼ぶ
名前は違ってもよい
class A:
def fly
class B:
def swim
class C:
def walk

インタフェース

Interface Object

インタフェース(API定義)

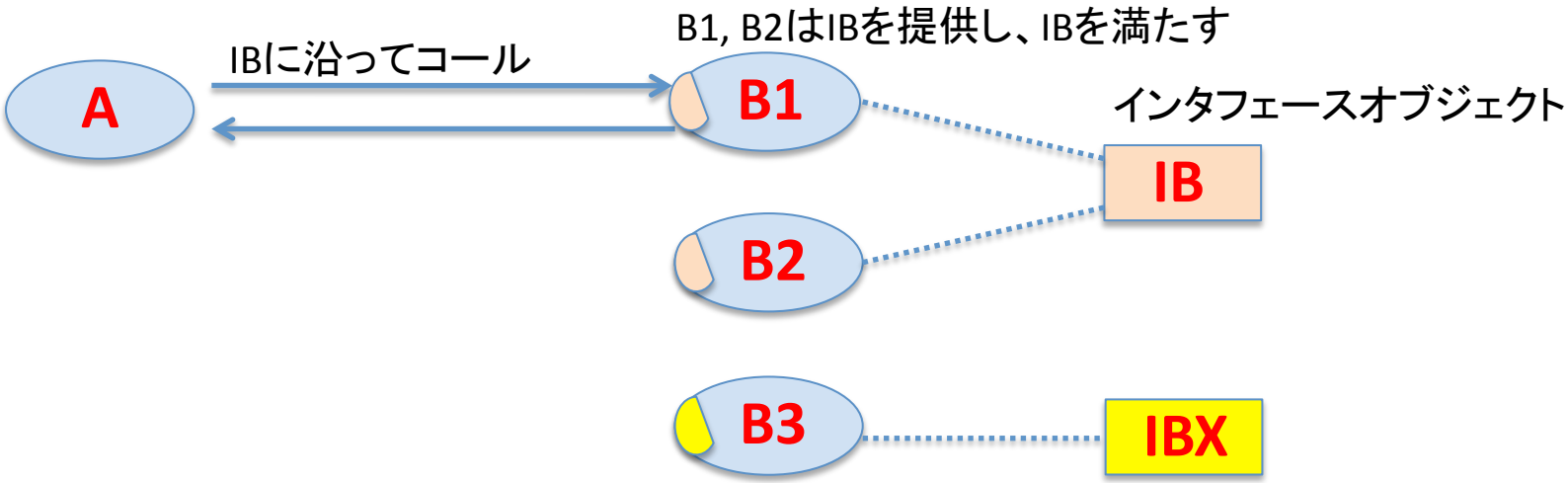


使用中のB1コンポーネントをB2コンポーネントに取り換えたい。

どうやって、B1とB1が交換可能かどうかを見分けるか？

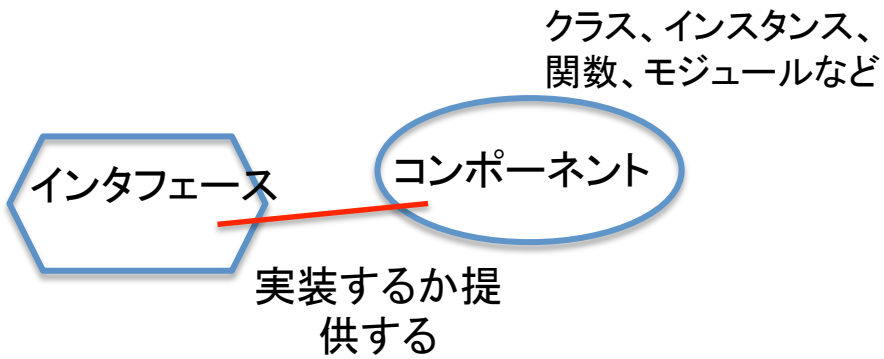
インタフェース(API定義)

コンポーネントの種類を識別するために、
インタフェースを作り、関連付ける。



呼ばれる側が同じインタフェースを提供し、それ
を満たす限り、コンポーネントとして交換可能。

インタフェース



- インタフェースはコンポーネントのAPIを記述した新しいタイプのオブジェクト
- 記述方法はclass文, def文, =を使を使うので、見かけはクラス定義に似ている
- でも、”Interface”を拡張するので、できるものは、インタフェースオブジェクト
- 書かれたものは実行できないし、アトリビュート参照できない。
- 名前は必ず”I”（大文字のアイ）で始まる
- インタフェースを実装したり、提供したりするものがコンポーネント

