embarcadero®

Developer Camp

【C1】Delphi/C++チュートリアルセッション

マルチデバイスの荒海にこぎ出す新人エンジニアのためのソフトウェア開発の心得

MVVM 入門

株式会社シリアルゲームズ　取締役
エンバカデロ MVP
細川 淳

はじめに

セッション概要

- マルチデバイスに対応するアプリケーションは、どのように設計すべきか？
- OS の違いは？
- 画面サイズの違いは？
- そもそもアプリケーションの設計とは？

「設計書の書き方」とかではなく
概念的な話になります

アジェンダ

- アプリケーションの設計
- Delphi で開発する事
  – 今までの問題点
- 打開策
  – MVVM

アプリケーションの設計

アプリケーションの設計

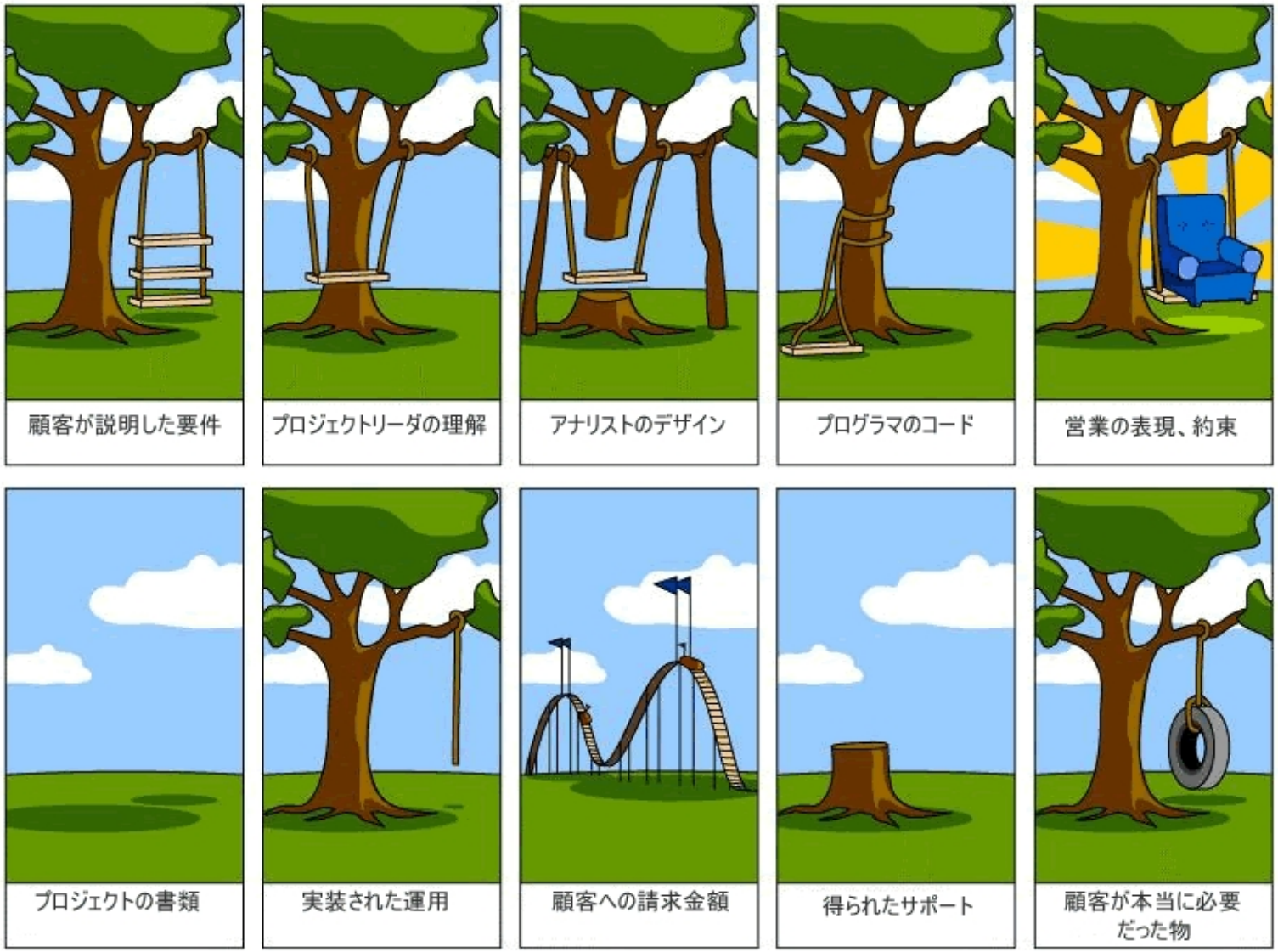
- 設計には大きく分けて4つあります（僕調べ）
  - 要件の定義
    - ユーザー要件定義書とか
  - システムの設計
    - システム構成
      - ハードウェア構成・ミドルウェア構成とか

アプリケーションの設計

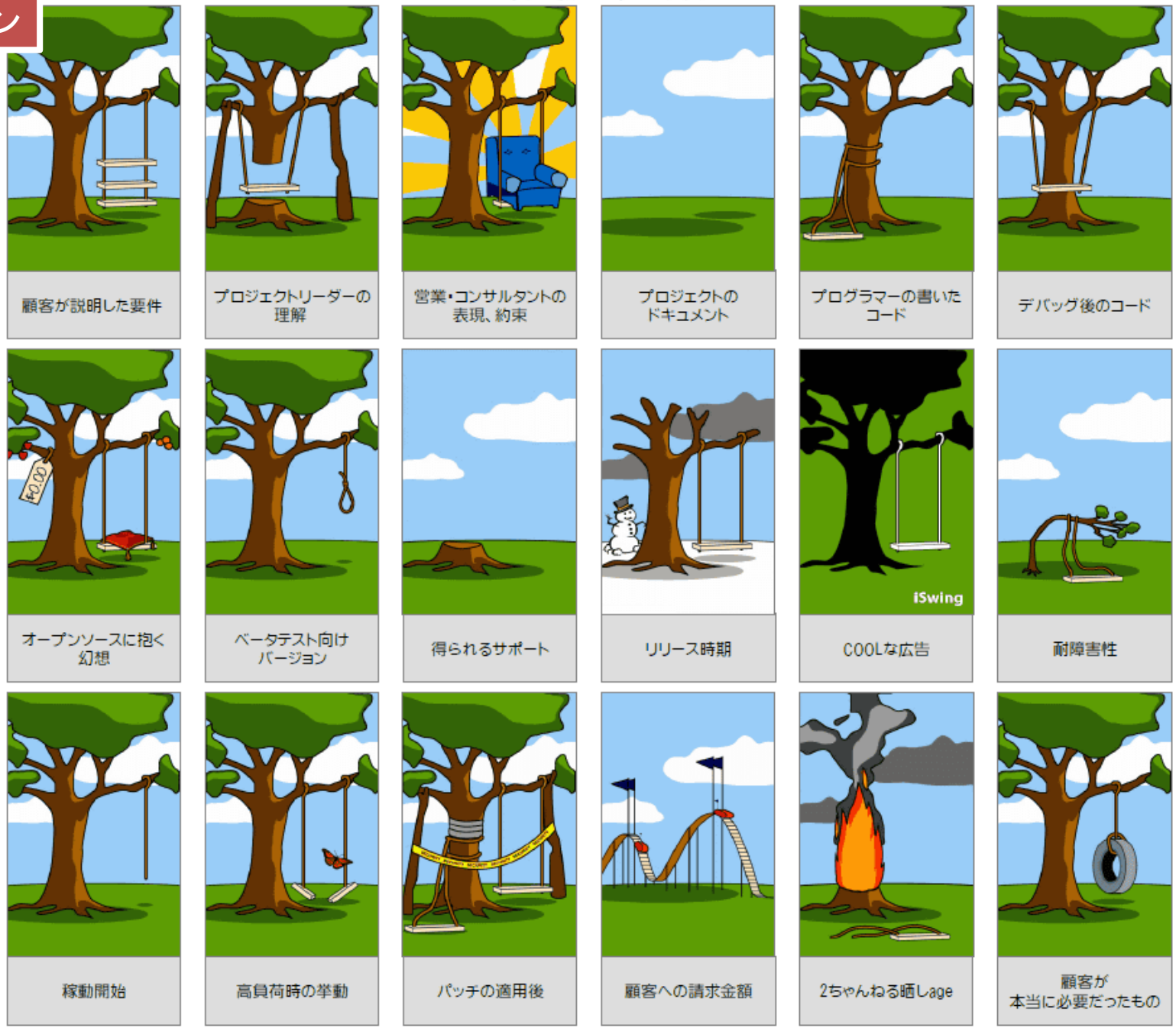
- コードの設計
  - 詳細設計とか
    - え、Excel仕様書とか……
- UI / UX の設計
  - 画面仕様設計
    - 開発者が考えることが多い（本当は良くない）

