10ヵ月でHDゲームを開発する方法 ～龍が如くを支えたテクノロジ～

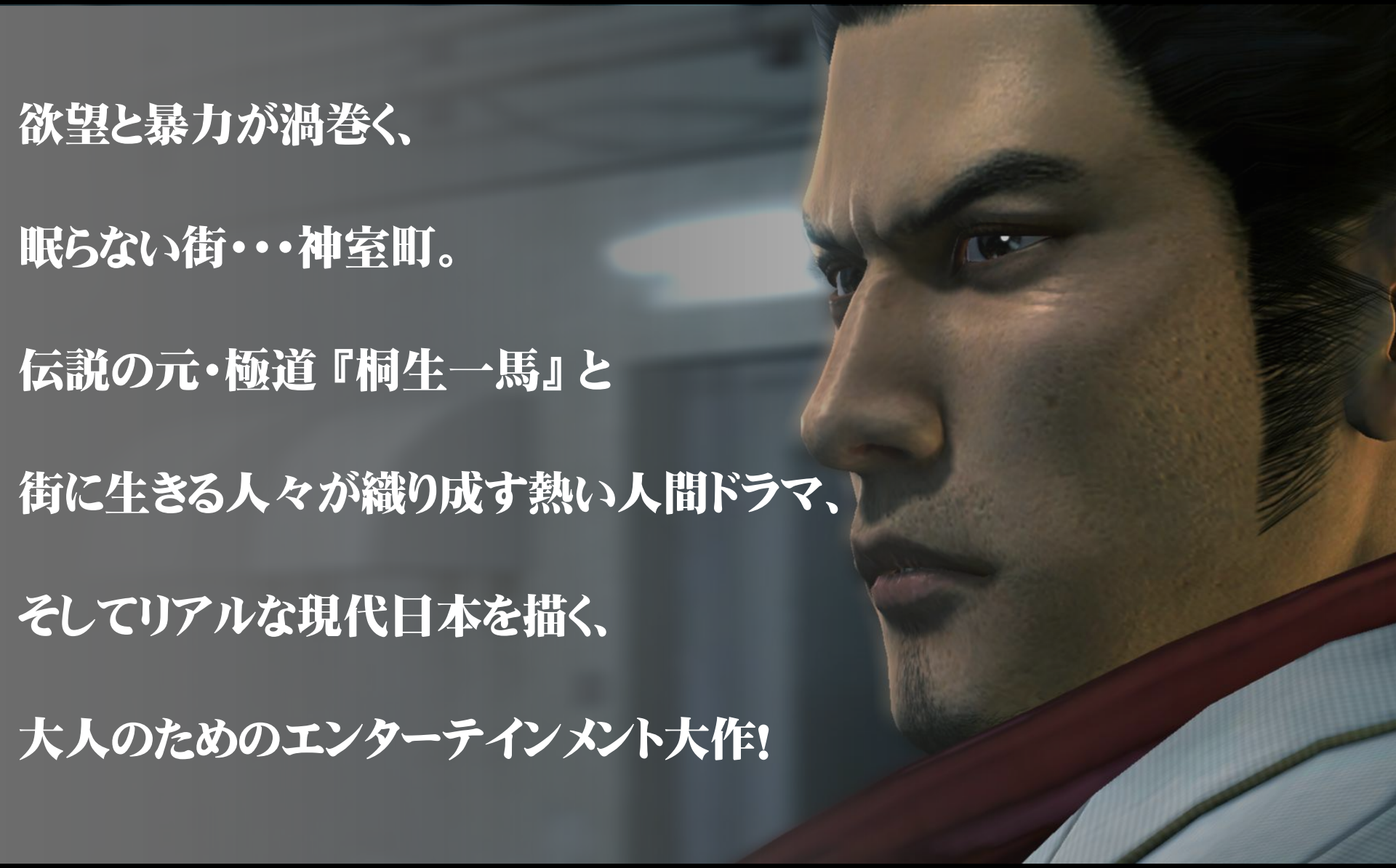
(株)セガ　第一CS研究開発部
加来 徹也
厚 孝
時枝 浩司

講演の前に

HDゲーム
→「ハイエンドプラットフォーム向けゲーム」

発表資料はCEDECのウェブサイトにて
公開予定（http://cedec.cesa.or.jp/2010/）

「龍が如く」とは



欲望と暴力が渦巻く、

眠らない街・・・神室町。

伝説の元・極道『桐生一馬』と

街に生きる人々が織り成す熱い人間ドラマ、

そしてリアルな現代日本を描く、

大人のためのエンターテインメント大作!

「龍が如く」シリーズの歴史

- PS2

2005年12月　龍が如く
2006年12月　龍が如く2

- PS3

2008年3月　龍が如く 見参！
2009年2月　龍が如く3
2010年3月　龍が如く4～伝説を継ぐもの～

はじめに

- 「龍が如く」シリーズは、毎年リリースする事が使命だったタイトル

  →開発期間は**10ヶ月**

- クオリティは維持（むしろ向上）を要求される

- プログラマとして出来る事は全てやる！

「それ以前」に大切なこと

- 企画を固め、きちんとした仕様書がある
- 効率よく(あるいは力業で)リソースを揃える

↑

これらについては割愛

全ての仕様が決定し、
全てのリソースが揃っている

お話の流れ

- 第１部・開発環境編
  ー事前に**様々な工夫**をして時間短縮
- 第２部・基礎技術（ライブラリ）編
  ー**PC**でPS3と同様の開発を行う、
  　マルチプラットフォームライブラリ
- 第３部・デバッグ編
  ー各種デバッグ用機能と**オートテスト**

第1部・開発環境編

株式会社セガ
第一CS研究開発部
加来 徹也

第一部でお話する内容

「時間」をいかに削る事が出来るか？

→開発環境の改善で出来ることを考える

- リソース管理の高速化
- ビルド時間の短縮
- PCでの代替開発
- コーディングしやすい環境づくり

第一部でお話する内容 1

- リソース管理の高速化
- ビルド時間の短縮
- PCでの代替開発
- コーディングしやすい環境づくり

リソース管理の高速化 (1/4)

- HDゲームではLANを通して数百GBのコピーが発生する

  ー龍が如く4では…

    実機データ：21.4GB（約19000ファイル）

    中間データ：398GB（約34万3千ファイル）

  →やむを得ないコスト

※ソースコードは Subversion で管理

リソース管理の高速化 (2/4)

- 高性能なサーバー/ネットワークの導入
  - ー導入コストを惜しまない
  - ー信頼性の高いものをチョイス

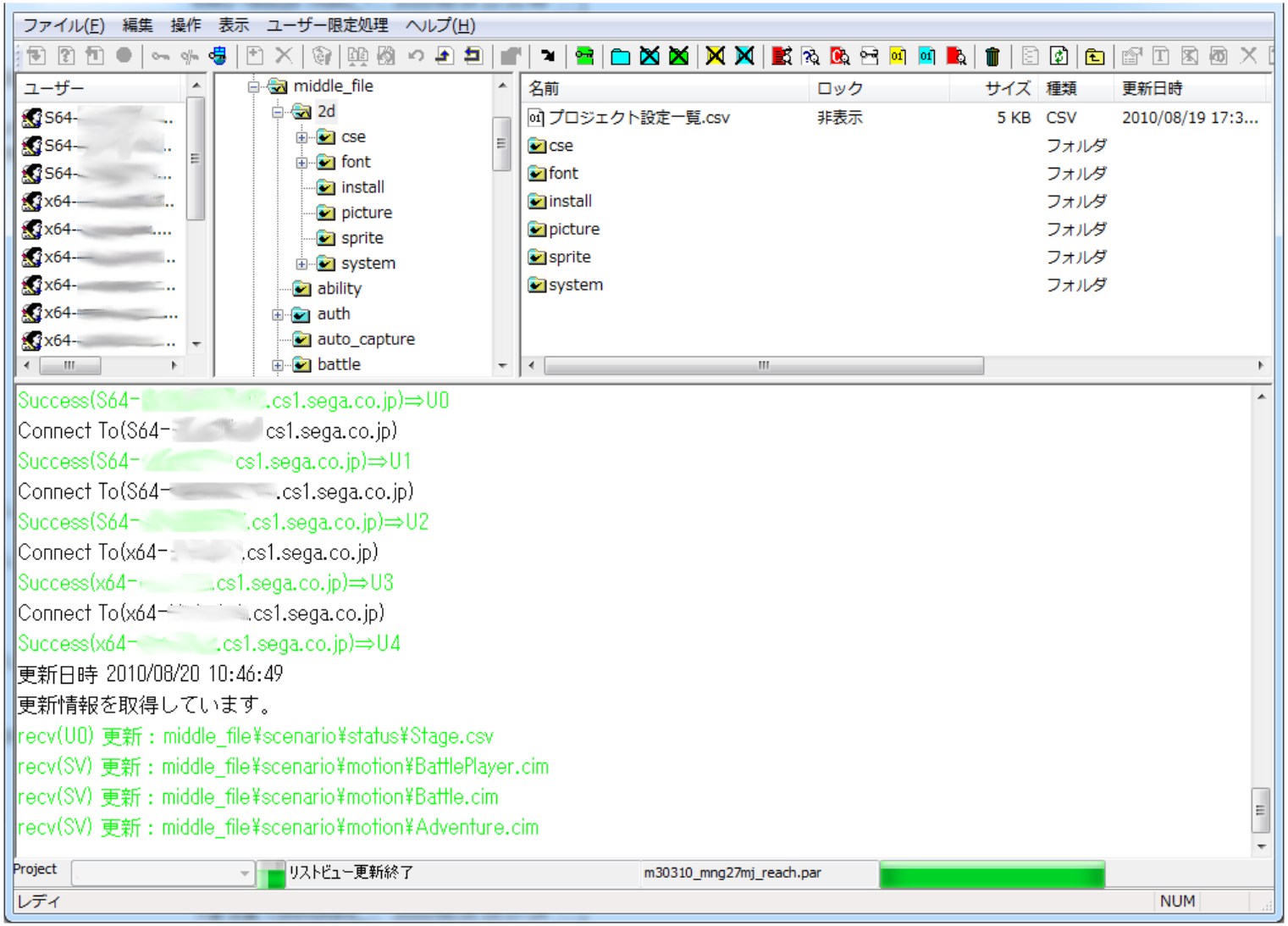
リソース管理の高速化 (3/4)