インタフェースのコード

- コード例

頭文字がI（アイ）

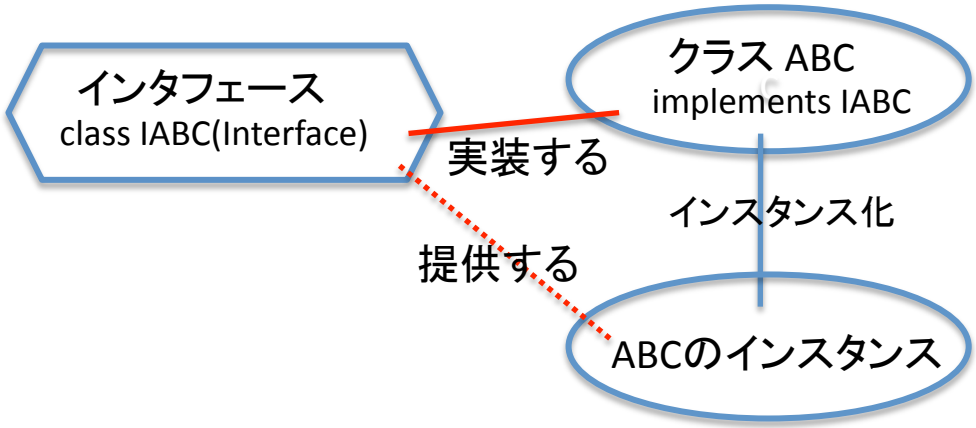
“Interface”を拡張

```
class ILocaleInheritance(Interface):
    """Locale inheritance support.

    Locale-related objects implementing this interface are able to ask for its
    inherited self. For example, 'en_US.dates.monthNames' can call on itself
    'getInheritedSelf()' and get the value for 'en.dates.monthNames'.
    """

    __parent__ = Attribute("The parent in the location hierarchy")

    __name__ = TextLine(
        title = u"The name within the parent",
        description=u"""The parent can be traversed with this name to get
                       the object.""")

    def getInheritedSelf():
        """Return itself but in the next higher up Locale."""
```

インタフェース(API)の記述

インタフェース

- コンポーネントとの関係




```
class Inheritance(object):
    """A simple base version of locale inheritance.

    This object contains some shared code amongst the various
    'ILocaleInheritance' implementations.
    """

    implements(ILocaleInheritance)

    # See zope.i18n.interfaces.locales.ILocaleInheritance
    __parent__ = None

    # See zope.i18n.interfaces.locales.ILocaleInheritance
    __name__ = None

    def getInheritedSelf(self):
        """See zope.i18n.interfaces.locales.ILocaleInheritance"""
        if self.__parent__ is None:
            raise NoParentException('No parent was specified.')
        parent = self.__parent__.getInheritedSelf()
        if isinstance(parent, dict):
            return parent[self.__name__]
        return getattr(parent, self.__name__)
```

ここで実装

インタフェースの役割

- APIの記述

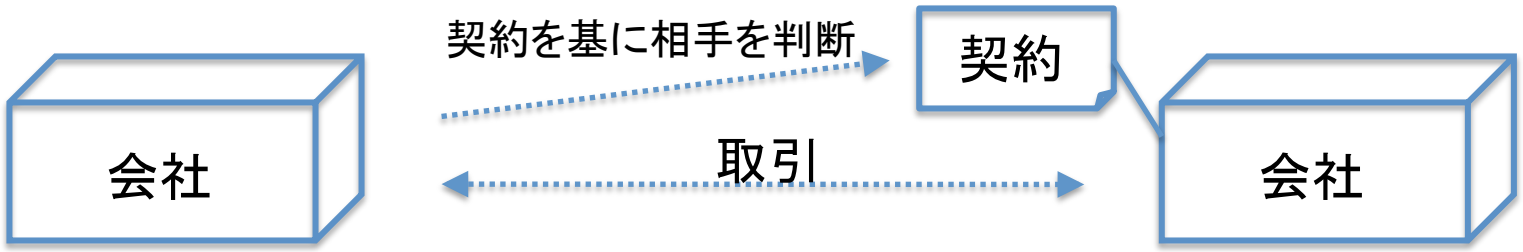
- コンテンツスキーマの記述
  - スキーマはインタフェースの一種

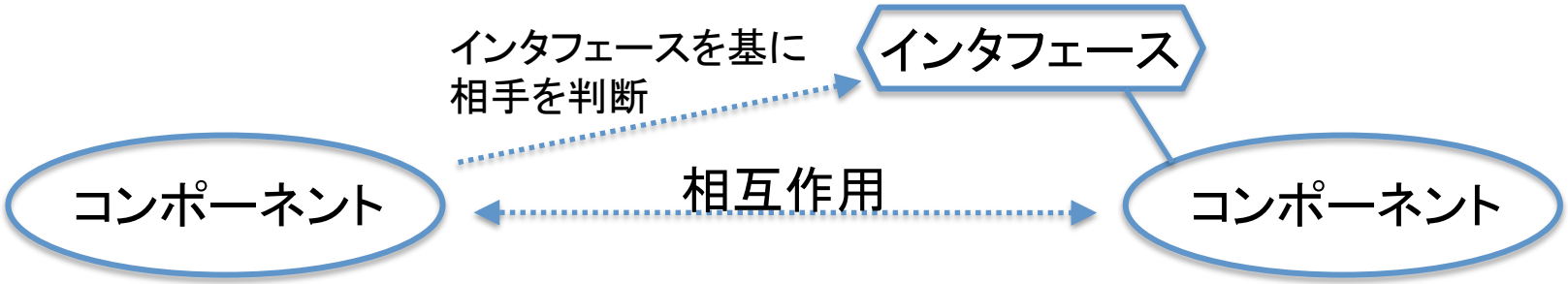
- マーカインタフェース（種類を識別する標識）
  - コンポーネントを、それが提供しているインタフェースで見分ける
  - 一つのコンポーネントは、複数のインタフェースを持てる
  - APIが何も書かれない抽象インタフェースもある