アプリケーションの設計
要件定義・システム設計

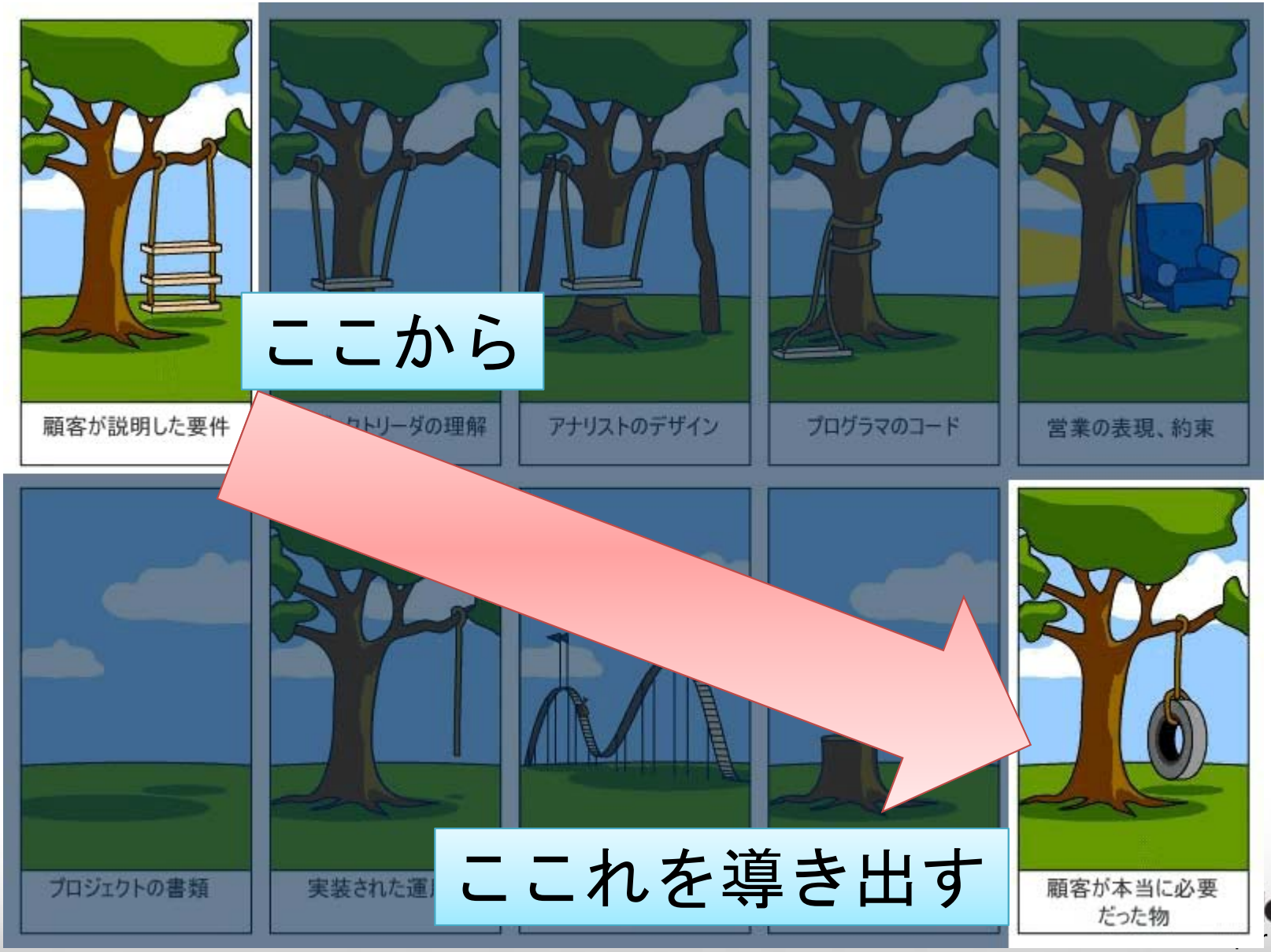
アプリケーションの設計-要件定義



最近バージョン


【プロジェクトの姿 – 顧客が本当に必要だったもの】
顧客が説明した要件
プロジェクトリーダーの理解
営業・コンサルタントの表現、約束
プロジェクトのドキュメント
プログラマーの書いたコード
デバッグ後のコード
¥0.00
オープンソースに抱く幻想
ベータテスト向けバージョン
得られるサポート
リリース時期
iSwing
COOLな広告
耐障害性
稼動開始
高負荷時の挙動
パッチの適用後
顧客への請求金額
2ちゃんねる晒しage
顧客が本当に必要だったもの
Copyright © 2006-2009 maniyax.net and 2006 ProjectCartoon.com. All Rights Reserved.

