データベース設計論

第10回

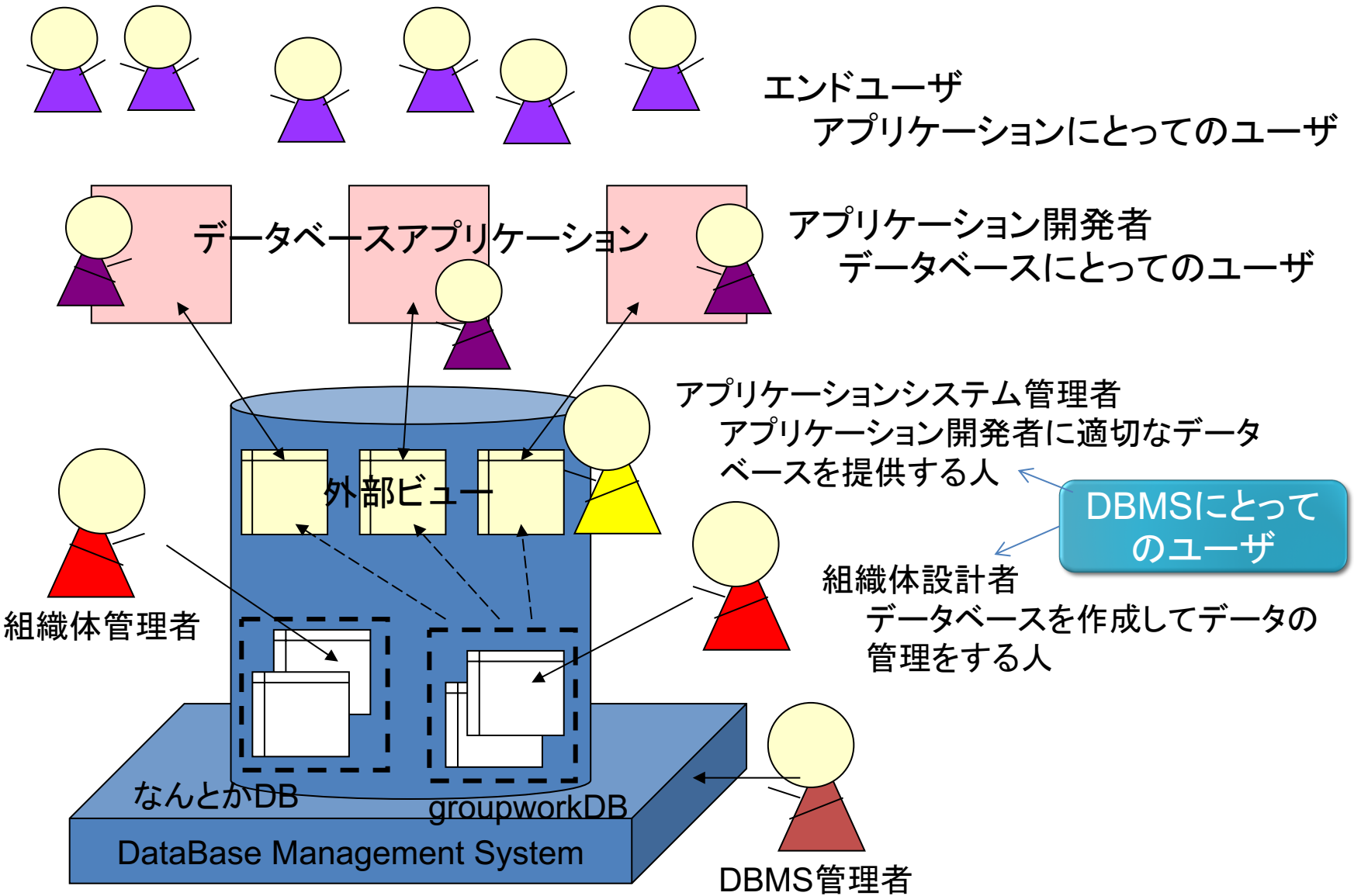
アプリケーションとデータベースの関係

本日の内容

- 「アプリケーションとデータベースの関係」

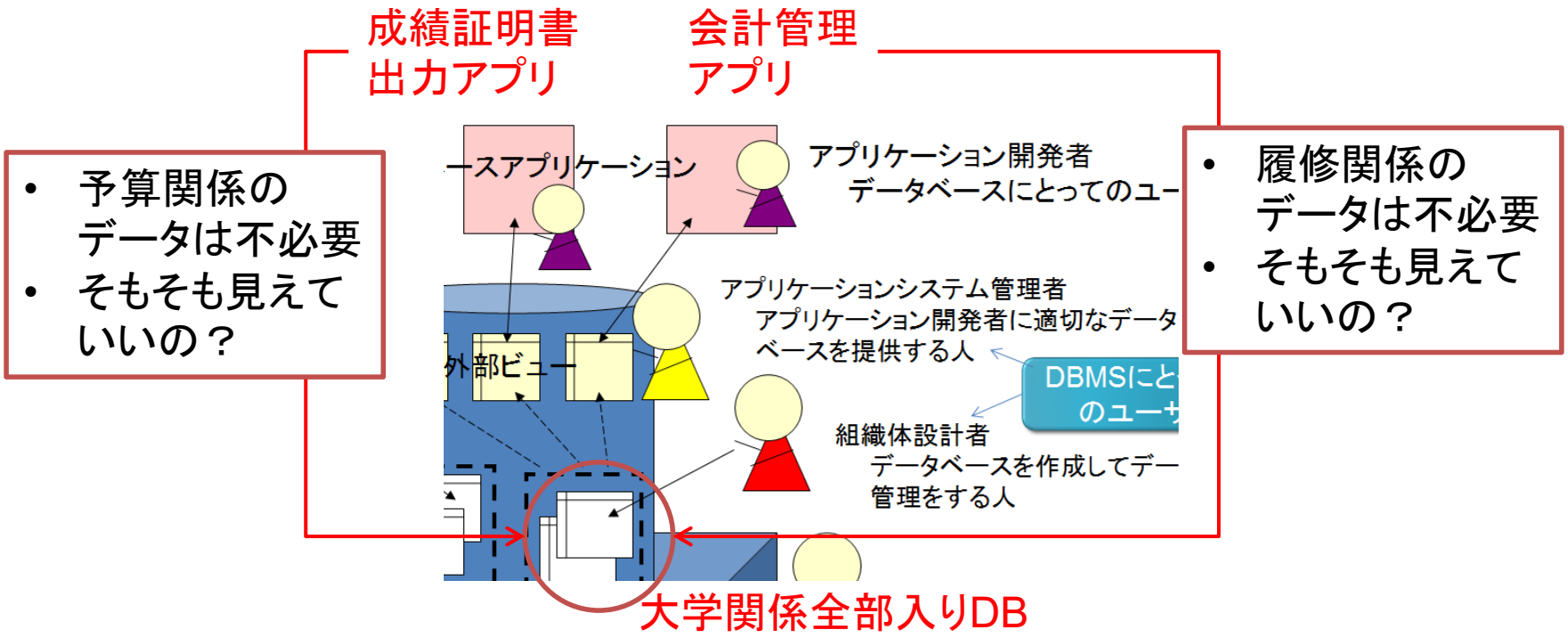
- Keywords
  - DBMSの利用者
  - データ独立性
  - ビュー（外部スキーマ）
  - ODBC, JDBC