リソース管理ツール”**TiVersion**”を作成

- 最大**5ノードとP2P接続**を使用
  ーサーバーの負荷を軽減
- 職種に応じた**「必要ファイル」の選択**
  ーファイル転送量の根本的な削減
- バージョン管理機能は**必要最低限**
  ーバックアップはサーバーで行う

リソース管理の高速化 (4/4)

TiVersionの外観

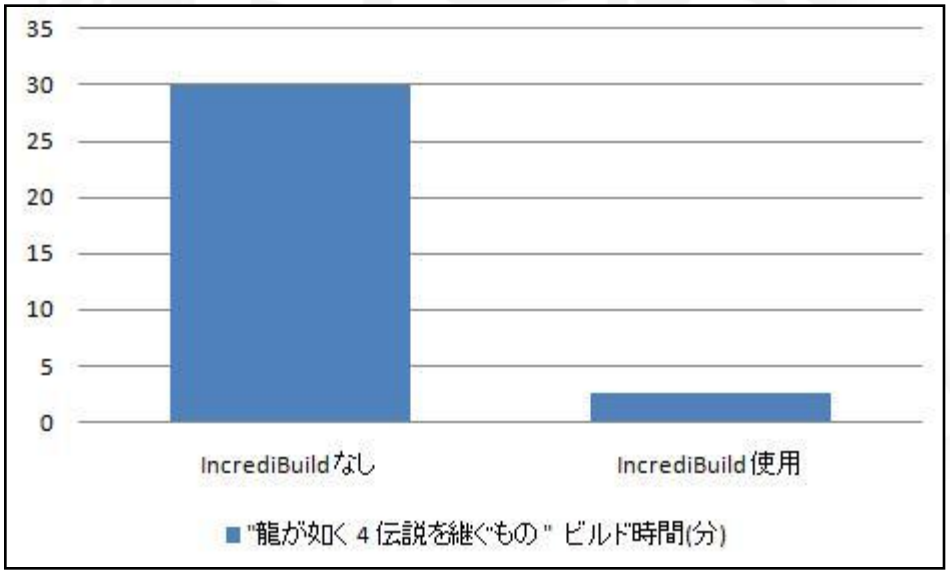


第一部でお話する内容 2

- リソース管理の高速化
- ビルド時間の短縮
- PCでの代替開発
- コーディングしやすい環境づくり

ビルド時間の短縮 (1/2)

- HDゲームはソースコードの量も膨大
  ー「龍が如く4」は120万行
- 分散ビルドで解決
  ー **Xoreax社「IncrediBuild」を使用**



ビルド時間が1/10に

ビルド時間の短縮 (2/2)

- リンク時間の短縮＝PCの高性能化
  - **64bitOS**(WindowsXP、7)の早期導入
  - メモリの増設(**4GB～6GB**)
  - 早いローカルストレージ

第一部でお話する内容 3

- リソース管理の高速化
- ビルド時間の短縮
- PCでの代替開発
- コーディングしやすい環境づくり

PCでの代替開発 (1/2)

PS3開発機の問題点

- 起動時間の遅さ
  ープログラムの開発機へのダウンロード時間
  ー**数十秒**だが、累積すると**膨大**な時間となる
- デバッグ機能の不足
  ー**きめの細かい**デバッグを行いたい
- 開発機のコスト
  ー多人数での開発では**大きなコスト**になる

PCでの代替開発 (2/2)

- 素早い起動
  ーPS3開発機へのダウンロード時間が**無い**
- デバッグの容易さ
  ーPS3開発機では**実現困難**な様々な機能
- PCは全員が所持
  ーコスト安と、**大量のPC**による同時デバッグ

(詳しくは第3部にて)

第一部でお話する内容 4

- リソース管理の高速化
- ビルド時間の短縮
- PCでの代替開発
- コーディングしやすい環境づくり

コーディングしやすい環境 (1/2)

- 緩いコーディングルール
  ーー一定の基準の中でなら自由に
- 内製ライブラリ
  ーPCとPS3で同じ動作を実現
  ー質問しやすい、修正が迅速、理解しやすい

(詳しくは第2部にて)

コーディングしやすい環境 (2/2)

- チームの**コミュニケーションと育成**
  - ベテランから若手まで、最大**35名**
  - 「**プレイスポット**」で若手の育成
  - パーティションを立てず**雑談**しやすく
    - →質問や報告など**気軽**に行う

第2部・基礎技術（ライブラリ）編

株式会社セガ
第一CS研究開発部
厚 孝

ここでお話しする内容

- 開発用マルチプラットフォームライブラリ
  - 共通のソース記述
  - 共通のメモリ使用量
  - 共通の挙動
- シェーダの運用
  - シェーダの作成
  - シェーダの名前
  - シェーダのアサイン
- PlayStation 3での最適化
  - 最適化の指針
  - 龍が如くのパフォーマンスボトルネック
  - SPUの利用

ここでお話しする内容

- 開発用マルチプラットフォームライブラリ
  - 共通のソース記述
  - 共通のメモリ使用量
  - 共通の挙動
- シェーダの運用
  - シェーダの作成
  - シェーダの名前
  - シェーダのアサイン
- PlayStation 3での最適化
  - 最適化の指針
  - 龍が如くのパフォーマンスボトルネック
  - SPUの利用

開発用マルチプラットフォームライブラリ

・開発用とは？

製品リリースを目的としたものではなく、
あくまで、開発効率の向上を目的としたもの

「PCでPS3の代替開発を行うためのライブラリ」

開発用マルチプラットフォームライブラリ

考慮した機能

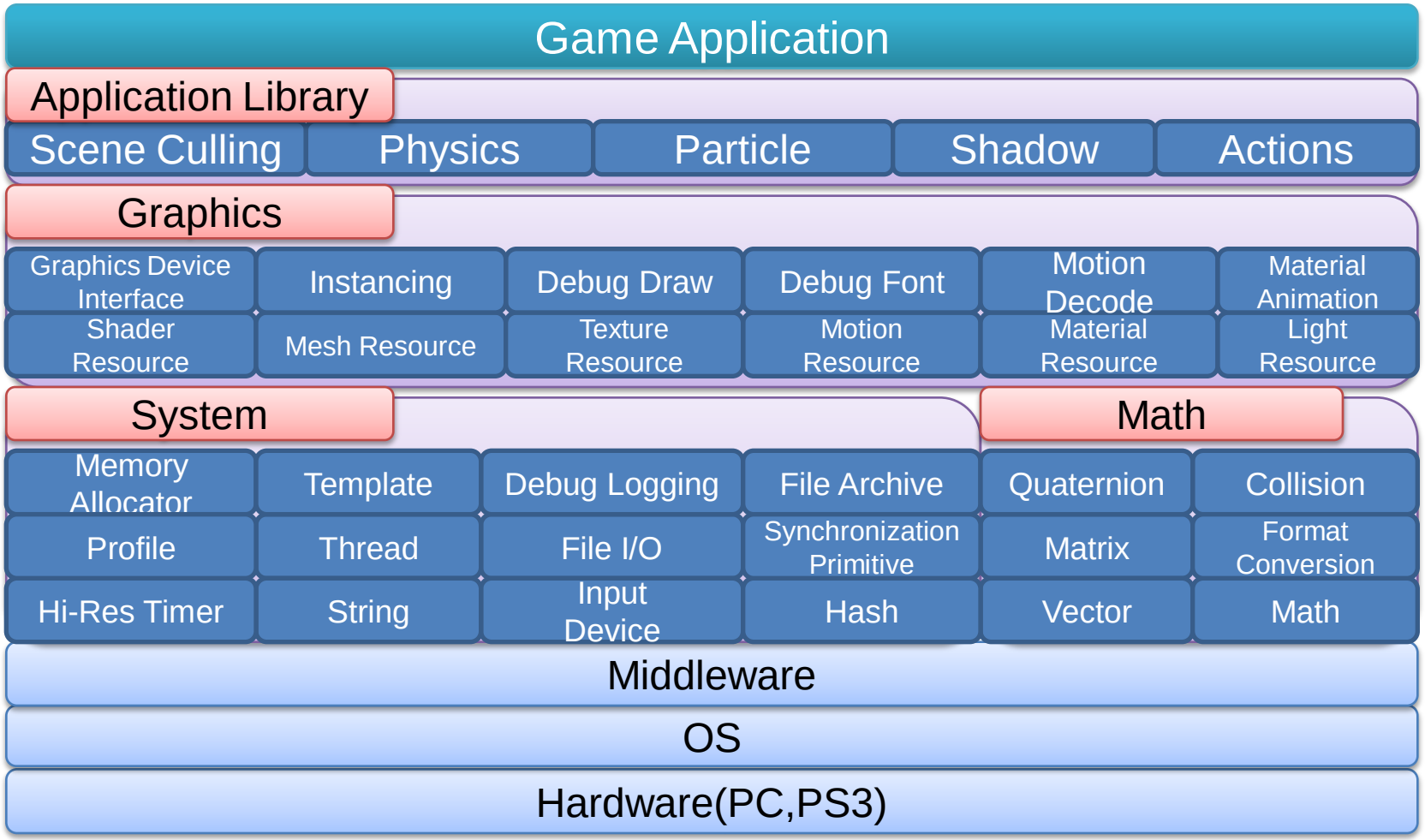
- 共通なソース記述
- 共通のメモリ使用量
- 共通の挙動
- 共通のバイナリデータ
  - PCではロード時にエンディアン解決
- DirectX9環境での動作
  - Widows Xpで動作
- 不正動作のASSERT処理
  - エラーレポートと組み合わせで早期の不正動作の検出