アプリケーションの設計-要件定義



アプリケーションの設計ー要件定義

- 大きな組織の場合
  - 設計と実装が別の場合が多い
  - さらに、営業と提案も別
  - これによって前述の問題が起きる

- スタートアップなど小さな組織の場合
  - 営業と提案と設計と実装が同じ
    - 顧客が求めていることさえ判れば、割と適切な実装ができる

アプリケーションの設計ー要件定義

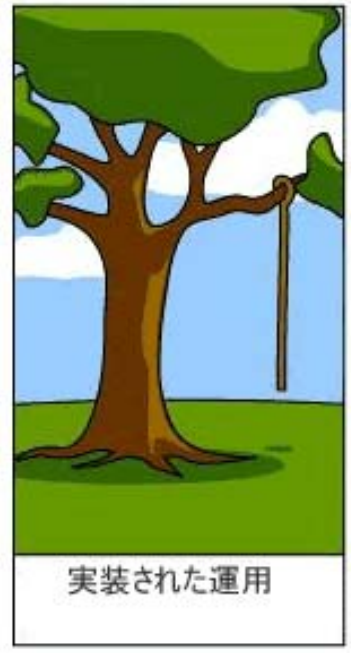
- 新人でも！！
  - 顧客との折衝や提案の場に連れて行って貰う
  - その場にいるだけでもOK
    - 先輩や営業が、どのような話をしているのかを聞く
      - 疑問があれば、帰社してから話を聞いても良い！
        - » 新人では想像できない何かの問題をはらんでいて、それを回避するための何かかもしれない！！

アプリケーションの設計
システム設計

- ハードウェアの構成
  - サーバ構成
    - ハードウェア選定
    - アーキテクチャ選定
  - DBの選定
    - Oracle, MySQL, MongoDB など、目的ごとに選定する
- ミドルウェアの構成
  - tomcat, node.js などなど、目的ごとに選定する

アプリケーションの設計
システム設計

- 割とダイレクトにお金が関わる部分
  - 本当はこういう構成にしたいけど**お金出せない**！って言われる可能性がある（言われた）
  - **運用でカバー**しようとか言われることがある



←こうなりがち。
なぜか？
運用に入った段階で開発者は次の開発に入るため

アプリケーションの設計
システム設計

- ここに関しては、あまり関与できる部分は少ないかも
  - 物理的な金額は、あまり変動しない
  - 新人では調達費用など判らない
    - ただし、将来のために学ぶ必要は有り