DBMSの利用者


エンドユーザ
アプリケーションにとってのユーザ
データベースアプリケーション
アプリケーション開発者
データベースにとってのユーザ
アプリケーションシステム管理者
アプリケーション開発者に適切なデータベースを提供する人
DBMSにとってのユーザ
外部ビュー
組織体管理者
組織体設計者
データベースを作成してデータの管理をする人
なんとかDB
groupworkDB
DataBase Management System
DBMS管理者

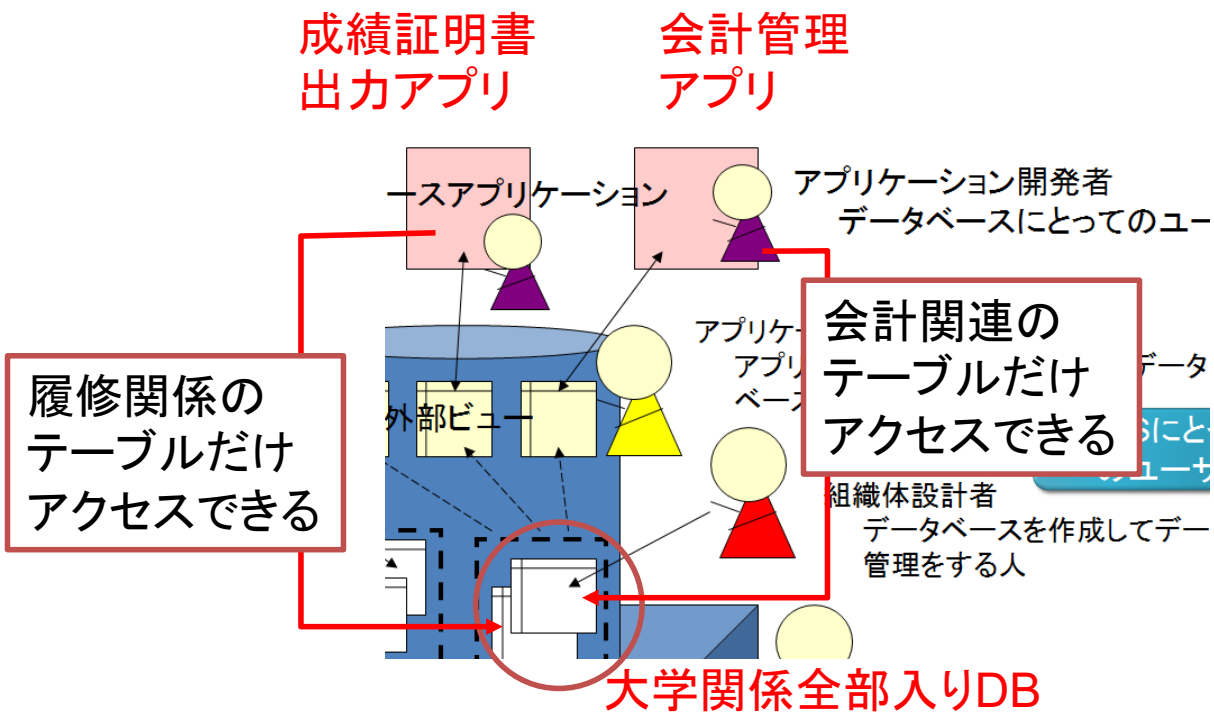
さて…

- アプリケーションシステム管理者はなぜ必要？
- 何でも入っている巨大なデータベースがあった時
  - 例）大学関係（教員名簿、予算、学生名簿、履修、成績）が全部入っているデータベース



アプリケーションシステム管理者のお仕事

- アプリケーション開発者に適切なDBやテーブルのアクセス権を与える



一部のテーブルだけ見せたい場合は？

- Groupworkデータベースの例
  - studentsに教員も学生も全員入ってしまっている（リレーション名的にそれはまずいのだが…）
- 学生が主体で開発するアプリがGroupworkのメンバー一覧を観れるようにしたい
  - 教員はいらない
  - パスワードもいらない

元データ (students)

stid	name	grade	password
1	山田花子	3	x;;jkaj;ajklak
2	渡辺知恵美	20	4ej;sj;j;sjkld
3	浅田紀子	2	sj;ekejrhjwk

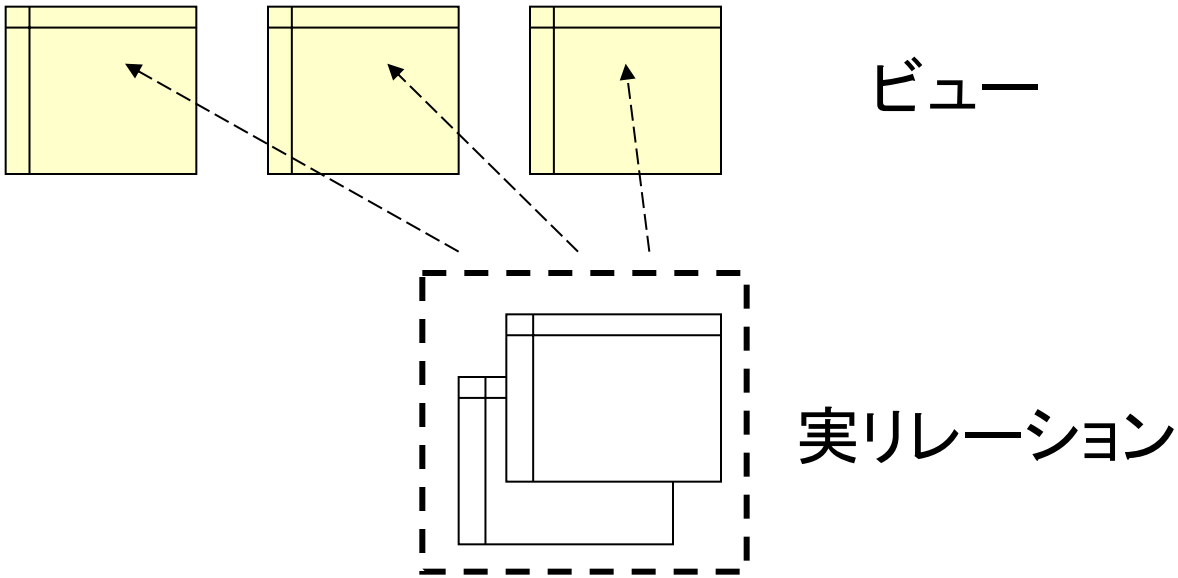
アプリ開発者に提供するデータ

stid	name	grade
1	山田花子	3
3	浅田紀子	2

さてどうする？

ビュー（仮想リレーション）

- 実リレーションから導出される結果リレーションをビューとして保存できる
- ビューが保存するのは結果リレーション本体でなく，ビューを導出した問合せ文
  - ビューの利用者は一見普通のリレーションに見えるが，実はそのつど実リレーションに問合せして，実態を生成する