考慮していない機能

- PCでの汎用性
  - 特定のGPU環境でのみ動作
- PC動作の最適化
- DirectX10は未対応
- 動作速度のエミュレーション

開発用マルチプラットフォームライブラリ

サポートする範囲



共通のソース記述(1/2)

- 共通の型

プラットフォームやコンパイラに依存しない形で同じサイズを保証

・u32 （32bit 符号なし整数)
・f32　(32bit 浮動小数点数)
・vf128 (ベクトルfloat）
　など

vf128型とは？
vf128型はCPUのサポートするベクトルレジスタを使用する型
PC (SSE)
typedef __m128 vf128;
Play Station 3 (VMX)
typedef vector float vf128;

関数の引数と戻り値に使用する事でレジスタ渡しが行える。
inline関数との組み合わせで、const 参照渡しよりも良好な最適化結果が見込める

共通のソース記述(2/2)

- コンパイラ依存の記述をマクロで吸収
  - アプリケーションレベルでのソースコード記述を同一に
    - 強制インライン

      PC

      （そのまま）

      PS3

      ```
      #define __forceinline  inline  __attribute__((__always_inline__))
      ```

      記述例

      ```
      __forceinline void force_inline_method( void );
      ```

    - データ整列

      PC

      ```
      #define ALIGN_16( d )          __declspec(align(16) ) d
      ```

      PS3

      ```
      #define ALIGN_16( d )          d __attribute__((aligned(16) )
      ```

      記述例

      ```
      ALIGN_16( u32 aligned_data_tbl[128] );
      ```

共通のメモリ使用量

- 内部クラスのメモリ使用量を同一に

メモリ使用量の大きいプラットフォームに合わせる

```
class csbgl_surface_gs : public csbgl_surface
{
        #if !_PLAYSTATION3
            u8 m_reserved[ 44 - sizeof(csbgl_surface) ];
        #endif
};
```

同一に出来ない部分は分離しアプリ管理外メモリから確保

```
class cgs_ps{
    csbgl_pixelshader_gs* mp_ps_tbl[16]; ← 別のメモリアロケータから確保
};
```

ほとんど存在しないが、シェーダのバイナリだけは
共通化できなかったため例外的に対応

共通の挙動

- APIやハードの違いを吸収
  - PC上での挙動をPS3と同等に

フラグメントシェーダ内で
深度テクスチャからの深度値の取得

Nvidia製GPUでDX9上で深度値を取得

PS3と同様の描画パスを実行できる

```
float _tex2D_depth_to_float( sampler2D s , const float2 uv ){
	float4 uv_tmp = float4( uv , 0.0f , 1.0f );
	float f_scale = 0.5f;
	float f_now   = 0.5f;
	for( int i = 0 ; i < 24 ; i++ ){
		f_scale *= 0.5f;
		uv_tmp.z = f_now;
		float c = tex2Dproj( s0 , uv_tmp ).r;
		if( c > 0.0f ) f_now += f_scale;
		else f_now -= f_scale;
	}
	return saturate( f_now );
}
float _tex2D_depth_tex( sampler2D s , const float2 uv ){
#if _PLAYSTATION3
	return texDepth2D_precise( s , uv );
#elif _WINDOWS
	return _tex2D_depth_to_float( s , uv );
#endif
}
```

ここでお話しする内容

- 開発用マルチプラットフォームライブラリ
  - 共通のソース記述
  - 共通のメモリ使用量
  - 共通の挙動
- シェーダの運用
  - シェーダの作成
  - シェーダの名前
  - シェーダのアサイン
- PlayStation 3での最適化
  - 最適化の指針
  - 龍が如くのパフォーマンスボトルネック
  - SPUの利用

シェーダの作成

- シェーダの作成はプログラマ
  - シェーダ総数が増えすぎるのを抑制
  - 現実的でない冗長なシェーダを事前に排除
- シェーダソースコードの共通化
  - PC/PS3に加えて、DCCツール用のシェーダも共通化
  - 環境の違いはシェーダ共通ヘッダ内のプリプロセッサで解決
  - PS3のフラグメントシェーダでは静的分岐をエミュレーション
- シェーダコンパイラの違いを吸収するフロントエンドを用意
  - fxc.exe(PC),sce-cgc.exe(PS3)のコマンドライン呼び出しを共通に
  - シェーダビルドはVisual Studio上で行う

シェーダの名前

- シェーダはルール化された名前で取り扱い
  - 例) r_o1dst_b(vr)2d
- ブレンド式、テクスチャ種別、テクスチャ座標等から構成

r → RIGID頂点	_
_	b → ブレンド
o → opaque(不透明)	(vr) → 頂点カラーR要素
1 → テクスチャ座標1	2 → テクスチャ座標2
d → ディフューズマップ	d → ディフューズマップ
s → スペキュラマップ	
t → ノーマルマップ	

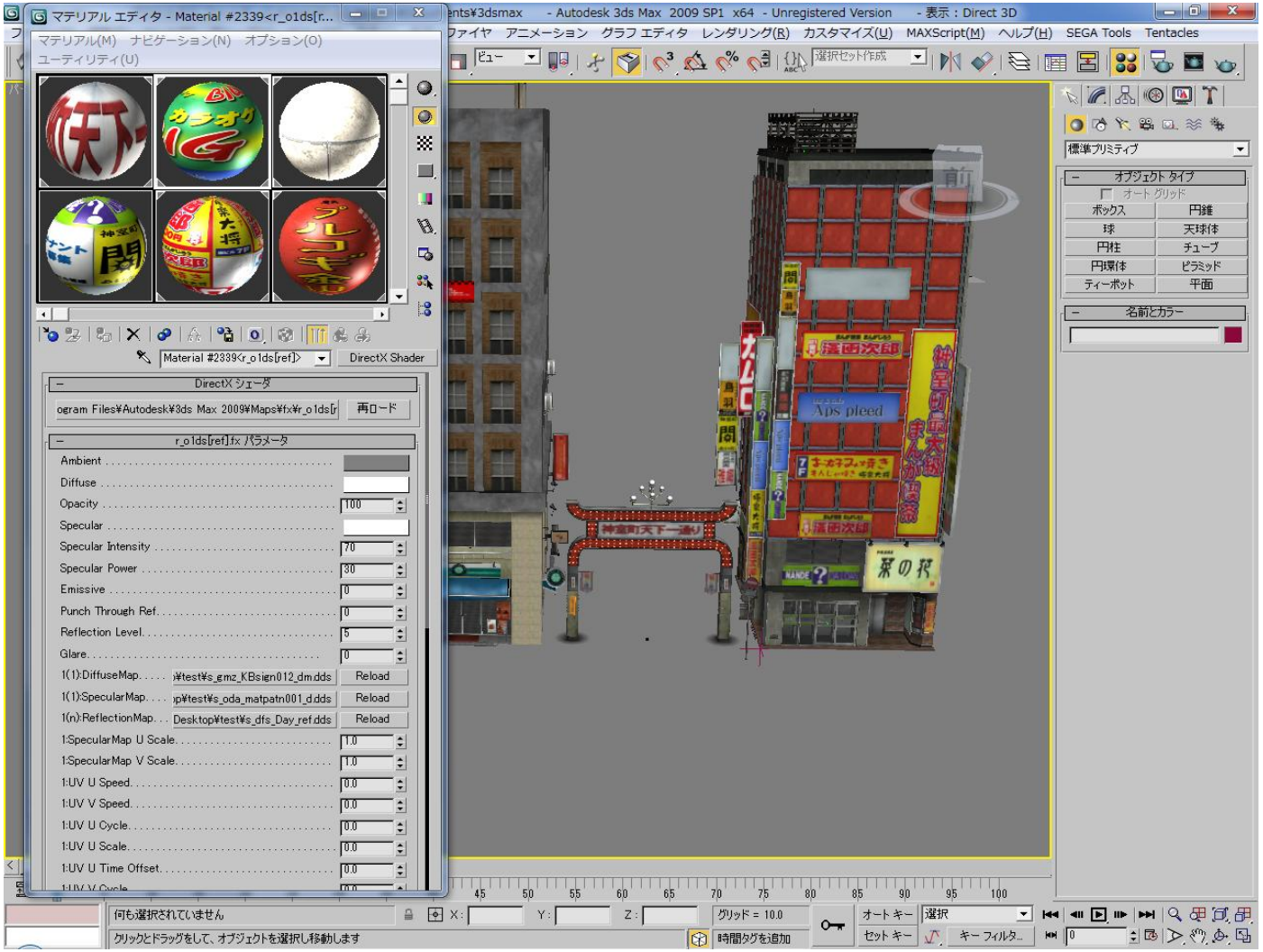
- 特殊動作をするものは[]で囲んだ機能名で付加
  - 例) r_o1dst[eye] →眼球用シェーダ
- 新規シェーダのリクエストはこの名前ルールを元に行う
- 最終的にシェーダのfxファイル名として使用

シェーダのアサイン(1/3)

- デザイナがDCCツール上でシェーダファイル名指定で行う
  - 3ds MaxとSOFTIMAGE®|XSIに対応
- シェーダのパラメータとして、マテリアルとテクスチャを設定
  - UVアニメーション、個別パラメータも設定可能
- ツール上のプレビューウィンドウ上で状態を確認
  - 基本的な質感は実機と同様
  - ポストエフェクト処理と、ダイナミックな光源は未対応
- モデルのマテリアル情報としてシェーダ名とパラメータを出力
  - 出力には独自フォーマットのエキスポータを使用