- ドキュメントとして
  - コンポーネントが何かを知るのに、これを見ればたいていは十分

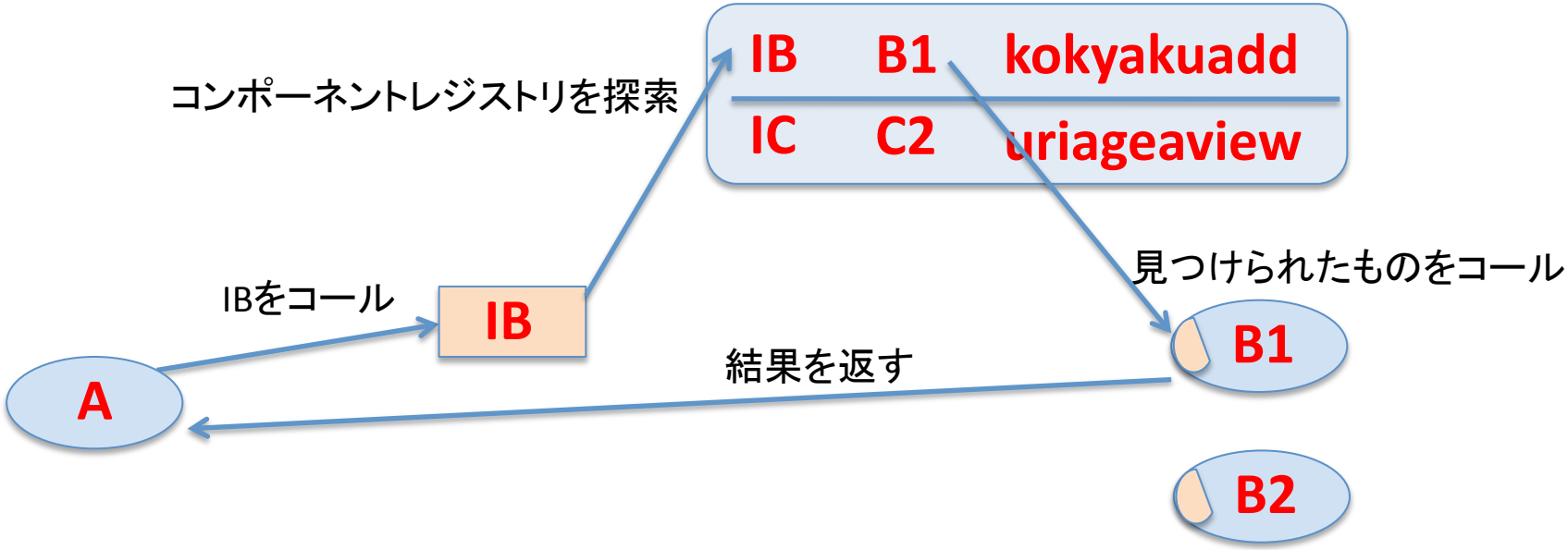
たとえて言うなら

•コンポーネントを会社に見立てれば、インタフェースは契約





コンポーネントレジストリ



- インタフェースをキーにしてコンポーネントが登録される。
- インタフェースをコールすると、レジストリに登録されたものがコールされる。
- レジストリ登録を変えると、別のコンポーネントがコールされる。
- 登録はZCMLで行われる。
- 必要に応じてURLの要素となる名前も登録される。