ビューの作成

- CREATE VIEW構文を使う

CREATE VIEW ビュー名 AS 問合せ文

学生の名簿情報のみを抽出したビューを作る

```
CREATE VIEW groupwork_students
AS
SELECT stid, name, grade
FROM students
WHERE grade between 1 and 4
```

ビューはなぜ必要なのか

- 実リレーションから必要なデータだけを抽出しておきビューとして保存できる
  - 実リレーションが多数存在し，お互いが複雑に関係しているとき，ビューを作ることで問合せ文を単純化することができる
- アクセス権の制御ができる
  - 利用者やアプリケーション開発者には，実利レーションでなくビューだけにアクセスできるように設定すればセキュリティを保護することができる

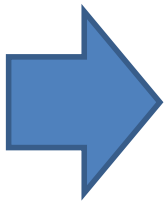
もう一つ大事な使い方

- リレーションスキーマの変更を余儀なくされたとき問題になるのは、すべてのアプリケーションに影響が出ること

ビューを使えば回避できる

- 例) これまで属性に年齢をつけていたが生年月日に変更することにした場合

id	name	age
1	Mao Asada	23
2	Elena Radionova	14
3	Adelina Sotnicova	17



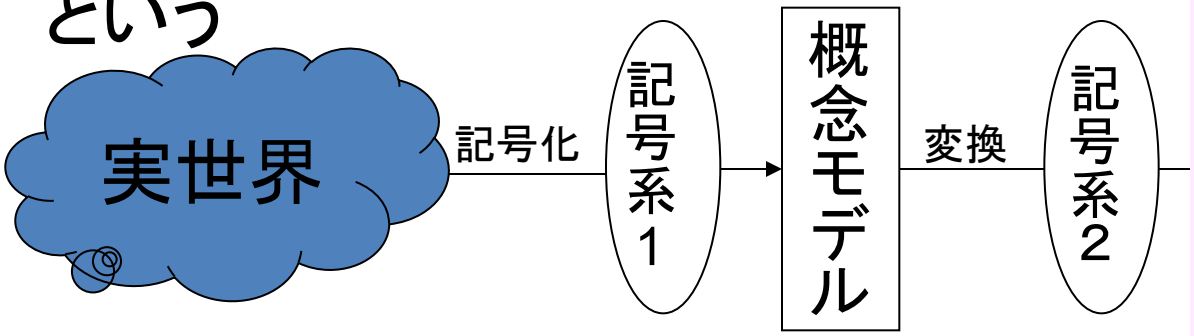
id	name	birthdate
1	Mao Asada	1990/9/25
2	Elena Radionova	1999/1/6
3	Adelina Sotnicova	1996/7/1

さてどうする？

※年齢計算は以下の関数でできることとします
calcage(birthdate , today)