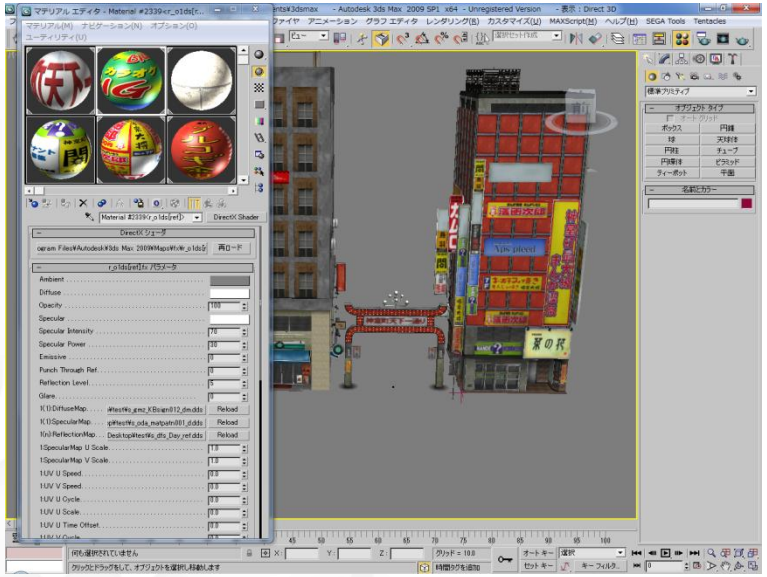
シェーダのアサイン(2/3)

- Autodesk 3ds Maxの場合



シェーダのアサイン(2/3)

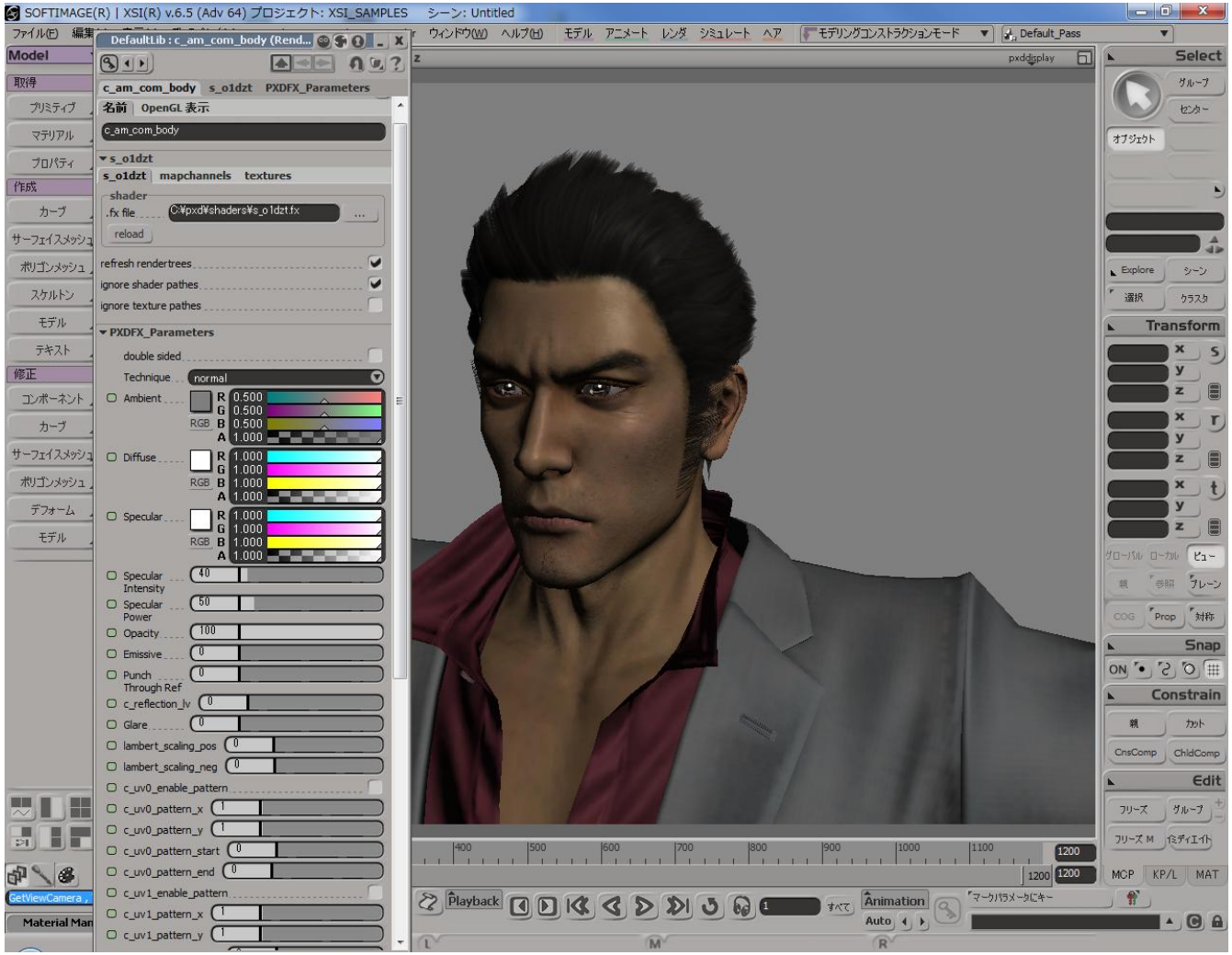
- Autodesk 3ds Maxの場合



- 標準のDirectXシェーダマテリアルを使用
- パラメータUIも標準機能の範疇で作成
- 3ds Maxのバージョンアップ時の対応が容易

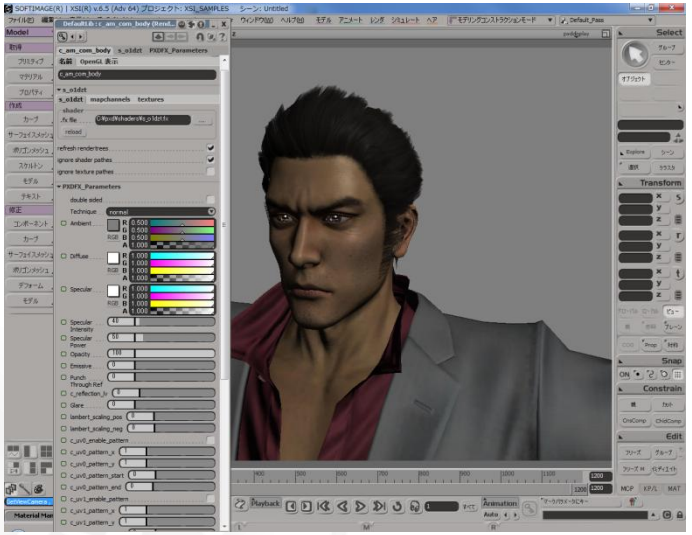
シェーダのアサイン(3/3)

- SOFTIMAGE®|XSIの場合



シェーダのアサイン(3/3)

- SOFTIMAGE®|XSIの場合



- XGSを使用して3ds Maxと同様のリアルタイムシェーダ環境を実現
- シェーダファイル自体は3ds Maxと同様のもの
- 実装コストはやや高い

ここでお話しする内容

- 開発用マルチプラットフォームライブラリ
  - 共通のソース記述
  - 共通のメモリ使用量
  - 共通の挙動
- シェーダの運用
  - シェーダの作成
  - シェーダの名前
  - シェーダのアサイン
- PlayStation 3での最適化
  - 最適化の指針
  - 龍が如くのパフォーマンスボトルネック
  - SPUの利用

最適化の指針

- PCとのマルチプラットフォーム環境は維持
  - バイナリ互換、同等の挙動
- ゲームアプリ開発者には出来るだけ負担をかけない
  - 中間ライブラリ層で吸収
- 効果の大きい最適化を施す
  - 実シーンの解析を行って最適化を行う

龍が如くのパフォーマンスボトルネック

龍が如くのパフォーマンスボトルネック

- オブジェクト数、マテリアル数が多い
  - 雑多な街並みの表現で増えがち
  - 同時に描画される人も多い
- 描画コマンドの発行数が非常に多い
  - 神室町の高負荷な場所では約12000回
  - 一つ一つは非常に軽量
  - GPU描画よりも先にPPUの描画セットアップ処理が間に合わない
- PPUの処理時間を短縮することが効果的

時間のかかるPPUの処理をSPUで実行

SPUの利用(1/2)

- モーション処理
  - 街中の大量の人のモーションの処理
  - ボーン数 約70 × 100人分
  - データの展開、階層構造解決、描画用セットアップを実行
- シーンのカリング処理
  - 各モデルの最小描画単位のAABB毎に最大128枚のオクルージョン平面を考慮したカリング処理を実行
  - 描画コマンド数を抑える効果もあり非常に有効
  - 1フレームのカリング対象のAABB数は1万個を超える
  - 前述のコール数はこの処理の後のもの

SPUの利用(2/2)

- 頂点シェーダのセット
  - 1フレームの頂点シェーダセット回数が1000回を超えるため、コマンドバッファコールのcall/retのオーバーヘッドがパフォーマンスに影響
  - RSXコマンドバッファに頂点シェーダコードをSPUのDMAで転送
- フラグメントシェーダのコンスタントパッチ
  - フラグメントシェーダにコンスタントレジスタを持たないRSXは、シェーダの書き換えでパラメータをセットする必要がある
  - SPUのLS内部に割り当てた仮想レジスタを元にパッチ処理を行う
  - 現在ではSCEからオフィシャルな手段が提供されているが、使用しているのはその提供前に独自実装したもの