アプリケーション設計
コード設計・UI設計

アプリケーションの設計

- Delphi での開発は
  - コード設計
  - UI 設計

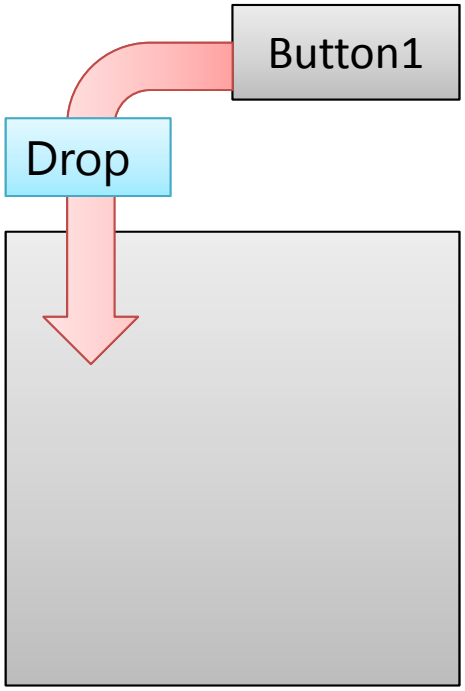
この２つが特に重要

- Delphi の場合、この２つが密接に関連しすぎている

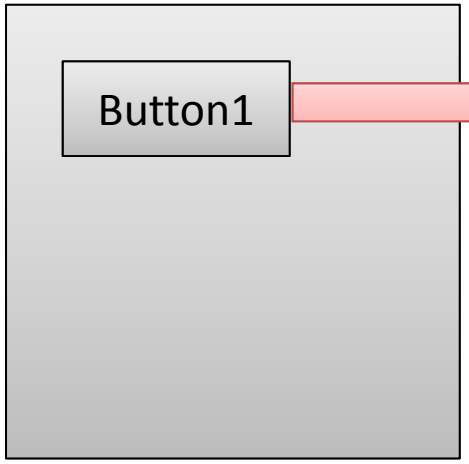
Delphi での開発

- 真の Visual 開発であることの問題

D&D によるビジュアル開発はとても簡単だが……

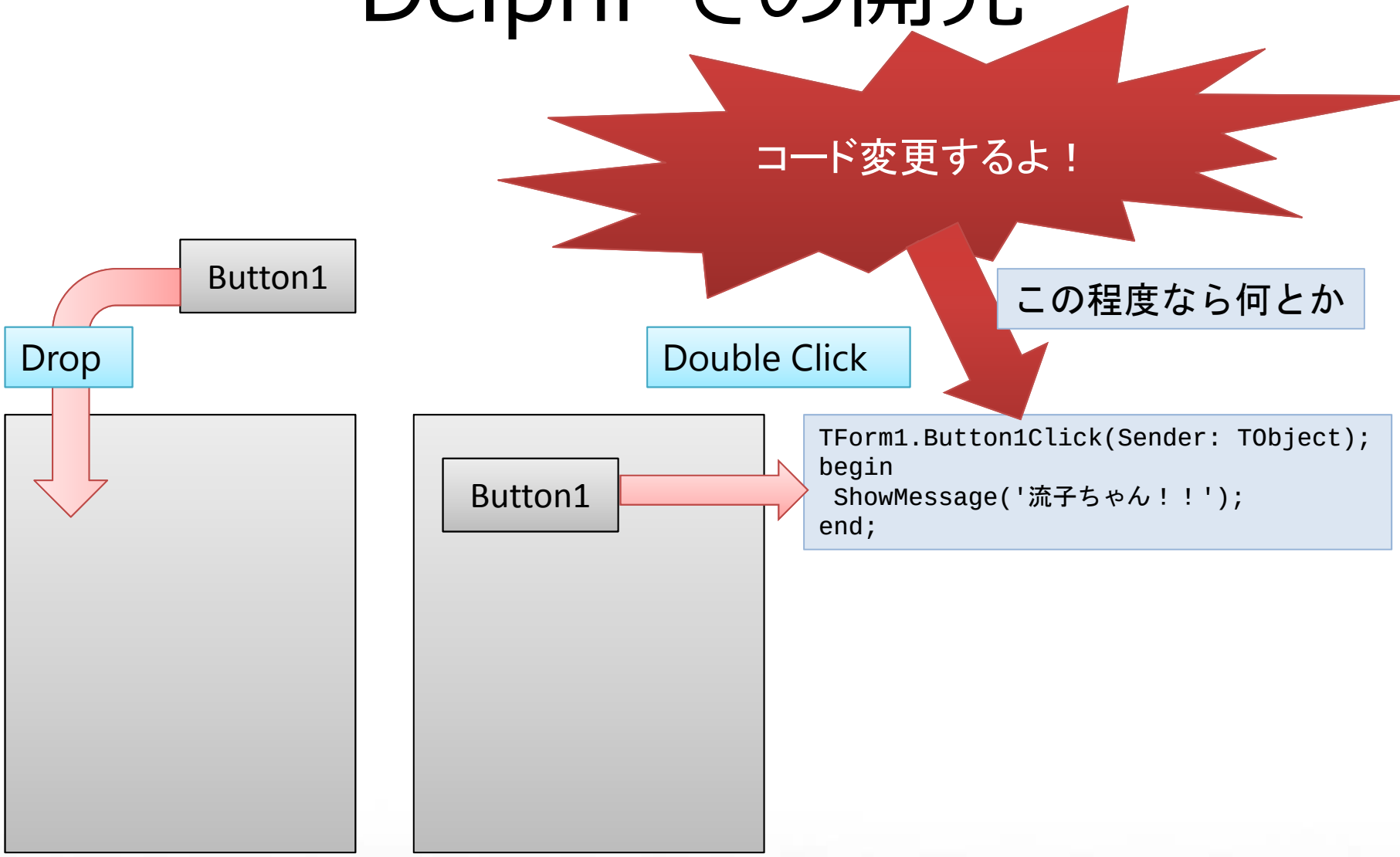


Double Click

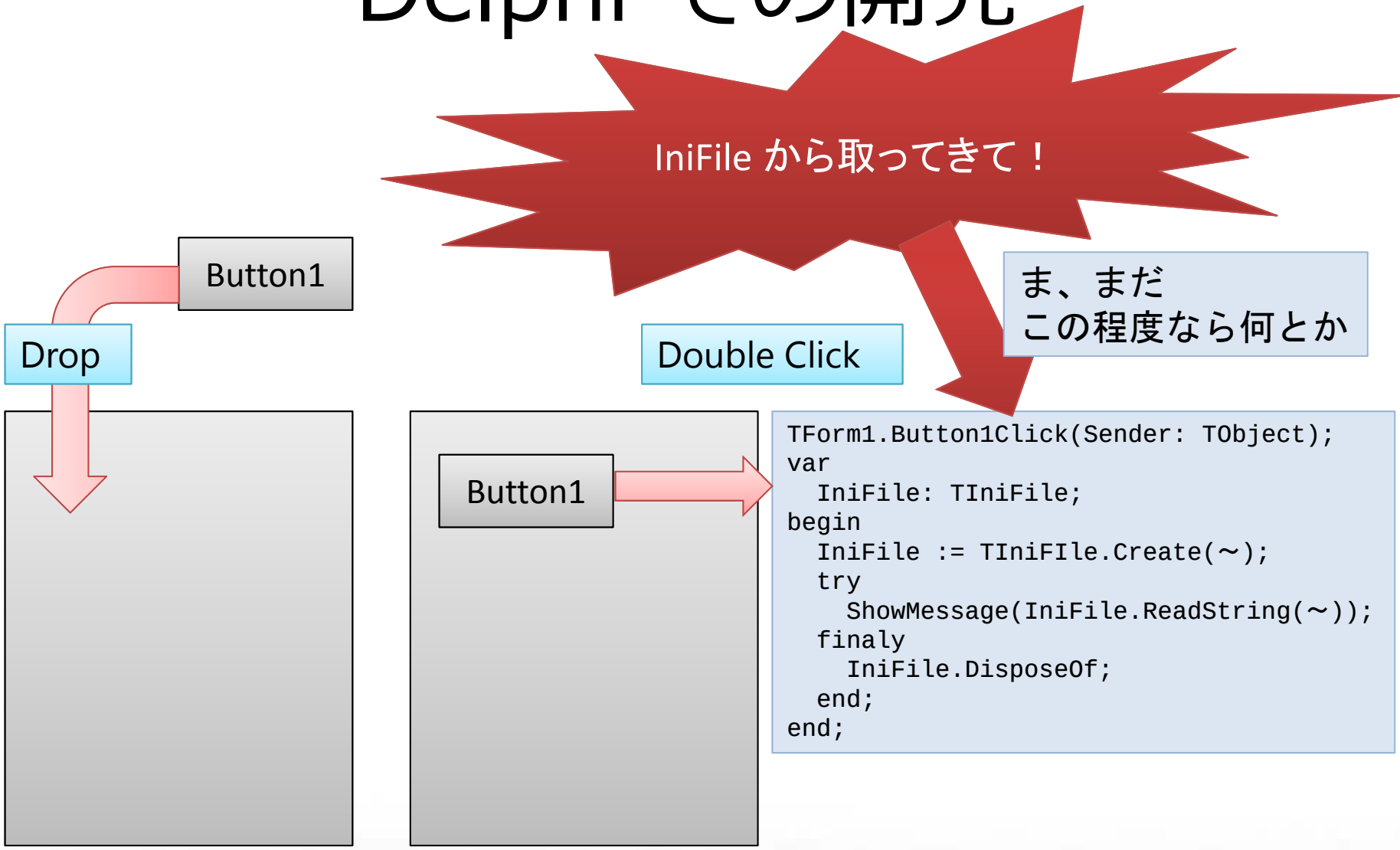



```
TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('皐月様');
end;
```