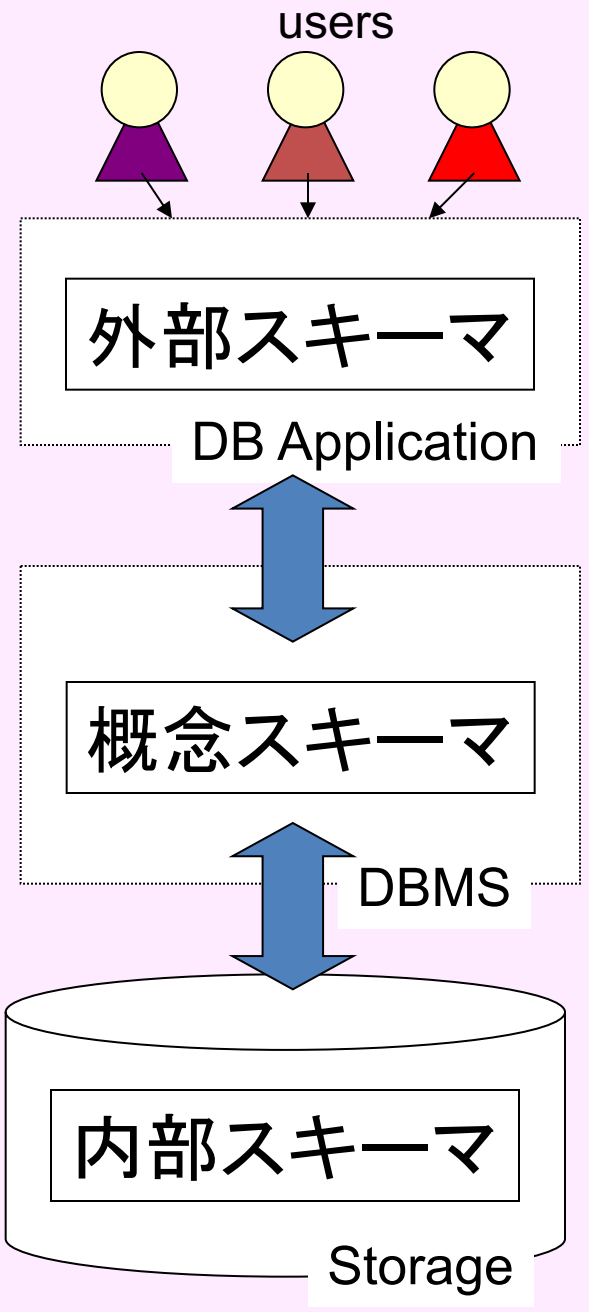
データベースの基本的概念：データ独立性

データベースとアプリケーション，ストレージの関係を独立して扱うことができることをデータ独立性という



Universe of Discourse(UoD):
対象世界，対象領域

ANSI/X3/SPARCの3層スキーマ

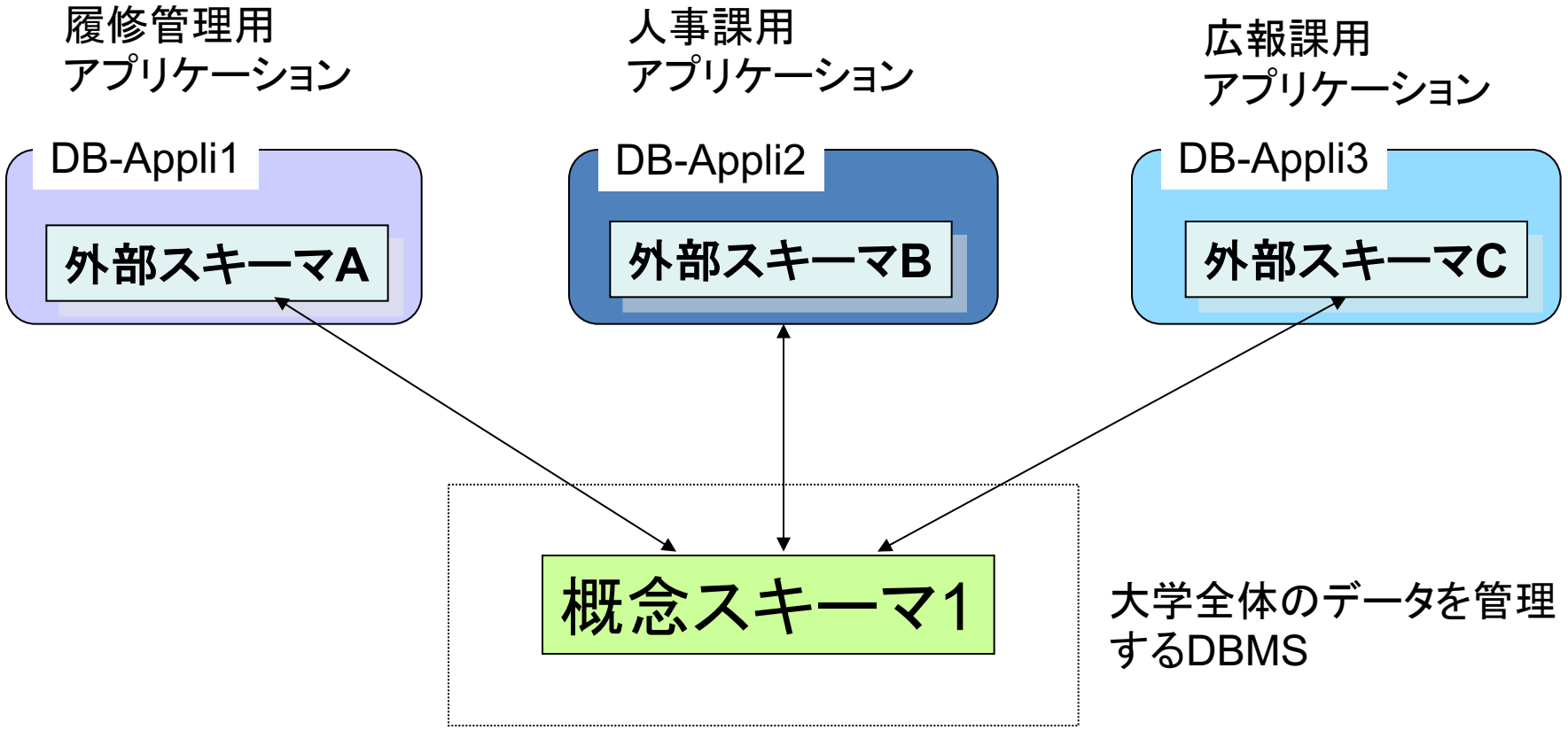


三層スキーマ

- 概念スキーマ(conceptual schema)
  - データモデルによりUoDをデータベースに「論理的に」記述したもの
  - 前述の「論理スキーマ」と同等のものである
- 内部スキーマ(internal schema)
  - 実際のストレージ（ディスク装置，ファイルなど）に格納されている形式
- 外部スキーマ(external schema)
  - データベースアプリケーションが利用するデータのデータ形式
  - ビューとよぶ

外部スキーマ(ビュー)