第3部・デバッグ編

株式会社セガ
第一CS研究開発部
時枝 浩司

ここでお話しする内容

- ランタイムデバッグ機能の紹介
- プロジェクト運用を支援する技術の紹介

ここでお話しする内容

- ランタイムデバッグ機能の紹介
- プロジェクト運用を支援する技術の紹介

ランタイムデバッグのための主な機能

- デバッグカメラ（free tracking view）
- デバッグストップ（pause）
- デバッグウィンドウ（tweak window）
- シナリオセレクタ（scene selector）
- どこでもセーブ・ロード（all-time save and load）
- スクリーン・ムービーキャプチャー（scene capture）

ランタイムデバッグのための機能（1/4）

- デバッグカメラ（free tracking view）
  - 自由に視点を切り替えることのできるカメラ
    - ゲームカメラで目視しづらいバグの確認に用いる。
    - ゲームパッドで操作可能

ランタイムデバッグのための機能（2/4）

- デバッグストップ（pause）
  - 実行フェーズを一時停止する機能
    - 動きの速いもののチェックに利用することが多い。
    - 1フレームずつステップ実行も可能。

ランタイムデバッグのための機能 （3/4）

- デバッグウィンドウ（tweak window）
  - 各種パラメータ類を調整可能なGUI風ウィンドウ
    - ゲーム進行からグラフィックの調整まで幅広く活用。

ランタイムデバッグのための機能（4/4）

- シナリオセレクタ（scene selector）
  - 途中のシナリオから開始することが可能
- どこでもセーブ・ロード（all-time save and load）
  - どこでもセーブとロードすることが可能
    - バグ報告には出来る限り、添付してもらう。
- スクリーン・ムービーキャプチャー（scene capture）
  - ビットマップやAVIでキャプチャ可能。
    - バグ報告には出来る限り、添付してもらう。

ここでお話しする内容

- ランタイムデバッグ機能の紹介

- プロジェクト運用を支援する技術の紹介

プロジェクト運用を支援する技術のポリシー

- 非デバッグ期間の有効活用
- 休日、深夜時間の有効活用
- 人手をかけない

プロジェクト運用を支援する技術

- ゲームアプリケーションのデイリーリリース（daily release）
- 自動ビルドチェック（auto build check）
- エラーレポート機能（exception report）
- AutoTest（動的テスト）

プロジェクト運用を支援する技術

- ゲームアプリケーションのデイリーリリース

  （daily release）

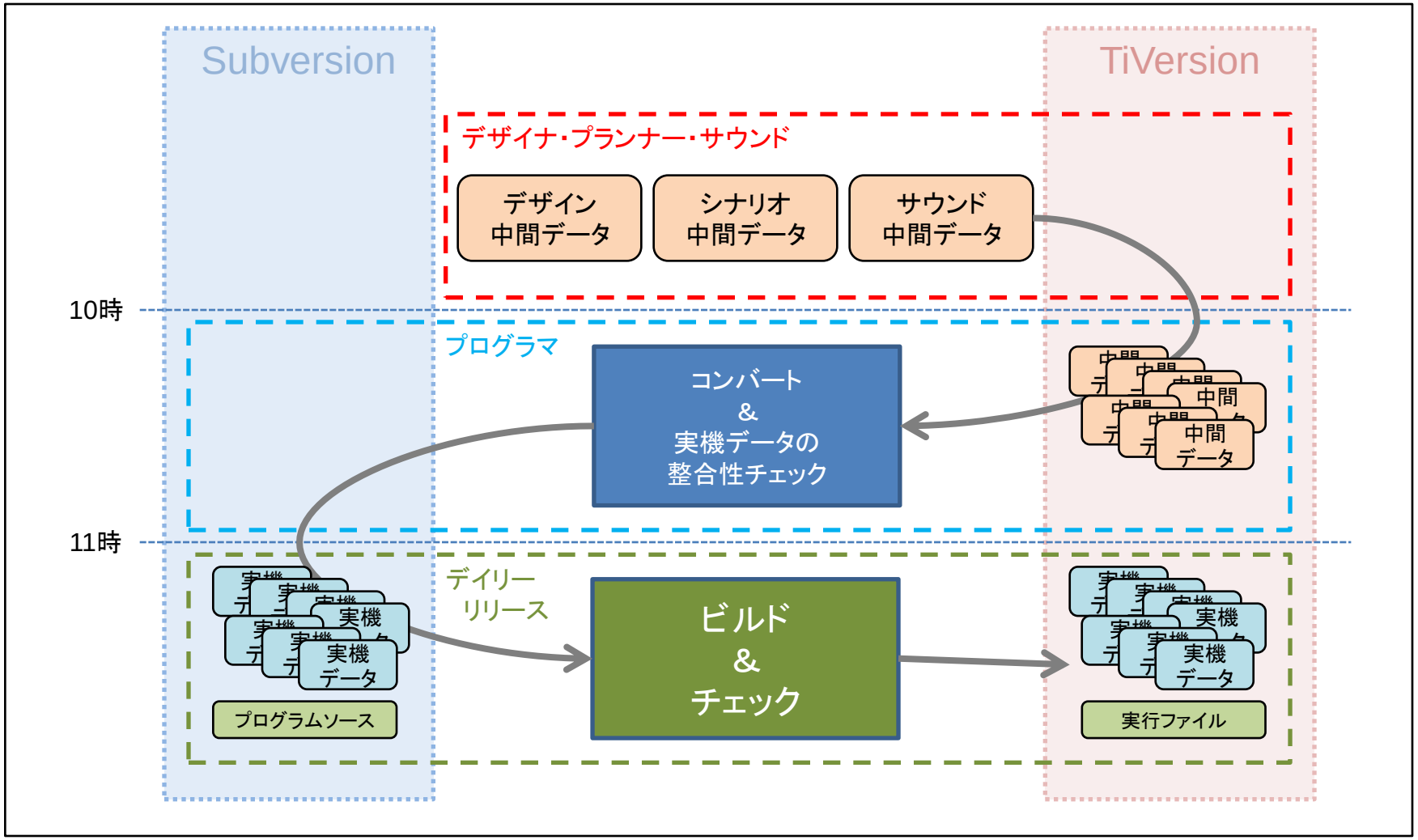
- 自動ビルドチェック（auto build check）
- エラーレポート機能（exception report）
- AutoTest（動的テスト）

ゲームアプリケーションのデイリーリリース(1/3)

- 1日1回、最新のゲームアプリケーションをビルド、チェックしてリリースする。
  - アプリケーションの配布はTiVersionで行う。

ゲームアプリケーションのデイリーリリース(2/3)

デイリーリリースの流れ



ゲームアプリケーションのデイリーリリース(3/3)

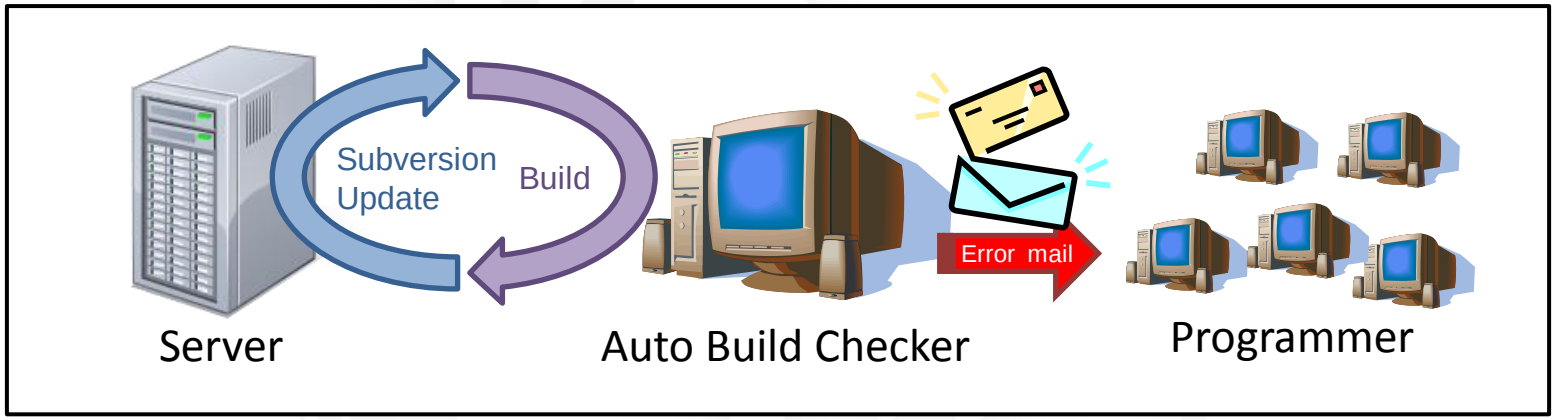
- 問題をプロジェクト全体へ伝播させるのを防ぐことができる。
  - エラーチェックを重点的に行うため。
- 規則正しいスケジュールで人が動く。
  - プロジェクトの規律向上。

プロジェクト運用を支援する技術

- ゲームアプリケーションのデイリーリリース（daily release）
- 自動ビルドチェック（auto build check）
- エラーレポート機能（exception report）
- AutoTest（動的テスト）

自動ビルドチェック（Auto Build Check）（1/3）

- 「Subversionの更新」と「ビルド」を繰り返し、ビルドエラーが発生した場合は「メールを送信」する。



自動ビルドチェック（Auto Build Check）（2/3）

ビルドチェッカーからのメッセージ例

```
PS3_Release
src/motion/ctrl_motionset.cpp(1497,3): error 145: a value of type "eCTRLTYPE" cannot be used to
initialize an entity of type "eSYNCNAME"
/*------------------------------------------
CodeName:       CiBuildErrorChecker
Revision:       (11424)
------------------------------------------*/
```

- ターゲットの差によるビルドエラー
  - 序盤はPC（debug）版、終盤はPS3（release）版
- コンパイラの差によるビルドエラー
  - PS3版（SNC）:“警告と注意を表示 (=2)”
  - PC版（VC）:“レベル4（/W4）”
- デバッグシンボルに関連するビルドエラー
  - #if __TOKIEDA__