で、結局のところ

Zope 2では、相手を相手が決めた名前で呼ぶ

```
x = ABC()
```

Zope 3では、相手をインタフェースで呼ぶ

```
x = getUtilities(IABC)
```

レジストリにこのインタフェースで登録されたコンポーネントが返される

あるいは、コンフィグ(ZCML)で定義した名前で呼ぶ

```
x = getUtilities(IABC, name=u'myABC')
```

レジストリにこのインタフェースとコンポーネント名で登録されたコンポーネントが返される

アダプタ

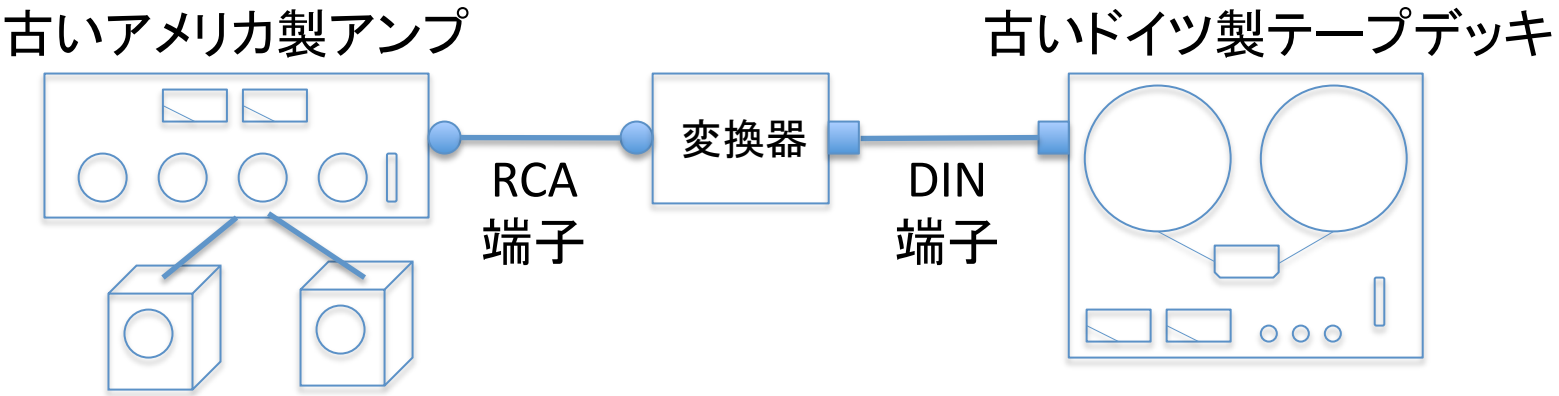
Adapter

アダプタ

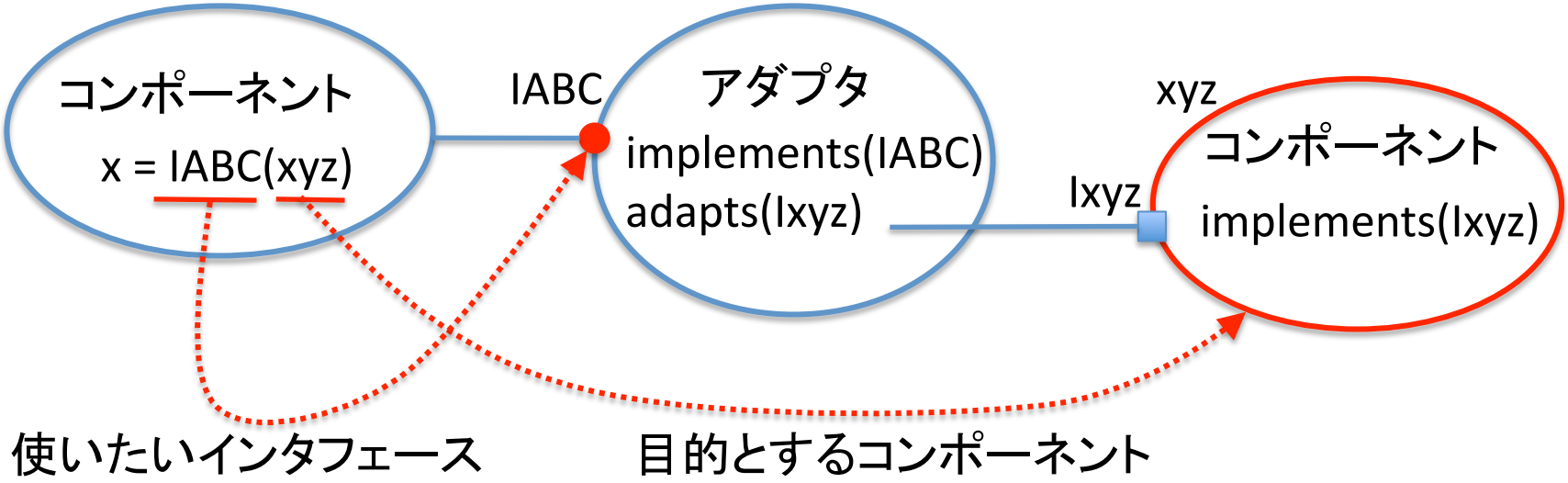
インタフェースの違いを吸収するための仲介コンポーネント

たとえるなら、

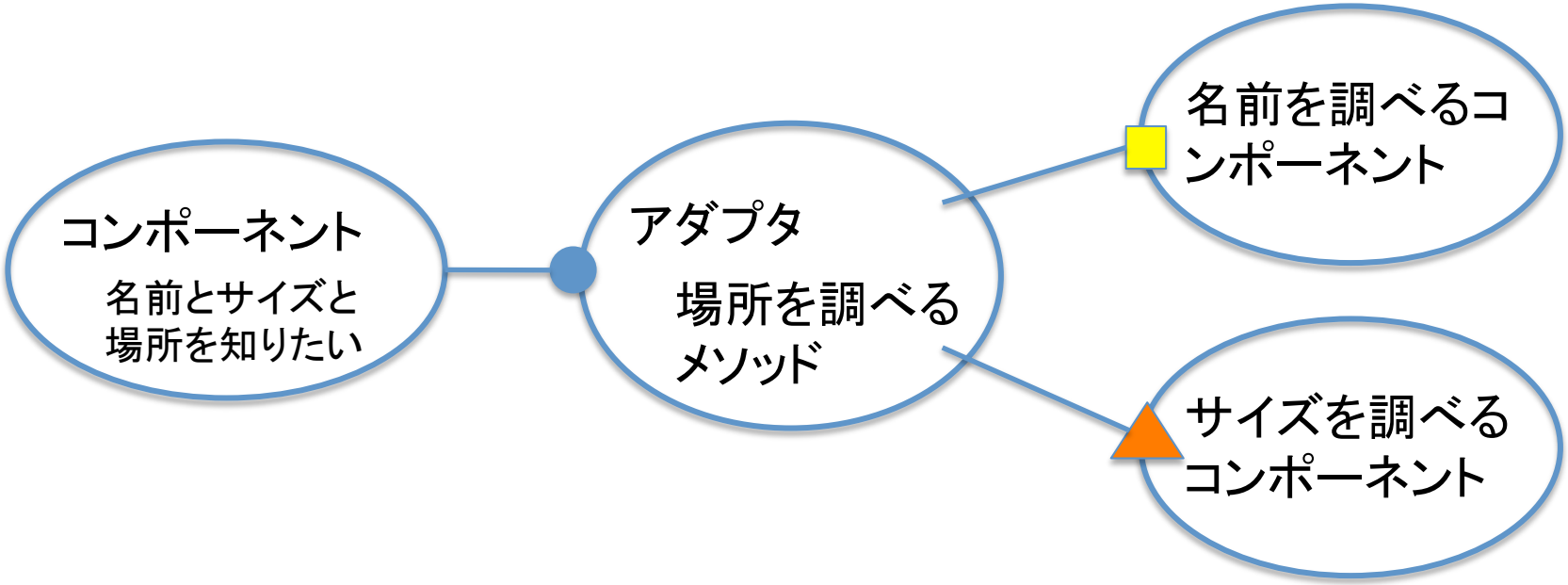
アダプタがインタフェースの違いを吸収



アダプタの使用


コンポーネント
x = IABC(xyz)
IABC
アダプタ
implements(IABC)
adapts(Ixyz)
xyz
Ixyz
コンポーネント
implements(Ixyz)
使いたいインタフェース
目的とするコンポーネント