- DBアプリケーションはDBMSに管理されている大きなデータベースに対して，自分の必要なデータだけを「ビュー」として利用することが出来る



ANSI/X3/SPARCの3階層スキーマ

外部スキーマ

概念モデル

論理的データ独立性

概念スキーマ

論理モデル

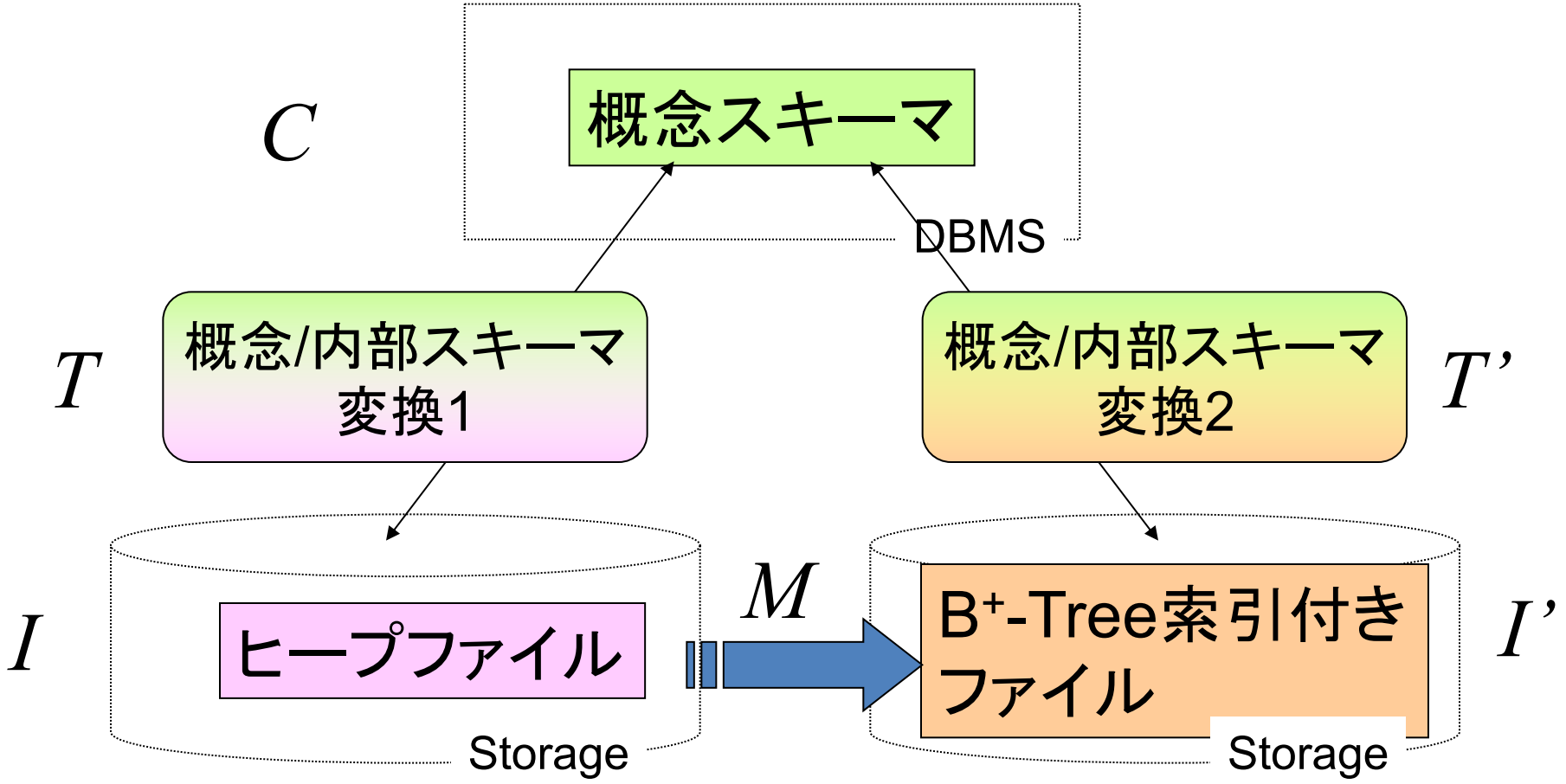
物理的データ独立性

内部スキーマ

内部モデル
物理モデル

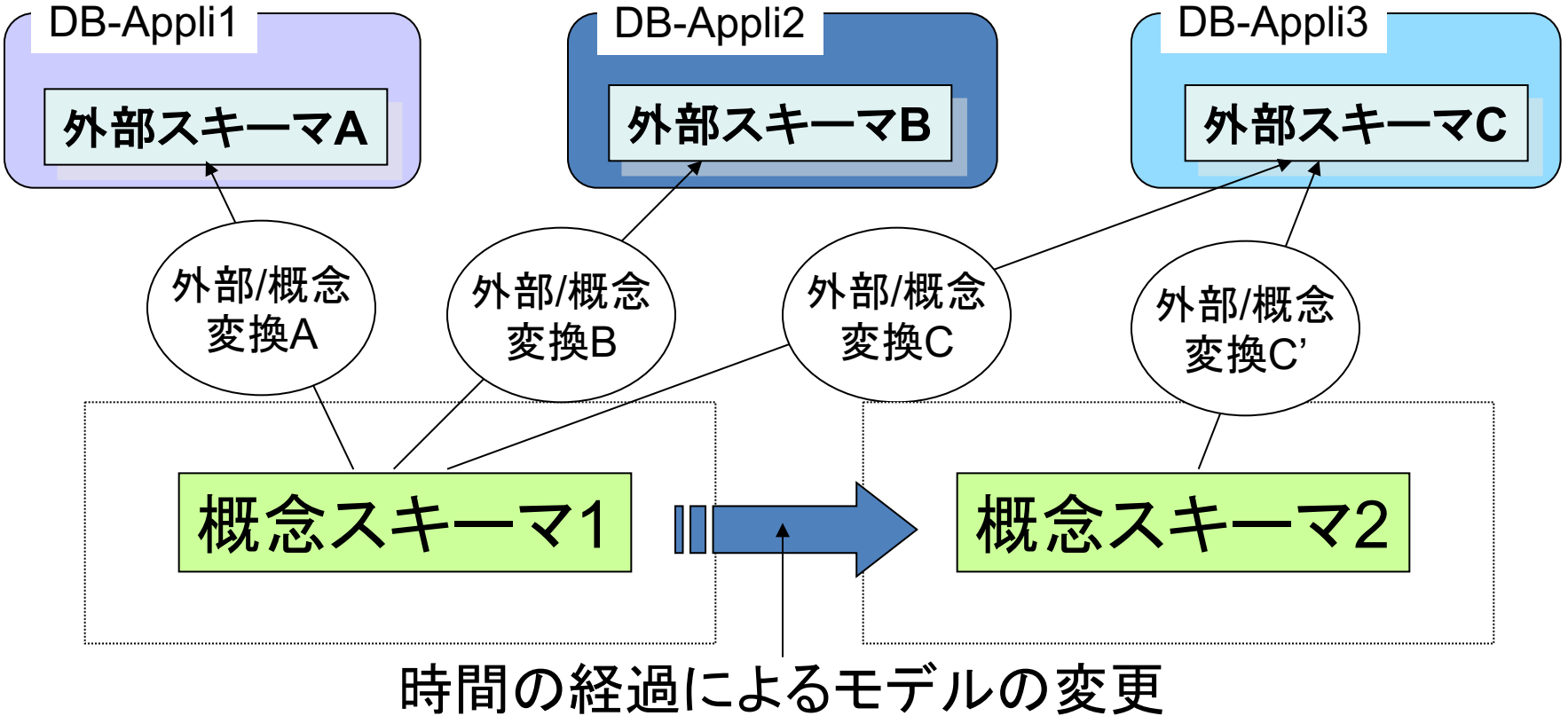
物理的データ独立性

- 内部スキーマの変更が概念スキーマに影響を与えないこと



論理的データ独立性

- 概念スキーマの変化がDBアプリケーションに影響を与えないこと

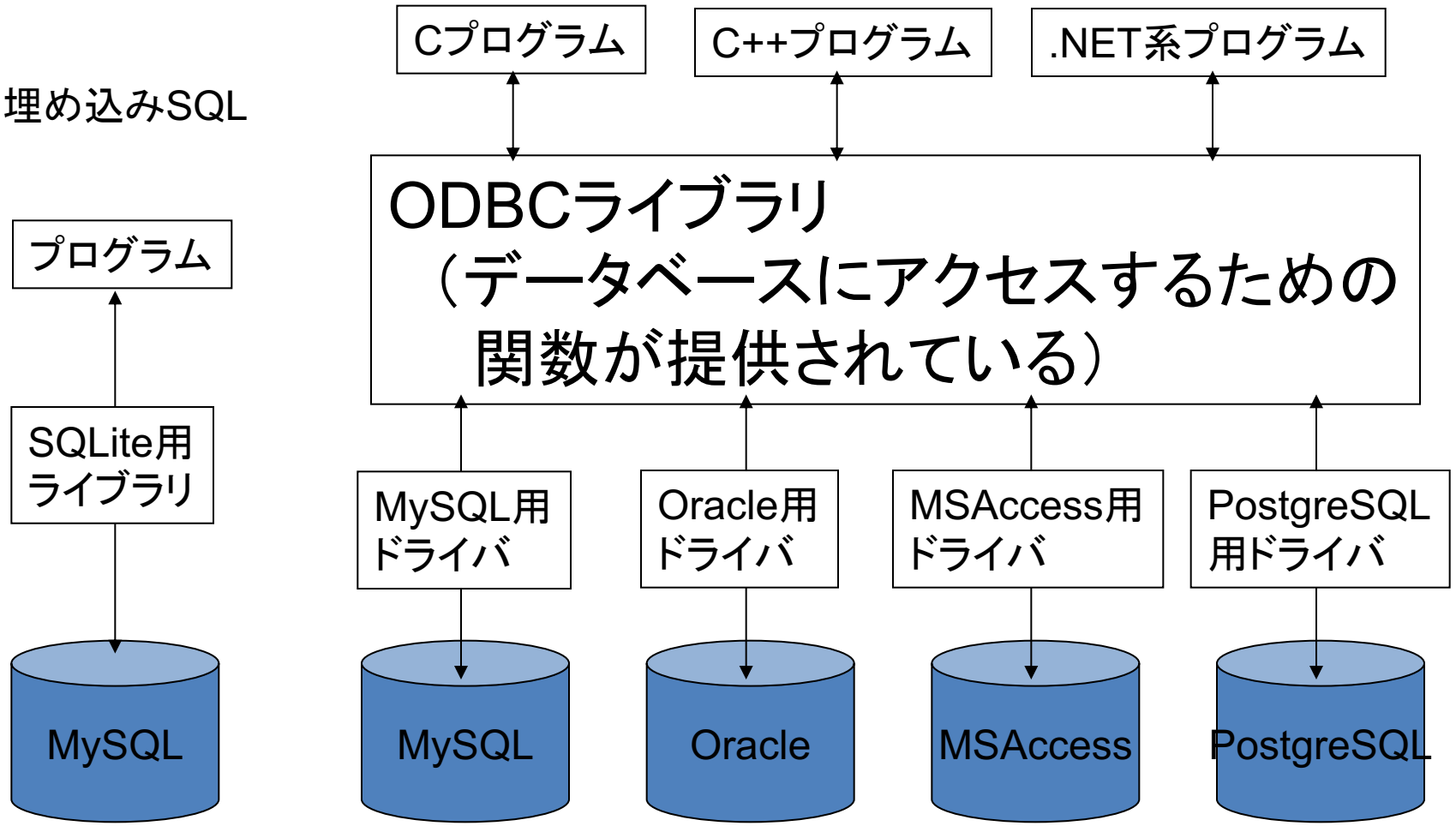


データベースアプリケーションの開発

データベースに接続するプログラムを作る方法

- 埋め込みSQL(Embedded SQL)
  - 各RDBMSベンダーがプログラミングからデータベースにアクセスするためのライブラリ（API）を提供している
  - それぞれの埋め込みSQLにより関数や使い方が異なる
  - Native Interface
- ODBC
  - Microsoftが提唱した，RDBMSに対する共通のアクセスインタフェース
  - 各RDBMSベンダーはODBCへのドライバを提供する
- JDBC
  - ODBCのJava版
- DBI(perl), PDO(PHP), ActiveRecord(Ruby)