自動ビルドチェック（Auto Build Check）（3/3）

- デイリーリリース環境で正常にビルドができる状態を維持できる。

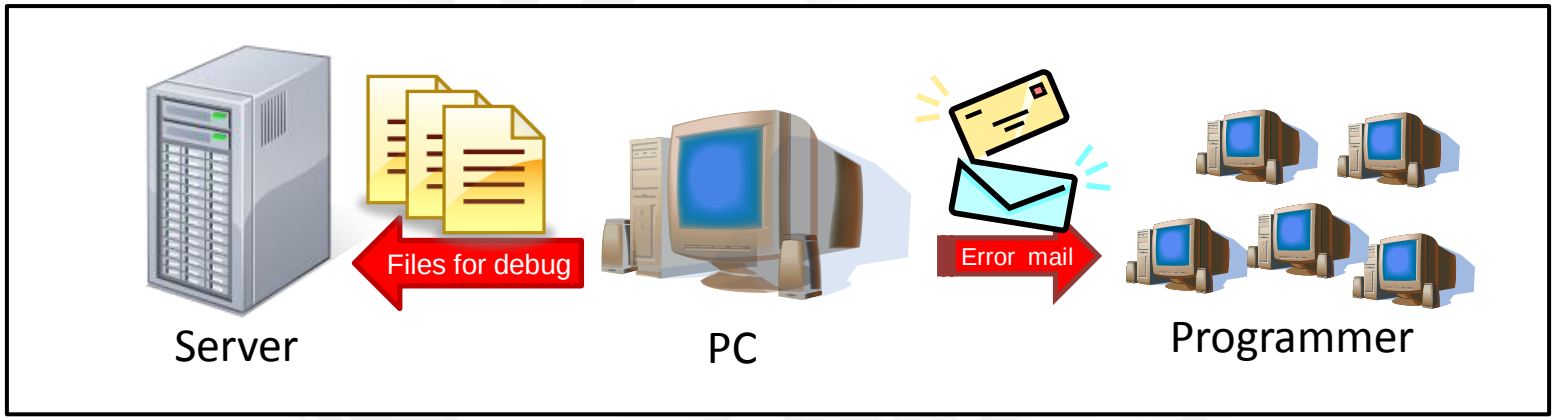
- 何によって警告が出てしまうか学習できる。
  - 危険なプログラミングを避けるようになる。

プロジェクト運用を支援する技術

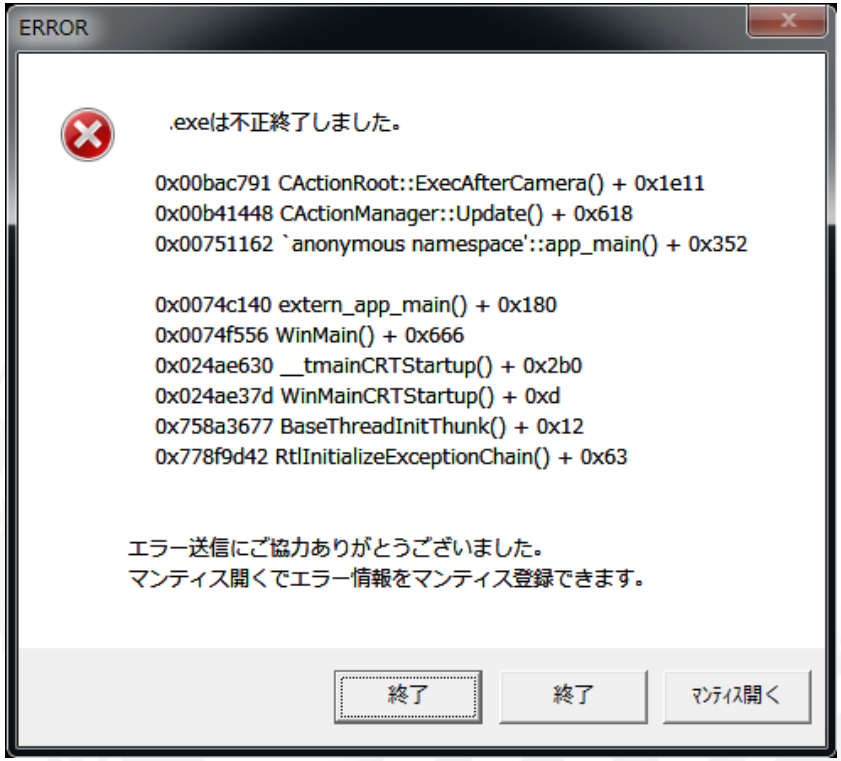
- ゲームアプリケーションのデイリーリリース（daily release）
- 自動ビルドチェック（auto build check）
- エラーレポート機能（exception report）
- AutoTest（動的テスト）

エラーレポート機能（exception report）（1/6）

- プログラム例外により異常終了した際に、
「デバッグに必要なファイル」と「メールを送信」



エラーレポート機能（exception report）（2/6）



- 例外ハンドラAPI
  - PC: **SetUnhandledExceptionFilter()**
  - PS3: **sys_dbg_register_ppu_exception_handler()**
    (cellFsOpen(“/app_home/EXEC:<path>”, …) で<path>のアプリケーションを起動可能)

エラーレポート機能（exception report）（3/6）

エラーメール例

パスは file:///(エラー送信サーバ)/2010_0113/(PC名).2010_0113_1441 です。
ログは file:///(エラー送信サーバ)/2010_0113/(PC名).2010_0113_1441/view.bat です。
coreは file:///(エラー送信サーバ)/2010_0113/(PC名).2010_0113_1441/core.bat です

```
0x01192b3d Motion::ResourceManager::RequestLoadGMT() + 0x16d
0x01197d80 Motion::ResourceManager::ChangeMotionLoadCondition() + 0x190
0x01193f37 Motion::ResourceManager::RequestLoadNext() + 0x597
0x01195320 Motion::ResourceManager::ExecAfterCamera() + 0xa0
0x0117d988 MotionManager::ExecAfterCamera() + 0x38
0x00b3e568 CActionManager::Update() + 0x618
0x0074ed42 `anonymous namespace'::app_main() + 0x352
0x00749d20 extern_app_main() + 0x180
```

送信情報へアクセスが可能
- 実行ログ
- 実行時設定ファイル
- スクリーンショット
- コアダンプファイル

コアダンプファイルを用いて即座にデバッグ可能。

エラーレポート機能（exception report）（4/6）

エラーレポートで送られるファイル

- **実行時ログ**
- 設定ファイル(.ini)
- スクリーンショット
- コアダンプファイル

実行時ログの例

```
:
[!MISSION        !] MISSION PARAM UPDATE [DISARM]
[!MISSION        !] MISSION PARAM UPDATE [MISSION ID]
[!MISSION        !] MISSION ID 0 -> 1
[!MSG            !] 0: COMMON_プレイヤー(桐生)
[!ACTION         !] ++++++++++++++++InsertAction[  221018][             CActionMission]        15 micro sec  <-- CActionRoot
[!MISSION        !] MISSION_MANAGER STATUS [eMM_STATUS_SYNC]
[!MISSION        !] SET FILE LOAD CANCELED 1 FILE mission_macro_oneshot.cpp LINE 83
[!ACTION         !] [    221018] PopInt
[!ACTION         !] ----------------DeleteAction[  221018][       CActionMissionSelect](0)         33 micro sec
[!MISSION        !] MACRO STATUS [NULL]
[!ACTION         !] ++++++++++++++++InsertAction[  221018][      CActionScenarioSelect]        16 micro sec  <-- CActionRoot
[!FADE           !] Set by mission_command.cpp 573
[!FADE           !] Set (old frame:0 alpha:1.00 prio:3000) => frame:1 alpha:0.00 prio:3000
[!MISSION        !] MACRO STATUS [NULL]
[!ACTION         !]                          ACTION_INT_CHANGE[  221018]
[!ACTION         !]                              action_int_stack[  1]  INT 2
[!ACTION         !]                              action_int_stack[  0]  INT 2
:
```

エラーレポート機能（exception report）（4/6）

エラーレポートで送られるファイル

- 実行時ログ
- **設定ファイル(.ini)**
- スクリーンショット
- コアダンプファイル

設定ファイル(.ini)の例

```
[SystemSection]
StartMission=-1          // 開始ミッション
ScenarioChapter=なし     // 開始チャプター

[FileSection]
FileManagementTime=0     // 1フレームあたりのファイル読み込み処理の最大時間（ミリ秒。0なら自動）
FileManagementLevel=0    // 1フレームあたりのファイル読み込み処理の回数（0なら自動）
FileInfoDispLevel=0      // ファイルアクセスログを出力（0～2）
FileReadTimeEmulate=0    // ファイル読込み速度をブルーレイ程度にエミュレートする（1ならON）
:
```