たとえば



インタフェースの違いを吸収しながら、各種コンポーネントの機能を組み合わせて提供する

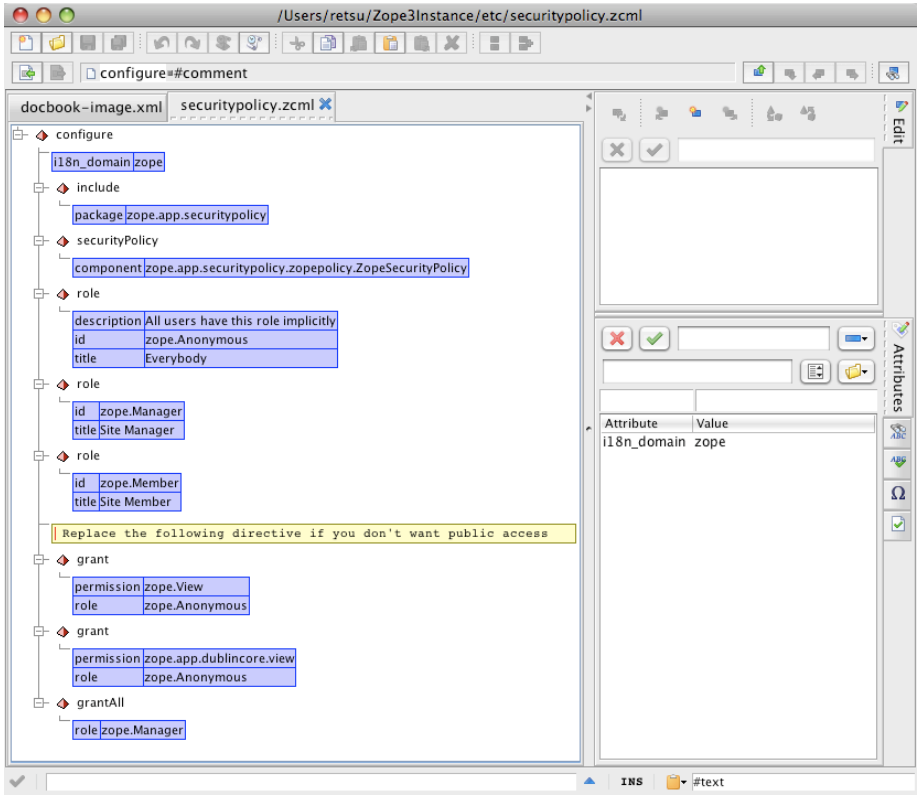
ZCML

Zope Configuration
Management Languate

ZCML

- XML形式のZope Configuration Management Language
  （Zopeコンフィグレーション管理言語）
- XMLなので、通常のXMLエディタや解析ツールが使える

Pythonを知らなくても、
コンフィグできる！



ZCMLですること

- インタフェースをキーにしてコンポーネントを登録
- URL要素に対応する名前を登録
- セキュリティ構成を定義
- メニュー構成を定義

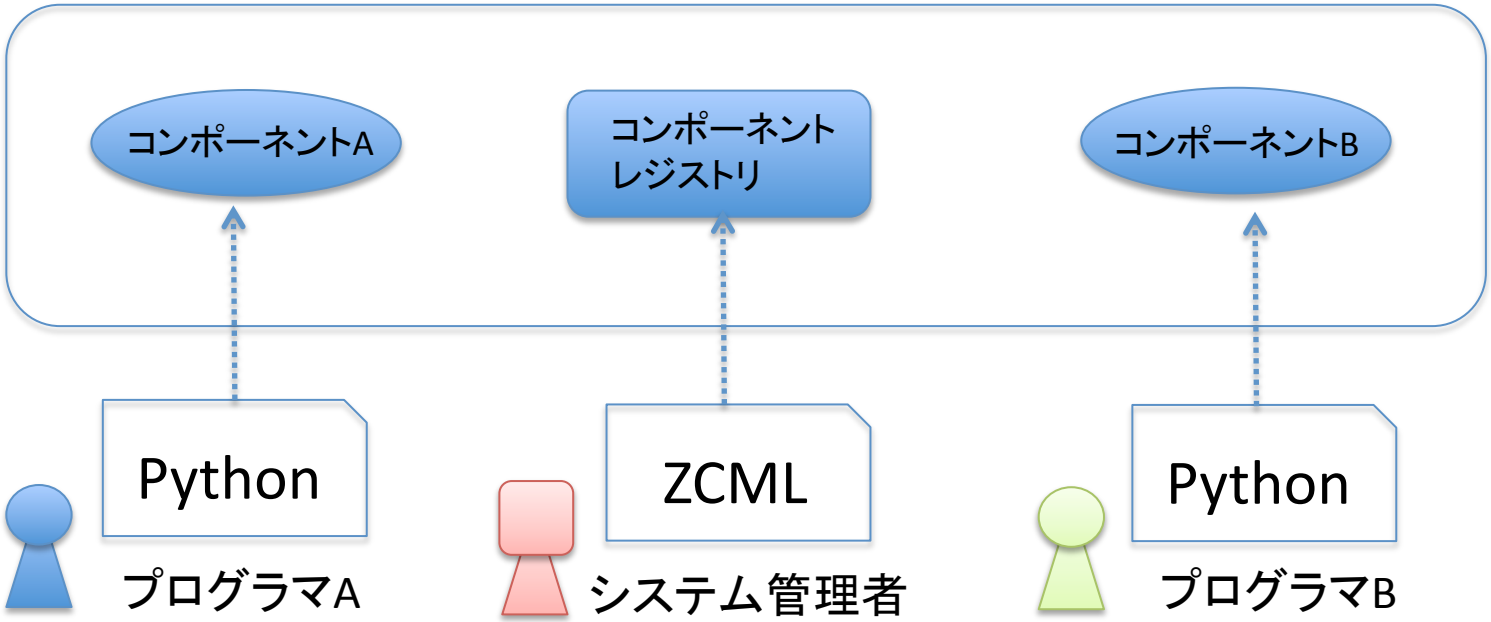
つまり、今までPythonコードで書いていた再利用時に変更する部分を、ZCMLで記述することによって外部化した。