埋め込みSQL，ODBCの仕組み



SQLite3にアクセスできるプログラミング言語

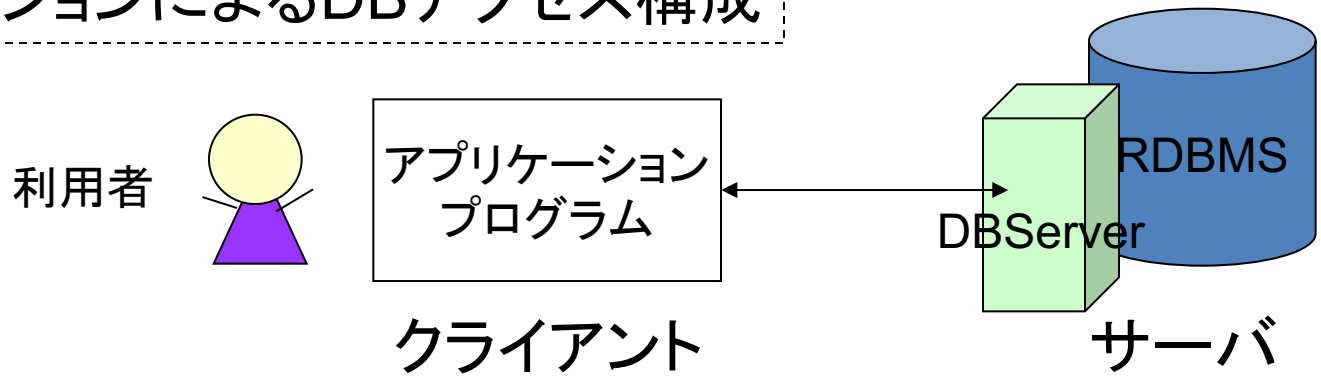
SQLite has bindings for a large number of programming languages, including:

- AutoIt[25]
- BASIC
- C
- C#
- C++
- Clipper//Harbour
- Common Lisp
- Curl
- D
- Delphi
- Elixir
- F#[26]
- FreeBASIC
- Free Pascal
- Go
- Haskell
- Haxe
- Java (on JVM and DVM)
- JavaScript[27]
- Julia
- Livecode
- Lua
- newLisp
- Nim
- Objective-C (on OS X and iOS)
- OpenLisp
- OCaml
- Perl[28]
- PHP
- Pike
- PureBasic
- Python[29]
- R
- REALbasic
- REBOL
- Ruby[30]
- Scheme
- Smalltalk
- Swift (on OS X and iOS)
- Tcl
- Visual Basic
- Xojo

Wikipediaより

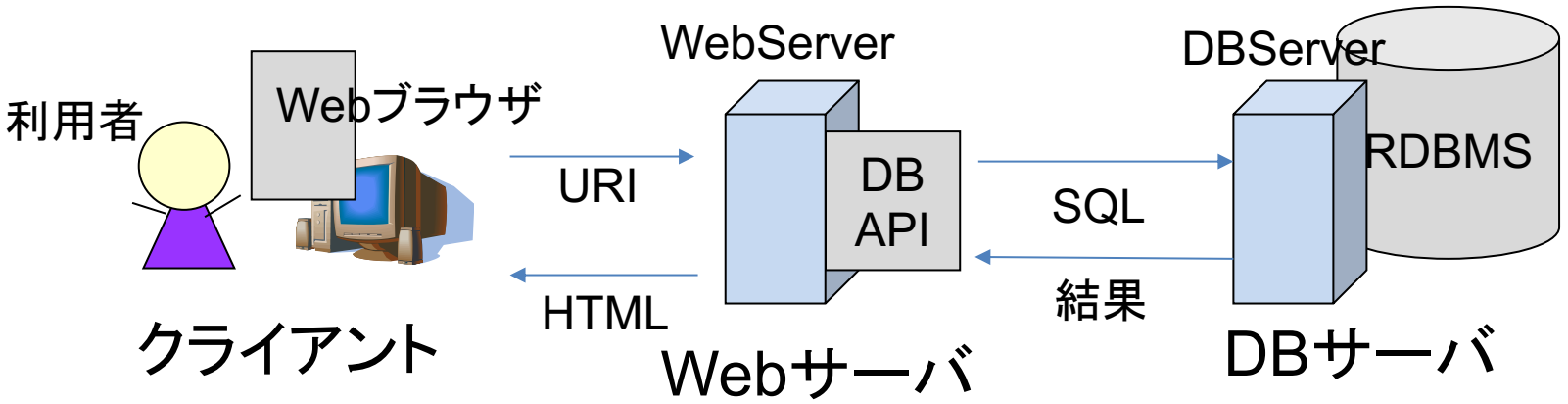
データベースアプリケーションの構成 (1/2)