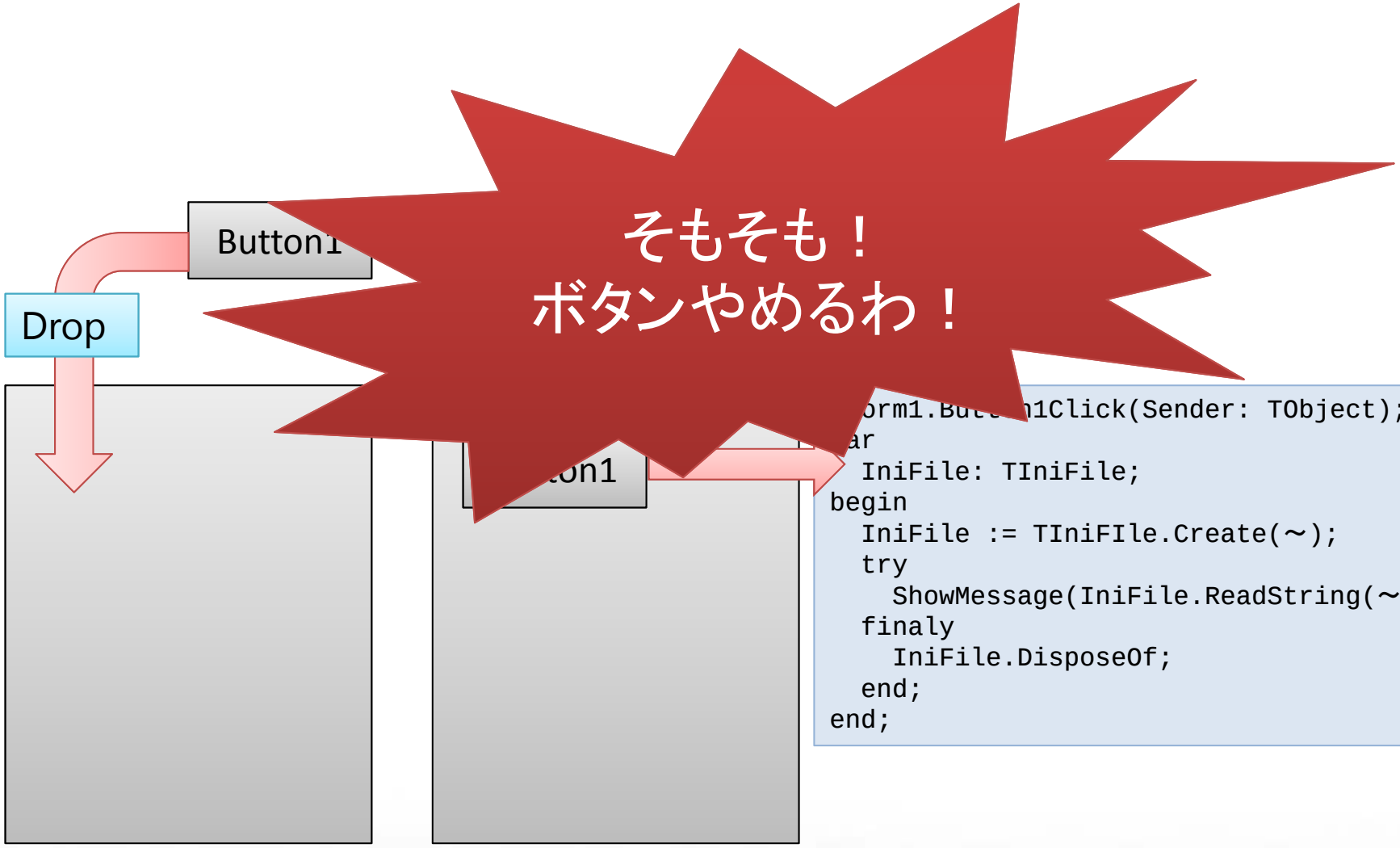
Delphi での開発



Delphi での開発



Delphi での開発

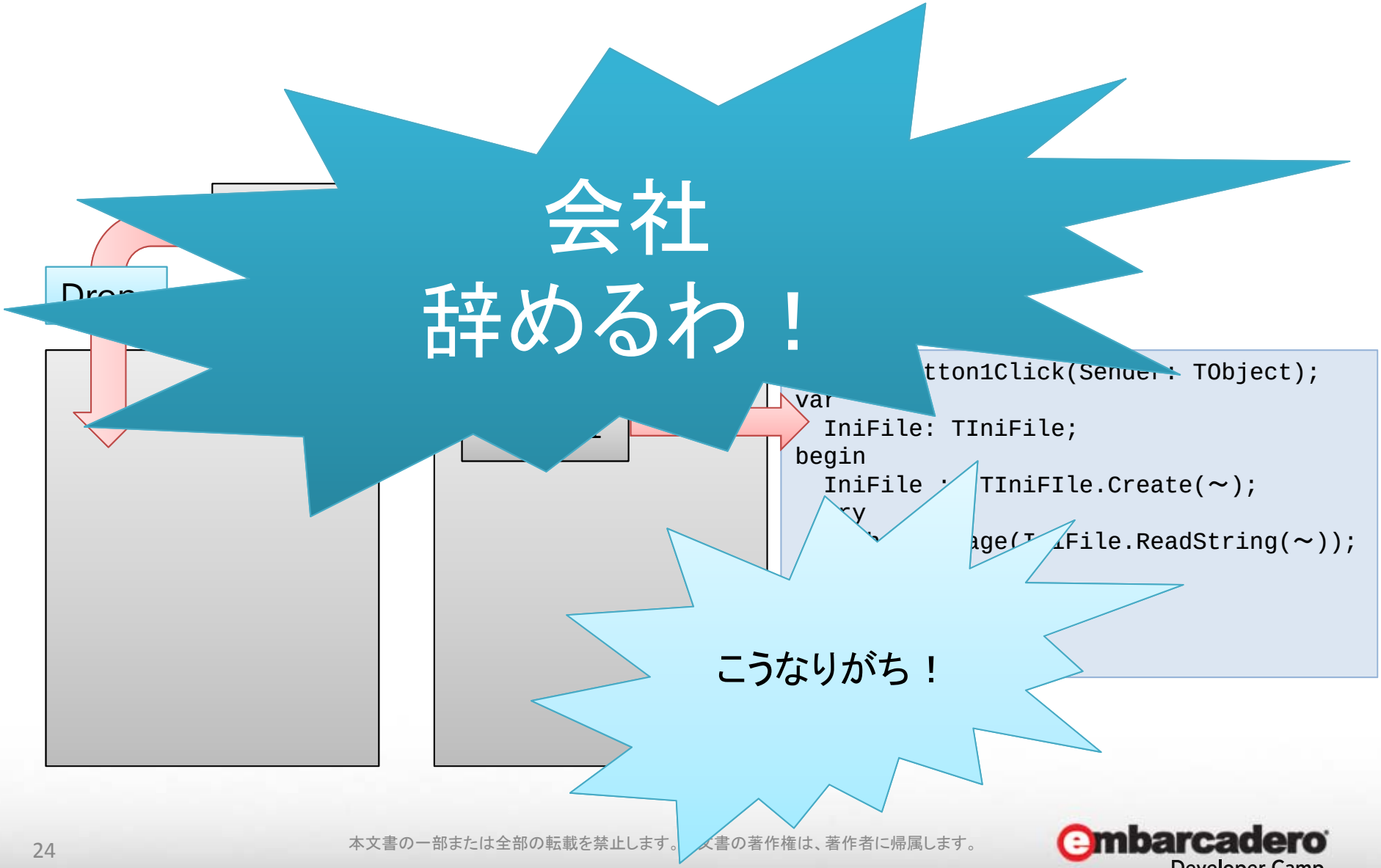



```
orm1.But  n1Click(Sender: TObject);
ar
  IniFile: TIniFile;
begin
  IniFile := TIniFIle.Create(~);
  try
    ShowMessage(IniFile.ReadString(~));
  finaly
    IniFile.DisposeOf;
  end;
end;
```

Delphi での開発


```
  orm1.But  n1Click(Sender: TObject);
ar
  IniFile: TIniFile;
begin
  IniFile := TIniFIle.Create(~);
  try
    ShowMessage(IniFile.ReadString(~));
  finaly
    IniFile.DisposeOf;
  end;
end;
```

Delphi での開発



Delphi での開発

- コードと UI が密接に関連している
  - UI の変更でコードが全て吹っ飛ぶ可能性を秘めている
  - 当然 UI が異なるマルチデバイス対応では、完全に書き直しが発生する

マルチデバイス時代の設計手法

Delphi でのマルチデバイス開発

- Windows / OSX
  - この２つの組み合わせだけなら、あまり考えなくて大丈夫
- iOS / Android
  - iPhone / iPad
    - この２つだけでも、解像度が違う
  - Android
    - 解像度地獄！！！

マルチデバイスの問題点

- 解像度が違う
- でも！**処理内容は大体同じ**
  - ただし、解像度やデバイスの能力によってできないことがある
    - Windows / OSX では GPS を提供していない場合が多い
    - iOS / Android では、移動体ゆえに通信環境が安定しない
    - 解像度に依存した処理

    など

Delphi でのマルチデバイス地獄

- 何も考えずに開発していると……
  - 各 OS 用にアプリを作っちゃう
  - 各 OS 毎にフォームを作り分ける
    - でも、ロジックをフォームに書きがち