そして、Pythonプログラマでなくても書ける。

ZCMLの意義

コンポーネントとコンポーネントをつなぎ合わせるプログラム言語

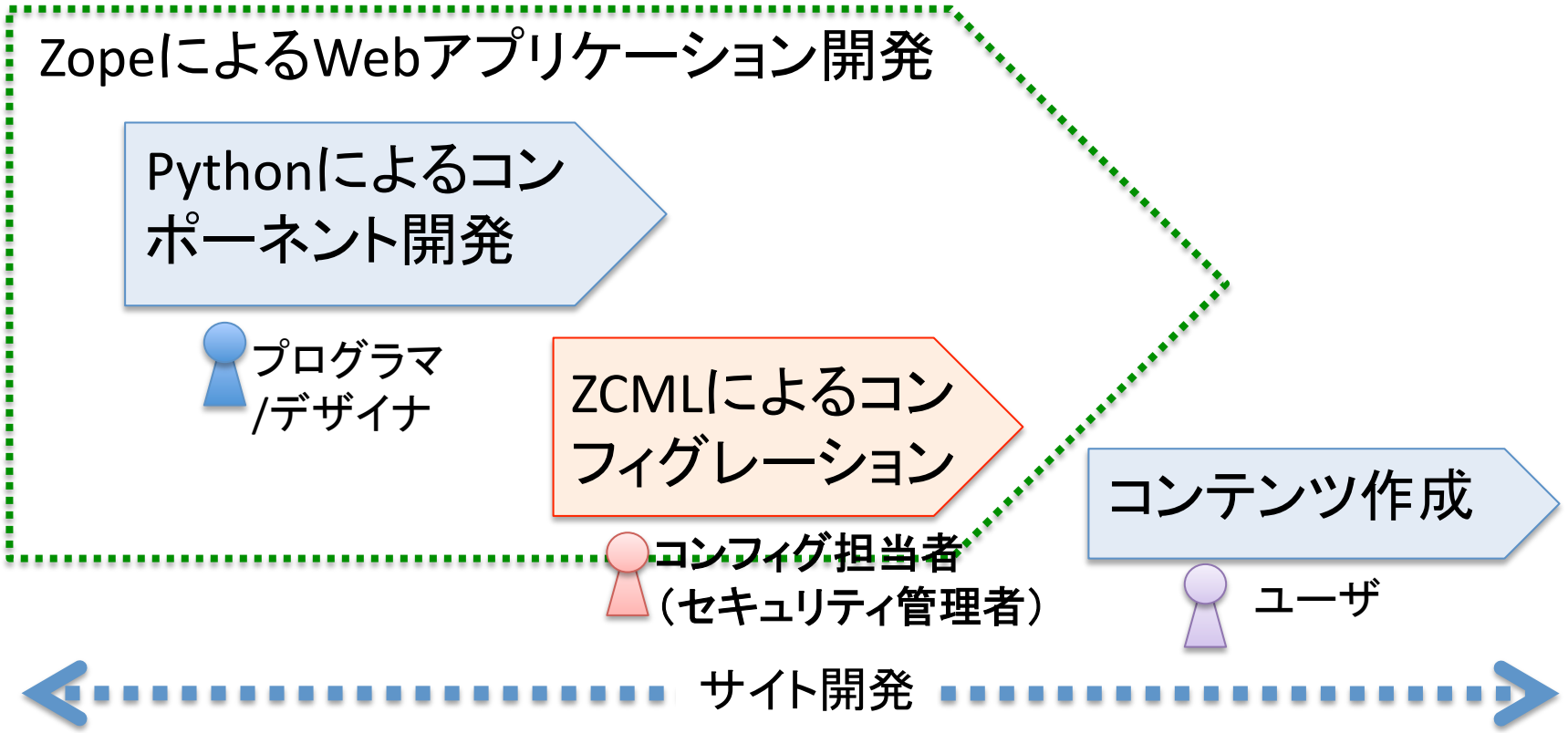
　　たとえば、どこかで別に開発され動いているPythonコンポーネントを自分のWebアプリケーションに組み合わせる。