アプリケーションによるDBアクセス構成

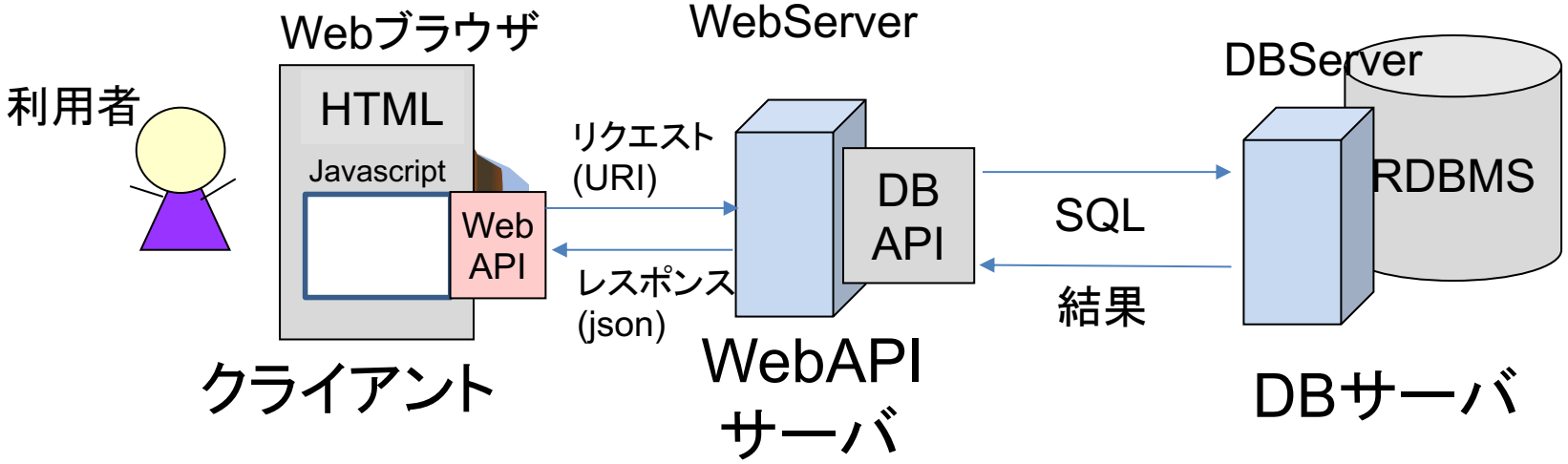


データベースアプリケーションの構成(2/2)

WEB+DBによる三層構成（基本形）



- 今時Webアプリの構成 (Ajax)



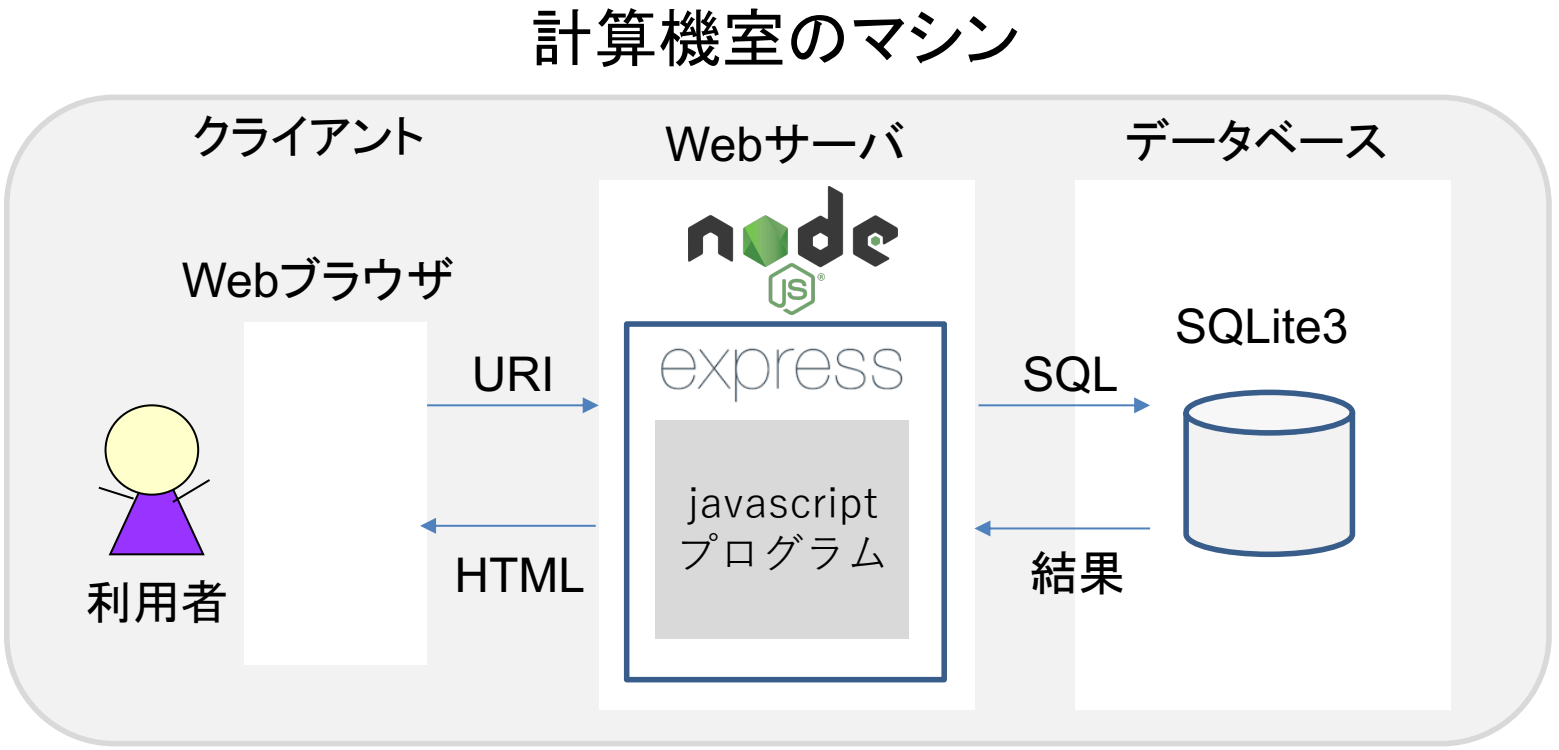
演習（クリスマスプレゼント付き）

- APIとはなんの略で
  何を表すものでしょうか?