- どちらにせよ！
  - 各 OS 毎に同じような処理を複数書いていたらムダ！ムダだけならまだ良い！
    - 同じバグが何カ所にもおおおおお！となる可能性が。

マルチデバイスの問題点

- そもそも、処理内容が大体同じだったら……

UI とロジック
分けてみよう！

MVVM の登場

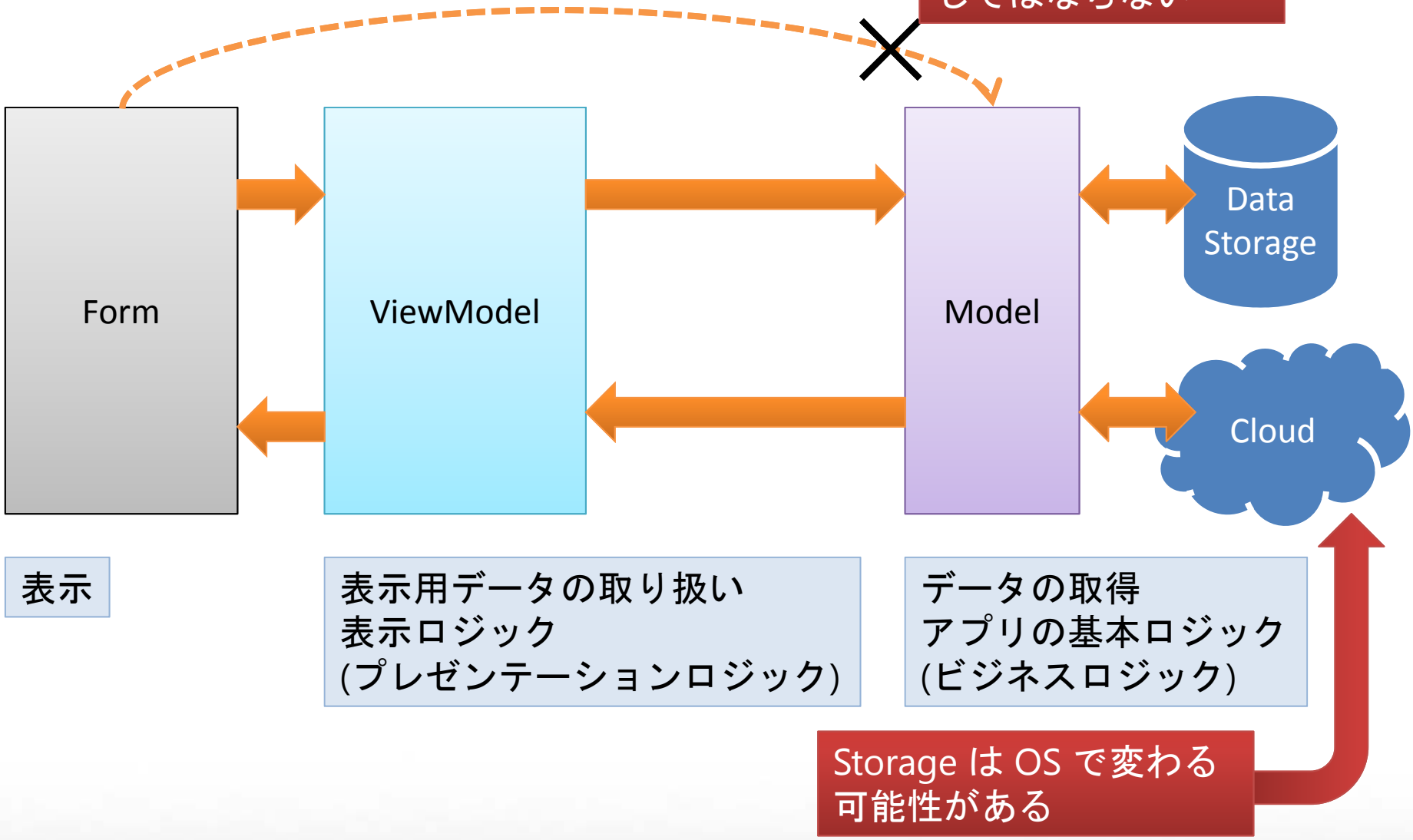
マルチデバイス時代の設計手法

- MVVM
  - **M**odel
  - **V**iew
  - **V**iew**M**odel
  - 比較的新しい設計手法
    - 元々は Microsoft の研究者が XAML との兼ね合いで考え出した
  - **最高にロック！**

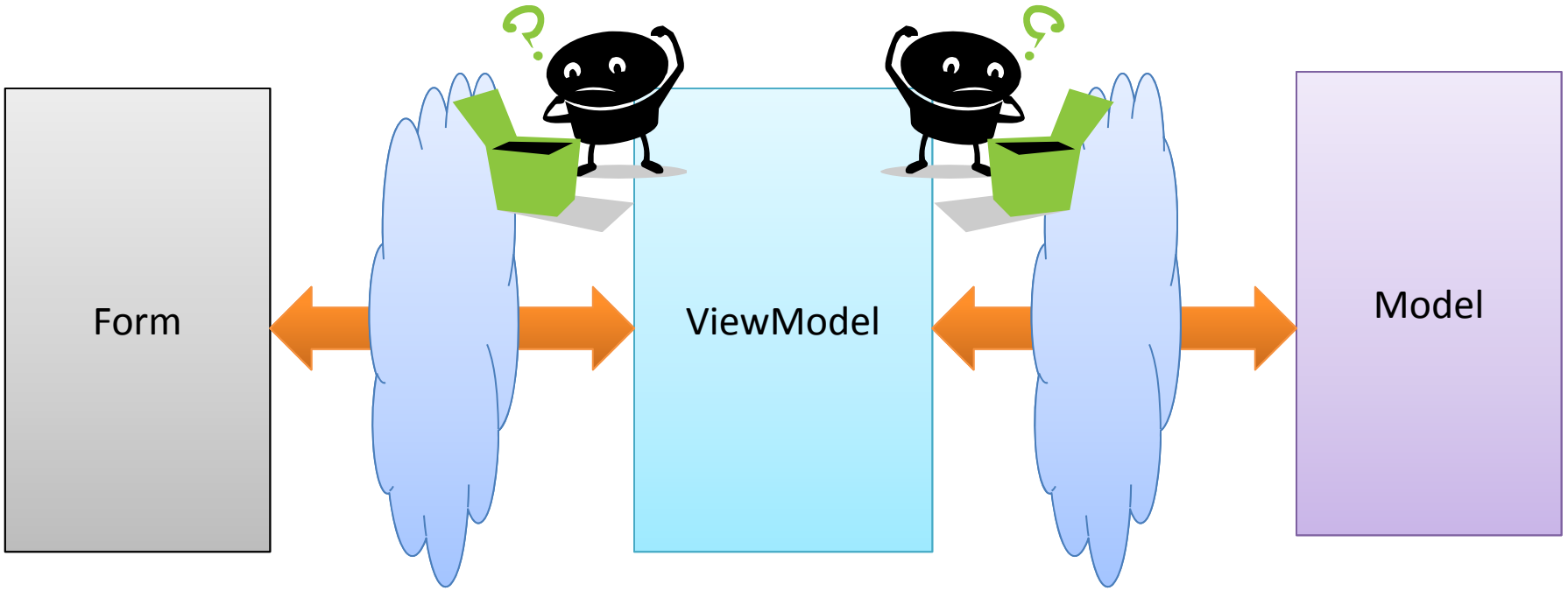
MVVM

- Model
  - データを提供するデータプロバイダ
  - ビジネスロジック（アプリの基本ロジック）
- View
  - データを可視化する UI 担当部分
- ViewModel
  - Model と View の橋渡し役
  - プレゼンテーションロジック（表示に関わるロジック）
  - Model / View の実装については「知らない」立場を取る