約1000種のデバッグ機能を管理

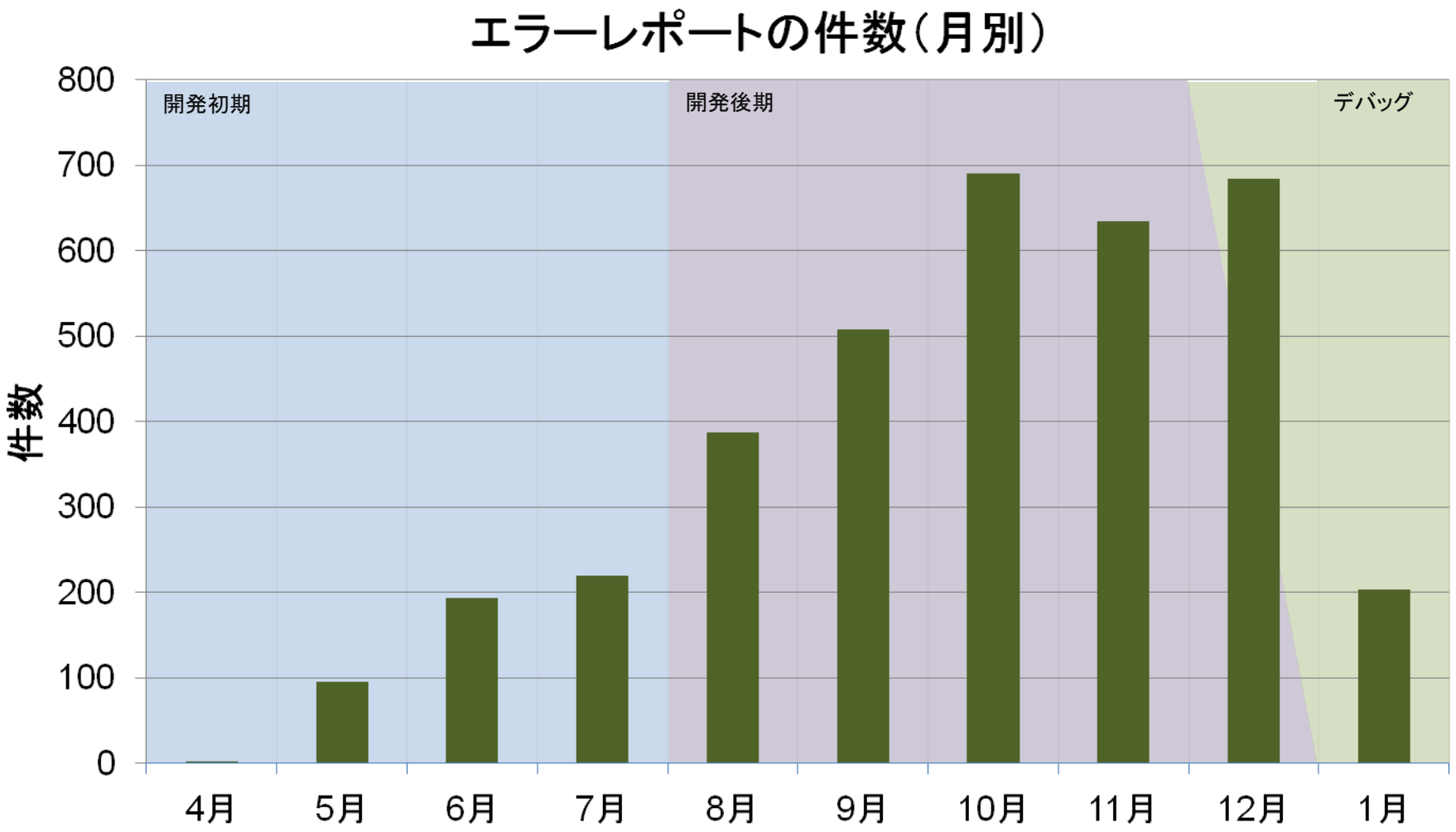
エラーレポート機能（exception report）（4/6）

エラーレポートで送られるファイル

•実行時ログ	•スクリーンショット
•設定ファイル(.ini)	**•コアダンプファイル**

- コアダンプAPI
  - PC： **MiniDumpWriteDump()**
  - PS3: **sys_dbg_signal_to_coredump_handler()**

エラーレポート機能（exception report）（5/6）



エラーレポート機能（exception report）（6/6）

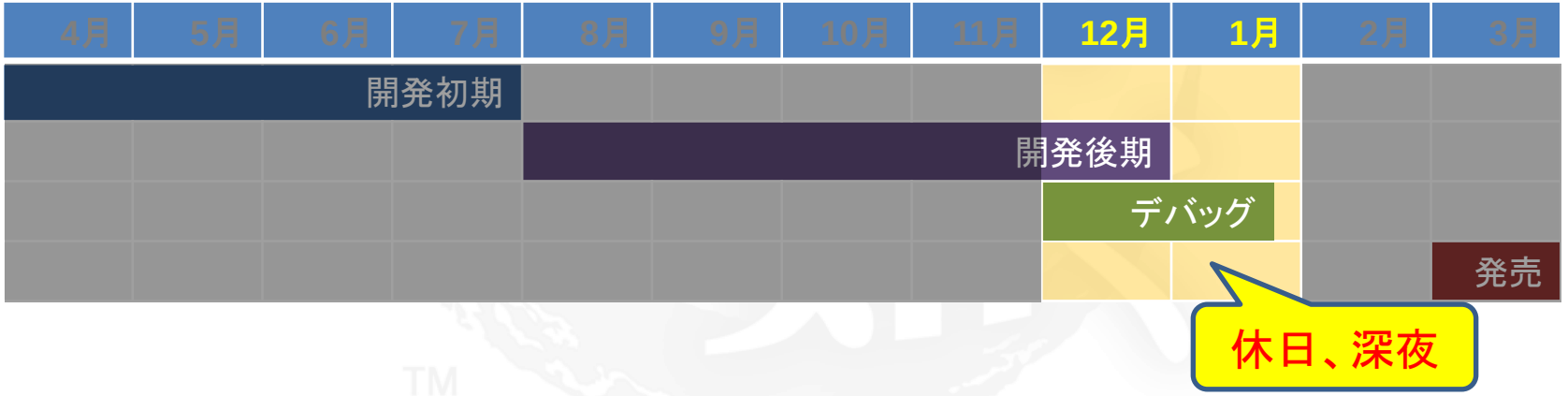
- 早期にバグを検知することができる。
  - ゲーム開発中もデバッグの無駄にはしない。
- バグ報告する人の手間を軽減できる。
  - 簡単で面倒なことはツールにやってもらう。

プロジェクト運用を支援する技術

- ゲームアプリケーションのデイリーリリース（daily release）
- 自動ビルドチェック（auto build check）
- エラーレポート機能（exception report）
- AutoTest（動的テスト）