- WebAPIとはなんですか?

演習：Web＋DBアプリケーションを作ろう

- 基本構成(基本形バージョン)ですが、
  クライアント・Webサーバ・DBサーバ全部同じホストに
  ある設定で動かします



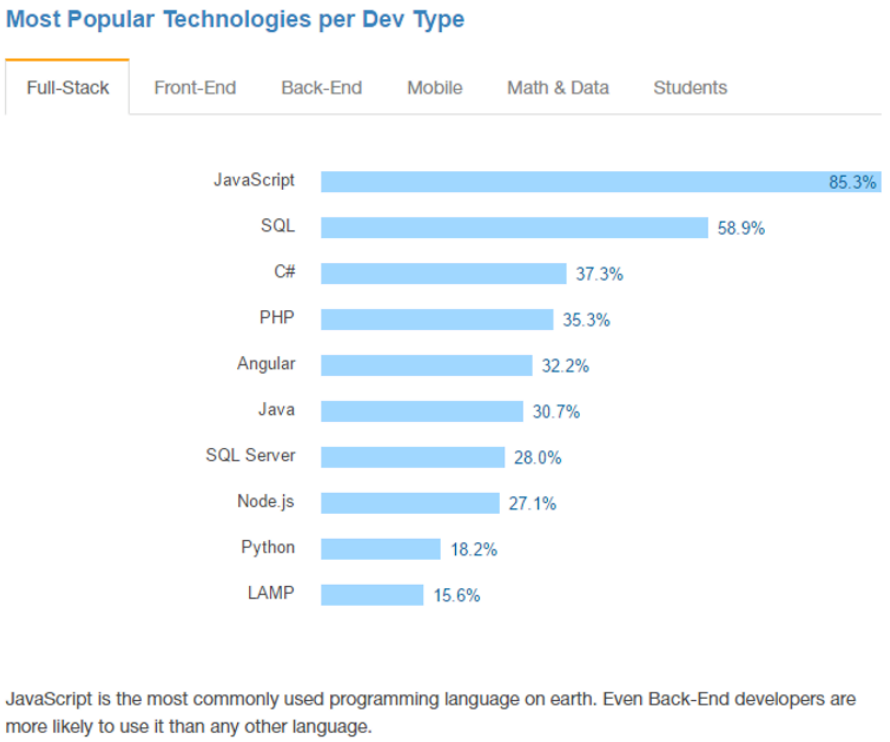
補足

- SQLiteはDBMSではなく「SQLでデータベースファイルにアクセスできるライブラリ」
  - データベースサーバは用意せずWebサーバにデータベースファイルを置いてアクセスする

- クライアントから同じホストにあるサーバにアクセスするときは
  http://localhost/xxx
  もしくは
  http://127.0.0.1/xxx
  と指定する

JavaScript

- (もともとは)ウェブブラウザ上で動作するスクリプト言語
  - クライアントサイド開発で使われる
- 最近はWebサーバでも使われる



サンプルプログラム

- https://github.com/OchaDB2019/sample1

app.js の一部

```
var sqlite3 = require('sqlite3').verbose()
var db = new sqlite3.Database('twitter.db')

app.get('/', function (req, res, next) {
    var query = "\
        SELECT t.account, u.name, t.datetime, t.content\
        FROM tweet t, follow f, user u\
        WHERE t.account = u.account and f.follower_account = 'mob1' and f.followee_account = t.account;\
        ";
        console.log("DBG:" + query);
    db.all(query, {}, function (err, rows) {
        if (err) {
            console.log("ERROR: " + err.message);
        }
        res.render('index', {
            results: rows
        })
    })
});

app.listen(3000, () => console.log('Example app listening on port 3000!'))
```

Webフレームワークに関するコード解説

- 12行目：
  app.get('/', function (req, res, next) {

  http://localhost:port/ というURLでアクセスしてきたときに
  実行するプログラムがここに書かれる

  });
- 23行目：
  res.render('index', { results: rows });

  views/index.pug をベースにHTMLを作成してクライアントに渡す。
  rows(データベース検索の結果が入ってる)をresultとして渡している。

views/index.pug

- pug (https://pugjs.org/) を使ってます

```
1  html
2      head
3          link(ref="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons")
4          link(rel="stylesheet" href="css/materialize.min.css" media="screen,projection")
5          meta(name="viewport" content="width=device-width, initial-scale=1.0")
6      body
7          script(type="text/javascript" src="js/materialize.min.js")
8          div(class="container")
9              each row in results
10                 div(class="tweet")
11                     div(class="tweetinfo") #{row['name']} @#{row['account']} #{row['datetime']}
12                     div(class="row")
13                         div(class="col s10 offset-s1 content") #{row['content']}
```

演習（クリスマスプレゼント付き）

- これまで以下のツール&専門用語が出てきました。
  これらについて調べて説明をしましょう。

express

https://expressjs.com/ja/

Webフレームワーク

データベースに関するコード解説

- 9行目: sqlite3ライブラリを呼び出す
  var sqlite3 = require('sqlite3').verbose();
- 10行目: データベース twitter.dbを読み込む
  var db = new sqlite3.Database('twitter.db');
- 19行目 : 問い合わせをして、その結果をrowsに入れる

```
db.all(query, {}, function(err, rows){
    if (err) {
        エラーが起きた時の処理
    }
    res.render('index', { results: rows});
}
```

計算機室での動かし方

最初の設定

大学の学科計算機室で動かす場合

~/.bash_profileというファイルを開き（なければ作る）、以下の１行を足す

```
export PATH=$PATH:~chiemi/lib/npm/bin
```

保存したら、以下を実行

```
% source ~/.bash_profile
```

自分のPCで動かす場合

- Node.js をインストール https://nodejs.org/ja/

とりあえず動かしてみる

1. 必要なモジュールをインストールする

```
% npm install
```

2. Webサーバを立ち上げる

```
% node app.js
```

3. ブラウザで http://localhost:3000 にアクセスしてみる

グループワーク：課題5

- 課題4で書いた「アプリの画面を構成するのに使っていそうなSQL文」を実際に実行し、アプリの画面を再現しましょう。
  - SQL文は3つ全部再現しなくてもいいですが、再現した数は評価で考慮します
    - 検索フォームとか入力フォームとかも再現するのも推奨
    - 画像の出し方について別のサンプル作ります
  - 画面デザインまで再現する必要はありません
    - 好きな人は是非どうぞ
  - 提出方法
    - ソースコードを githubにあげ、以下の内容をPDFで提出
      - 再現元の画面（アプリの画面。個人情報等は適当に消してください）
      - 実行結果の画面
      - プログラムと解説
      - 工夫したところ