MVVM

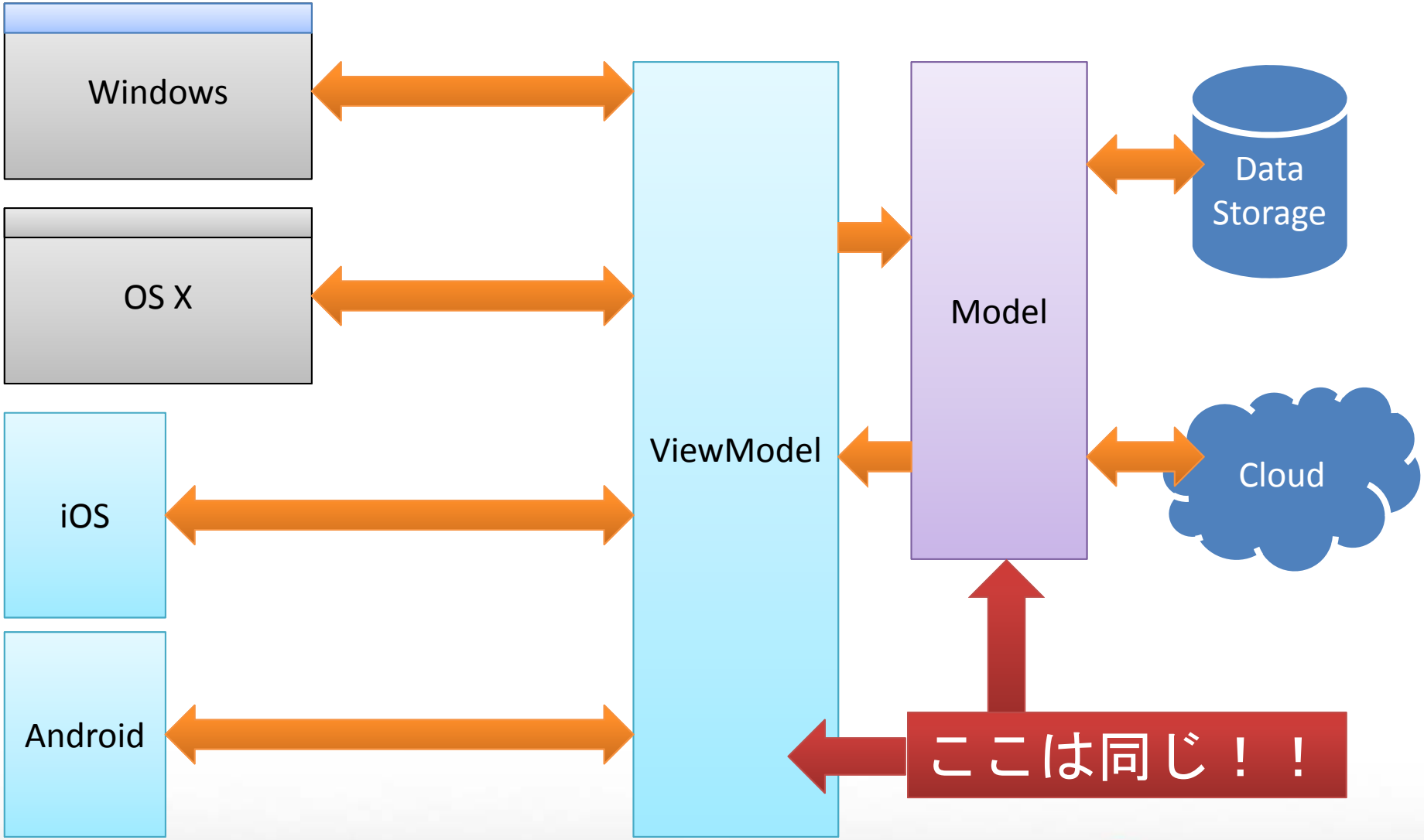


MVVM



ViewModel はそれぞれの実装を知らない！
そのため、View と Model を抽象化する作用がある！

マルチデバイスの MVVM

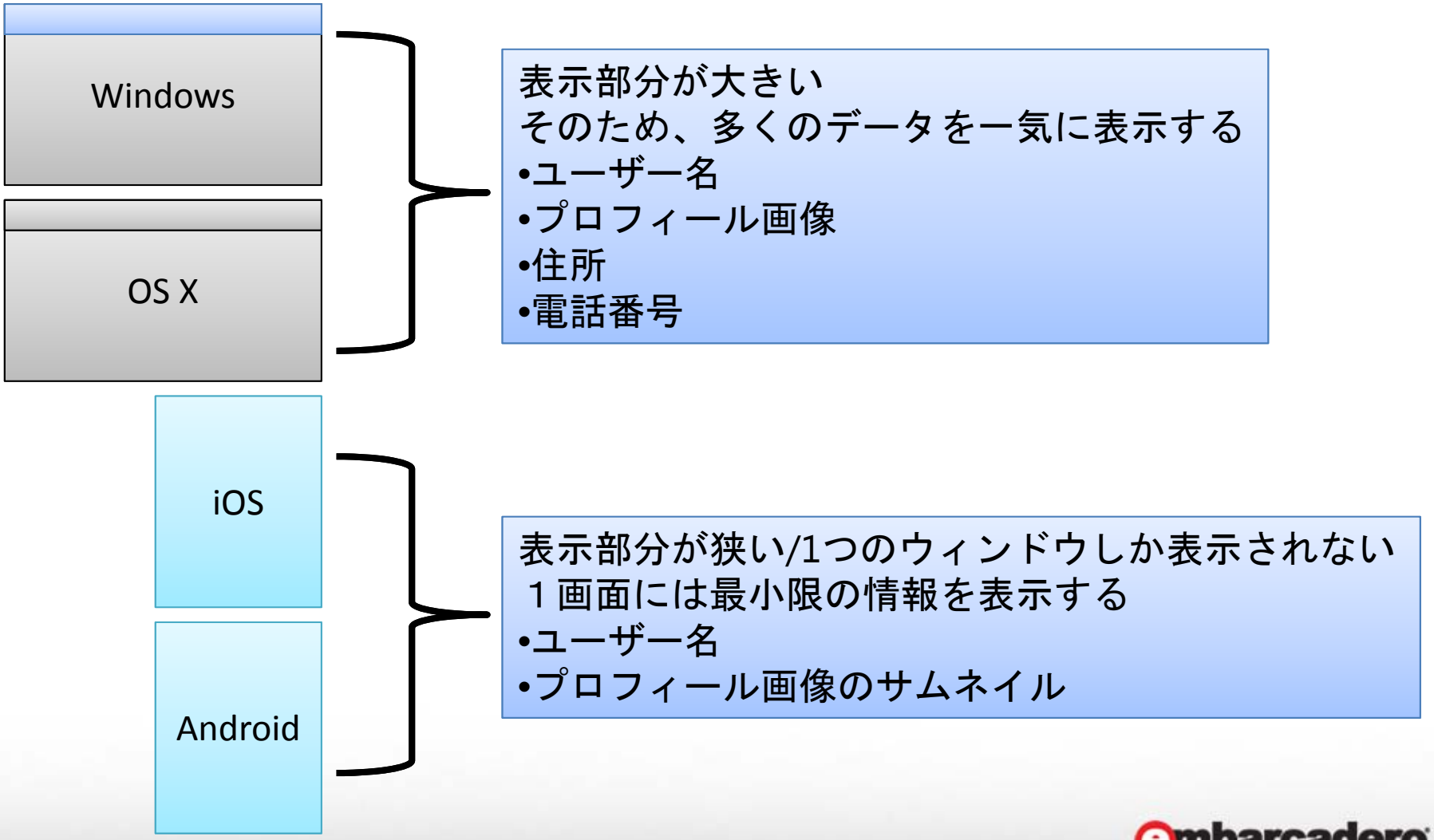


ViewModel まとめ

- ViewModel は View に必要な情報を渡す
  - データ
    - Model で提供されているデータなど
  - 状態
    - 非永続化データの保持（Model に保存しないデータ）
      - ログインしているか？など
  - 通知（イベント）
    - GCM / APNS からの通知など
  - 表示に関するロジック
    - あるデータを表示するかどうか？など

マルチデバイスの MVVM

ユーザーデータの表示



マルチデバイスの MVVM

ユーザーデータの表示

- Model の必要な要素
  - ユーザー名の提供
  - プロフィール画像の提供
  - 住所の提供
  - 電話番号の提供

マルチデバイスの MVVM

ユーザーデータの表示

- ViewModel の必要な要素
  - Model が提供している要素
  - プロフィール画像サムネイルの提供
    - ViewModel がサムネイルを提供するなど Model の足りない部分を補うことも
  - ログイン・ログアウト機構とその状態
    - イベントの発行
  - データ提供の可否判定
    - 未ログインユーザーの場合、プロフィール、住所、電話番号は提供しない

マルチデバイスの MVVM

ユーザーデータの表示

- View
  - 必要な情報を ViewModel に問い合わせる
  - 取得できた情報を表示する
    - 表示方法は各デバイスに適した形にする
  - 取得できなかった情報の表示
    - 未ログインユーザーの場合に、メッセージを出すなど
  - イベントハンドラの実装
    - ログインイベントなどを受け取って表示する

Delphi での MVVM の具体例

```
// いままでは直接データ（Model）や状態（ViewModel）を扱っていた
// しかし！ IniFile は Windows にしか存在しないし、マルチデバイスになったら困る……
// （TIniFile は Platform フリーで使えるけど、OS 固有の形式になっていないのでかっこ悪い）
procedure TForm1.Login;
begin
  // 処理
end;

procedure TForm1.Logout;
begin
  // 処理
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  IniFile: TIniFile;
begin
  if (FIsLoggedIn) then begin
     IniFile := TIniFile.Create('～');
     Label1.Text := IniFile.ReadString('UserName');
     IniFile.DisposeOf;
   end
   else
     Label1.Text := '表示できません';
end;
```

Delphi での MVVM の具体例

```
// MVVM では View には最低限のコードしか書かない！
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Text := ViewModel.UserName;
end;

// ViewModel に View に渡すデータを定義する
// そのデータは Model から取得する
// また状態に関する動作も ViewModel に任せる
type
  TViewModel = class
  private
    FModel: TModel;
    FIsLoggedIn: Boolean;
    FOnLogIn: TNotifyEvent;
    FOnLogOut: TNotifyEvent;
    function GetUserName; // データを提供して良いか判断し FModel.GetUserName を呼ぶ
  public
    procedure Login;