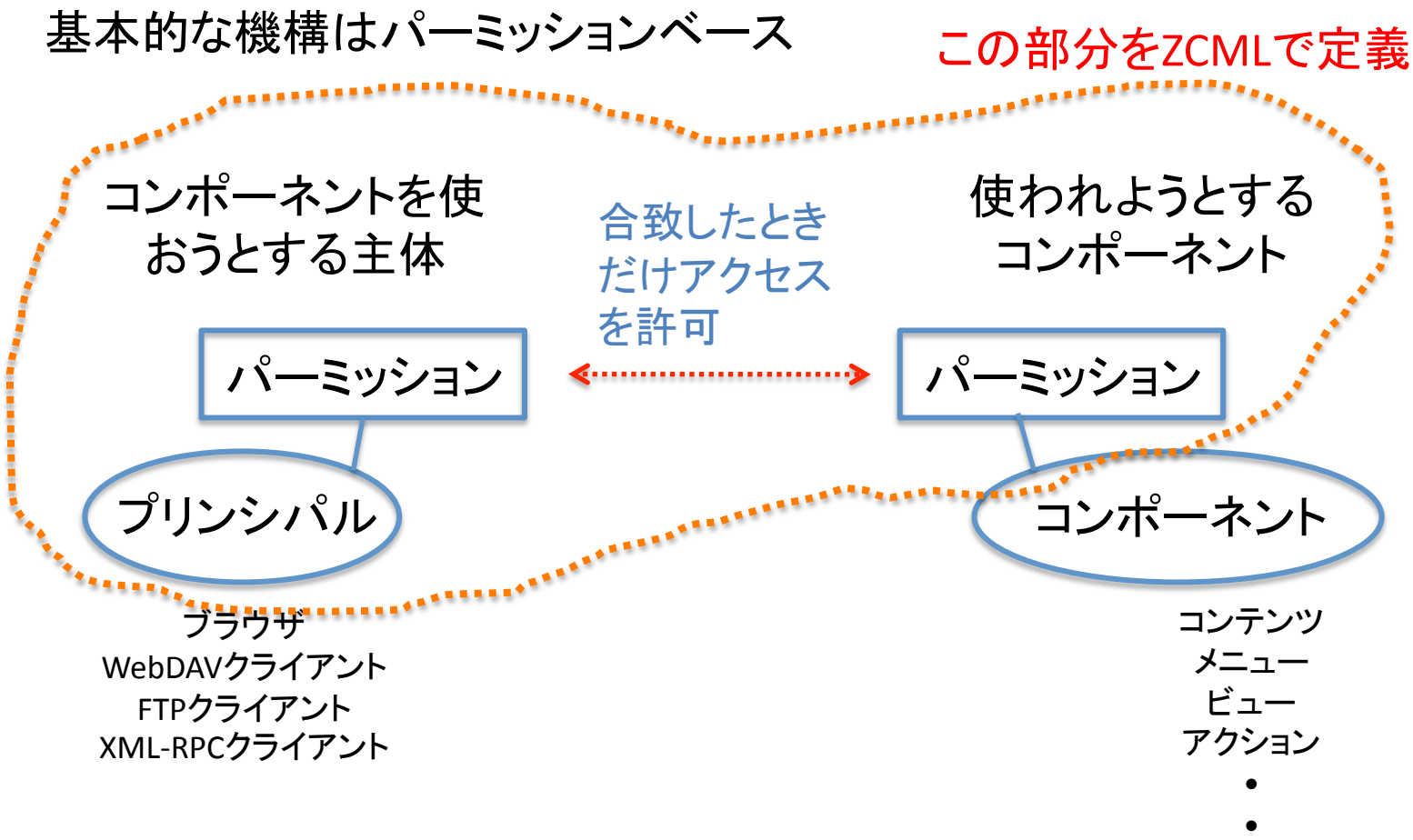
ZCML

Zope Configuration Management Language

コンフィグのためのXML派生言語



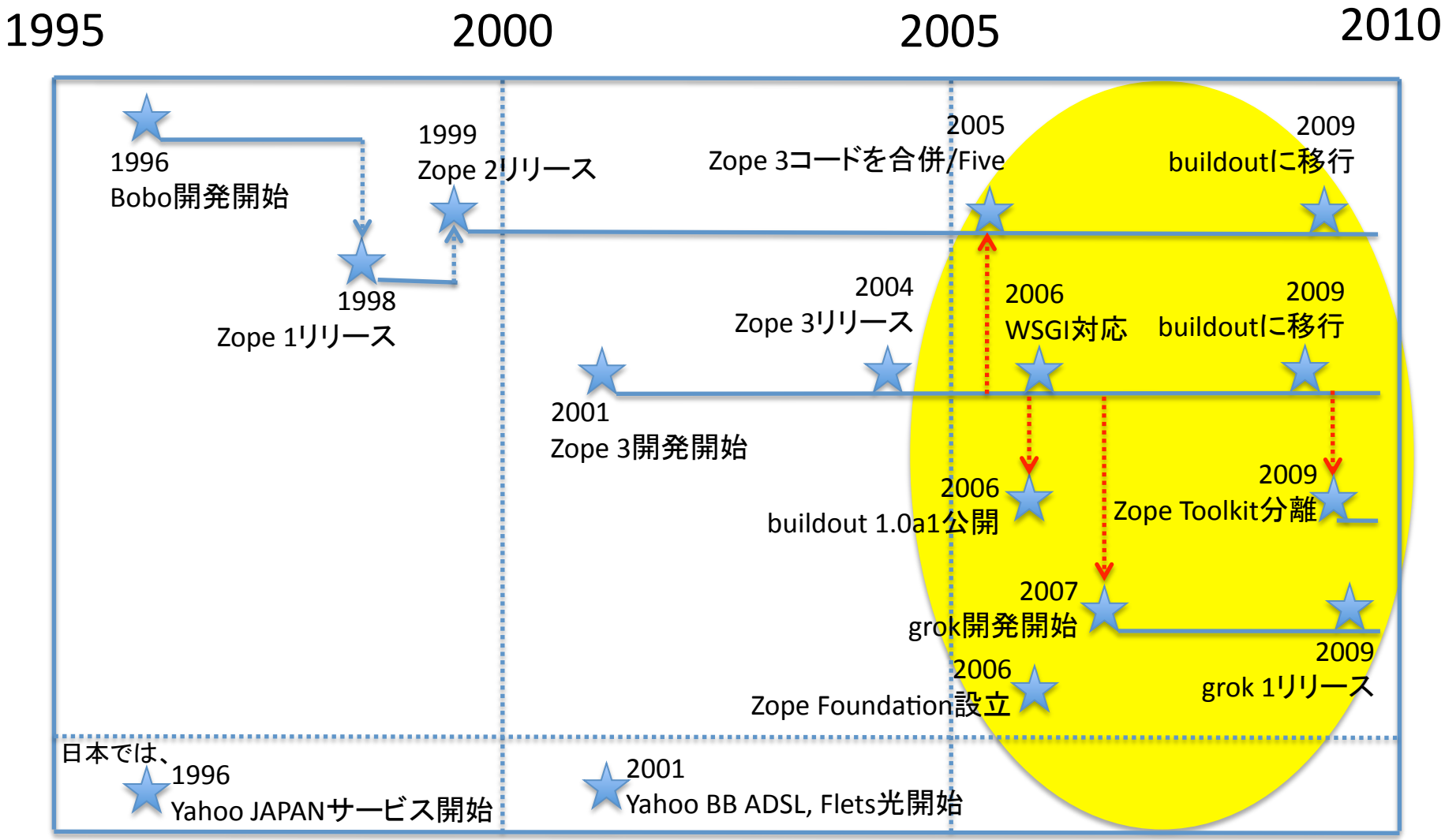
ZCMLによるセキュリティ定義



Ploneと、どう関係あるの？

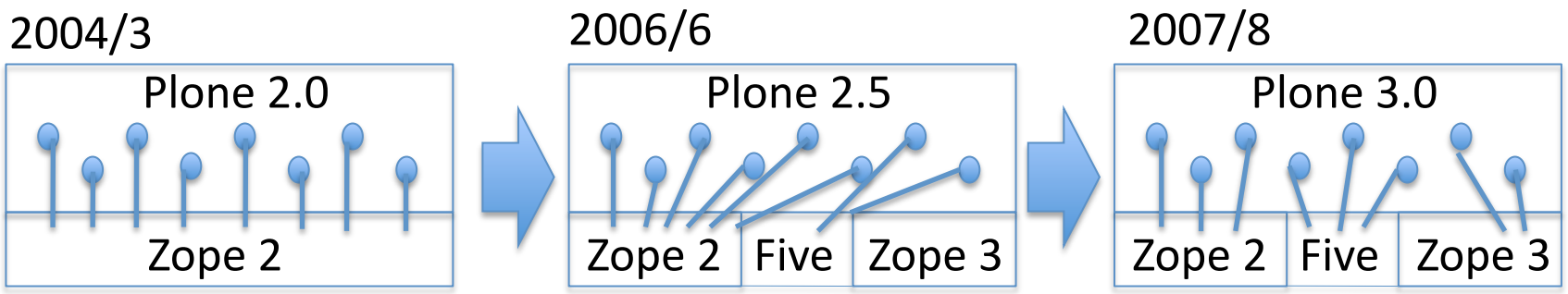
PloneってZope 2をベースにしてるでしょ、、、

現在の波


1995
2000
2005
2010
1996
Bobo開発開始
1998
Zope 1リリース
1999
Zope 2リリース
2001
Zope 3開発開始
2004
Zope 3リリース
2005
Zope 3コードを合併/Five
2006
WSGI対応
2006
buildout 1.0a1公開
2006
Zope Foundation設立
2007
grok開発開始
2009
buildoutに移行
2009
buildoutに移行
2009
Zope Toolkit分離
2009
grok 1リリース
日本では、
1996
Yahoo JAPANサービス開始
2001
Yahoo BB ADSL, Flets光開始

Zopeコードの統合とFive

- 2005年 Zope 2.8からZope 3とFiveを同梱
  - ZopeアプリケーションがZope 3コードを使い始める