    procedure Logout;
    property UserName: String read GetUserName;
    property IsLoggedIn: Boolean read FIsLoggedIn;
    property OnLogIn: TNotifyEvent read FOnLogIn write FOnLogIn;
    property OnLogOut: TNotifyEvent read FOnLogOut write FOnLogOut;
  end;
```

Delphi での MVVM の具体例

```
// プラットフォームごとに TModel を作り OS 依存コードを分ける
// どれを使うかは ViewModel に判断させるか、Model 自身に判断させる
// FireMonkey では比較的 Model 自身に判断させる手法が取られている（ViewModel に判断させると抽象化度が下がる）
type
  TModel = class
  protected
    function GetUserName; virtual; abstract;
  public
    class function CreateByOS: TModel; // こんな風にすると ViewModel は完全に Model の実装から分離される
  public
    property UserName: String read GetUserName;
  end;

  // Windows では IniFile からデータを取得する
  TModelWIndows = class(TModel)
  protected
    function GetUserName; override;
  end;

  // OSX では plist からデータを取得する
  TModelOSX = class(TModel)
  protected
    function GetUserName; override;
  end;
```

MVVMでの設計

MVVMによるアプリケーション設計

- 何をどこに持たせるかの大まかな指針
  - **Model**
    - アプリケーションの根幹に関わるロジック
    - 永続化データの提供と保存
  - **ViewModel**
    - 表示に関わるロジック
    - Model データの受け渡し
      - この際にチェックを入れたりする
    - 非永続化データの提供
  - **View**
    - 完全に表示だけ
    - それ以外の部分全て ViewModel / Model に任せる

MVVMによるアプリケーション設計

- 実装が分かれると
  - チーム分けが可能
    - **コード設計**
      - Model チーム
      - ViewModel チーム
    - **UI / UX 設計**
      - View チーム
      - 可能な限り、**UI / UX の専門家をチームに入れる**

MVVMによるアプリケーション設計

- 新人の段階では
  - –View を任されると良いかも知れない
    - ロジックを考える必要が無い
    - 受け取ったデータを表示するだけで済む
      - –作業領域が少ない
    - 上司より **iOS / Android などに詳しい**！
  - –積極的に上司に MVVM を薦めて、View から入って見て下さい！

テストについて

- MVVM によるメリットの1つとしてテストのしやすさがあります
  - GUI アプリケーションの問題は UI が関わるためテストがしづらい！
  - MVVM を使うと、ロジック部の自動テストが可能に！
    - **Model テスト可能！**
    - **ViewModel テスト可能！**
    - View はちょっと難しい！
      - しかし非常に単純化されれば、そもそもバグが混入しづらい！

まとめ

マルチデバイス時代の設計

- UI が異なる
  - 解像度
  - 文化
    - 「OK」「キャンセル」の位置とか

- MVVM の利用
  - 解像度とロジックの分離
    - 設計がしやすい
  - View は最大限に簡素にする
  - テストのしやすさ