  - 徐々に旧式なZope 2コードが廃止される



”Five”は Zope 2の中でZope 3の技術を使えるようにする技術

Ploneと、どう関係があるの？

- Ploneは、Zope 3への移行戦略として”Five”を採用した。
- その後、”Zope 3”への移行をあきらめ、Zope 2とZope 3のコードを統合することにした。
- 新しいPloneには、インタフェース、アダプタ、イベント、ZCMLなどのZope 3技術が使われている。
- Ploneはコンポーネントアーキテクチャ化されつつあり、配布方法のビルドアウト化と併せて、極めて柔軟に構成できるソフトウェアに変化しつつある。
- なので、Plone 4のコードを読んだり、アプリケーションを作るのに、Zope 3技術の理解が必要

Ploneでの使われ方

plone.orgのコードリポジトリを見てみよう

1. http://dev.plone.org/plone/browser/plone.contentrules/trunk/plone/contentrules

見つけましたか

- ZCML
- インタフェース
- スキーマ
- マーカーインタフェース
- アダプテーション

このZope 3本は、まだ役に立つの？

- もちろん！
- 今後、５～１０年は役に立ちます。
- コンポーネントアーキテクチャを、ここまでわかり易く学習できる文書は他にない。
- コード再利用のためにZopeに実装されたコンポーネントアーキテクチャは、実世界でうまくいき、現在その利用が拡大しています。