AutoTest（動的テスト）（1/4）

- ゲームパッドの入力をランダムで発生させて、総当たりランダムでプログラムの異常個所を見つける。

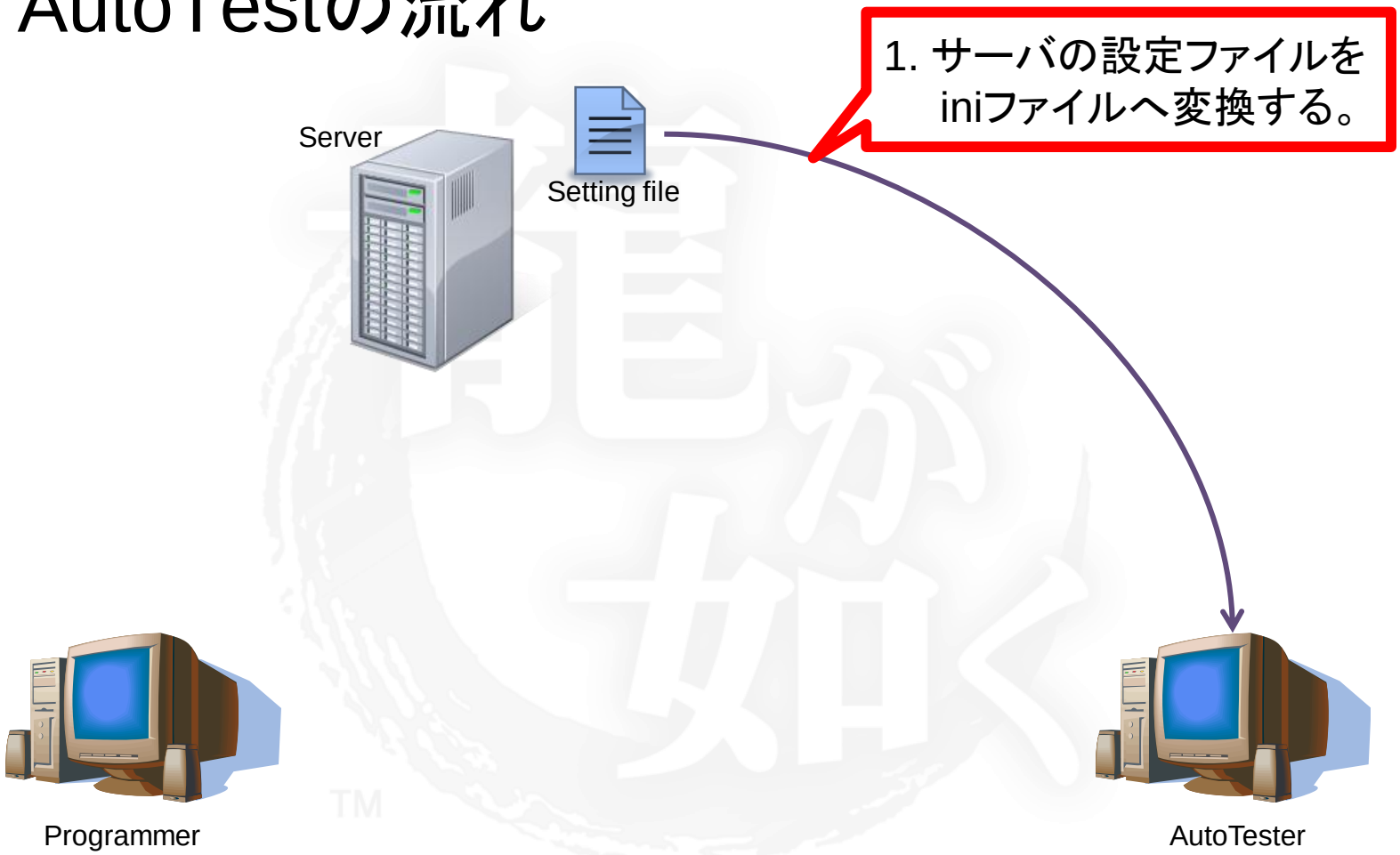


AutoTest（動的テスト）（2/4）

- AutoTestの流れ

AutoTest（動的テスト）（2/4）

- AutoTestの流れ



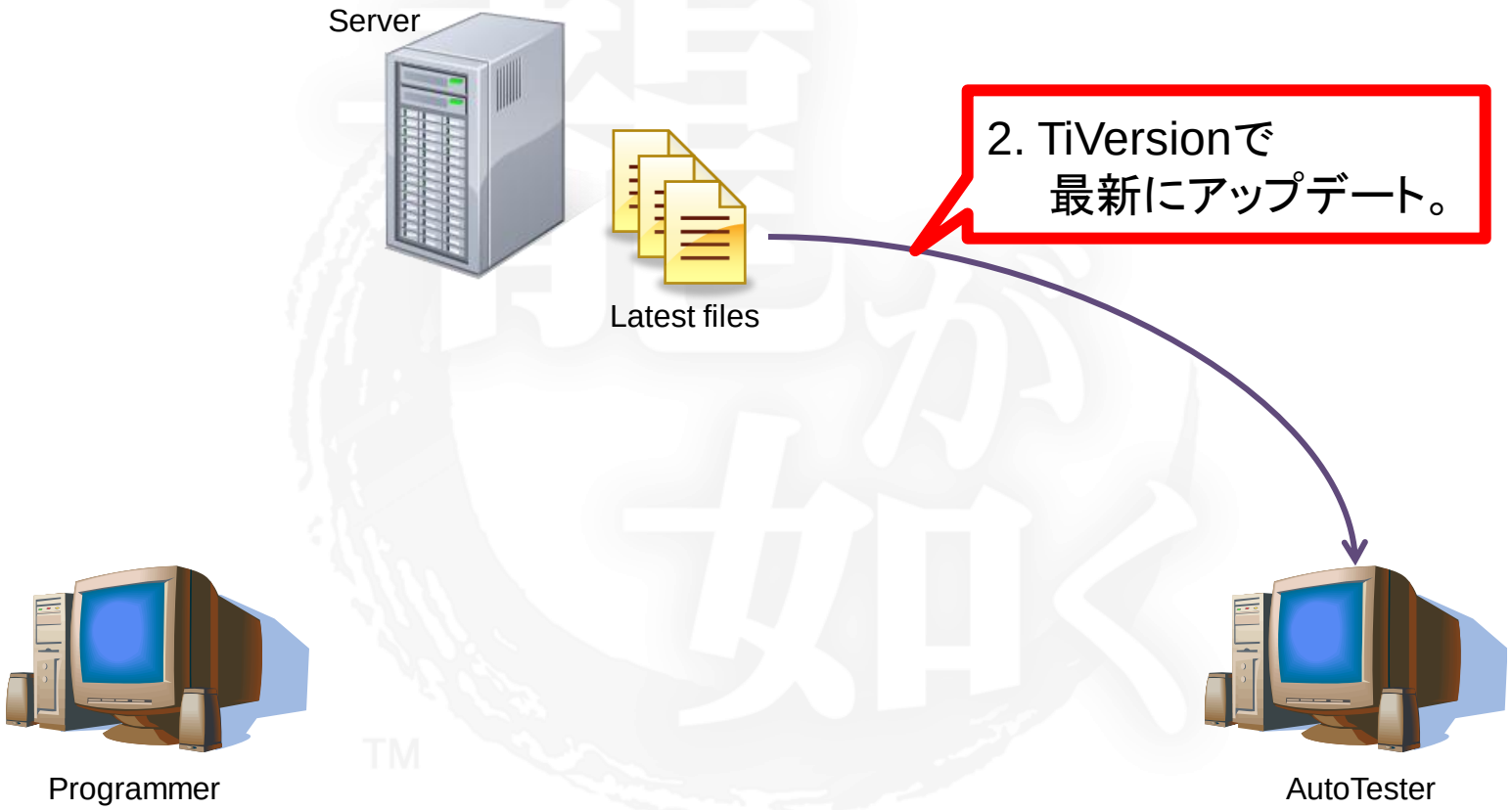
AutoTest（動的テスト）（2/4）

- AutoTest設定ファイル
  - 設定パラメータをExcelファイルに記載する。

種類	チャプター	PlayerPos	DIP	DIP	DIP	DIP	SECRET	SECRET	SECRET	SECRET	PROFILE	PROFILE
カテゴリ			SYSTEM	SYSTEM	SYSTEM	SYSTEM	主人公無敵	敵無敵	敵即死	ステージ移動なし	DebugSection	SystemSection
名前	チャプター	PlayerPos	非アクティブ時も実行	パッド無振動	MSAA Disable	コリジョン表示	PLAYER_INVINCIBLE	ENEMY_INVINCIBLE	ENEMY_SUDDENDEATH	NO_STAGEWARP	AddMoney	PlayerType
name0	PADV	(TEST)プレミアムアドベンチャーB	○	○	○	○		○		○		冴島
[プログラマ]												
name1	PADV	(TEST)プレミアムアドベンチャーA	○	○	○	○				○		谷村
name2	PADV	(TEST)ミニゲームカラオケ	○	○	○	○					10000000	
name3	PADV	(TEST)ミニゲームカラオケ	○	○	○	○					10000000	
name4	PADV	(TEST)プレミアムアドベンチャーD	○	○	○	○				○		桐生
name5	PADV	(TEST)プレミアムアドベンチャーF	○	○	○	○				○		冴島
name6	PADV	(TEST)プレミアムアドベンチャーA	○	○	○	○				○		秋山
[サウンド]												
name7	PADV	(TEST)ミニゲーム釣り	○	○	○	○		○		○	10000000	
[キャラ]												
name8	PADV	(TEST)プレミアムアドベンチャーA	○	○	○	○		○		○		秋山
name9	PADV	(TEST)プレミアムアドベンチャーB	○	○	○	○		○		○	○	桐生

AutoTest（動的テスト）（2/4）

- AutoTestの流れ

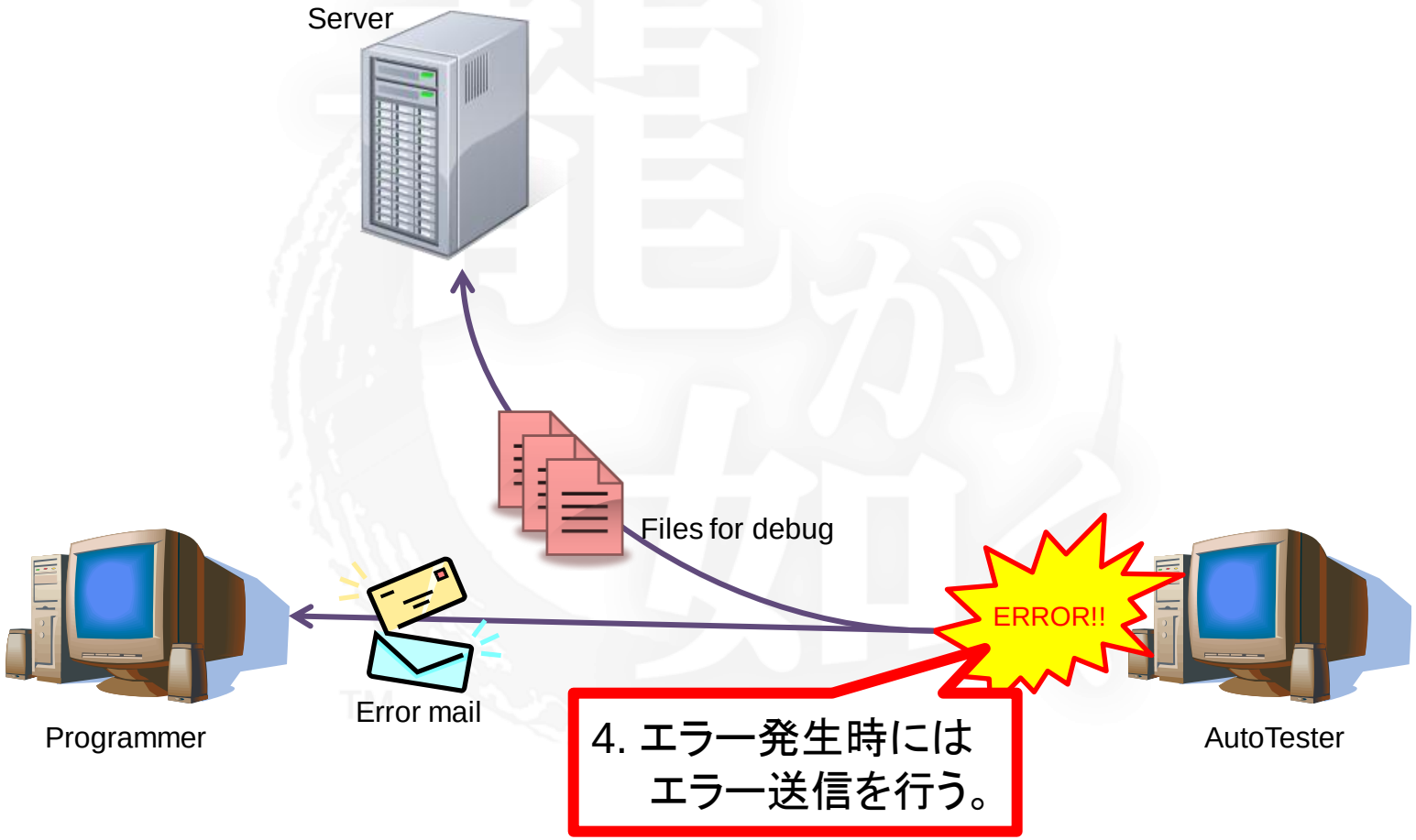


AutoTest（動的テスト）（2/4）

- AutoTestの流れ

AutoTest（動的テスト）（2/4）

- AutoTestの流れ

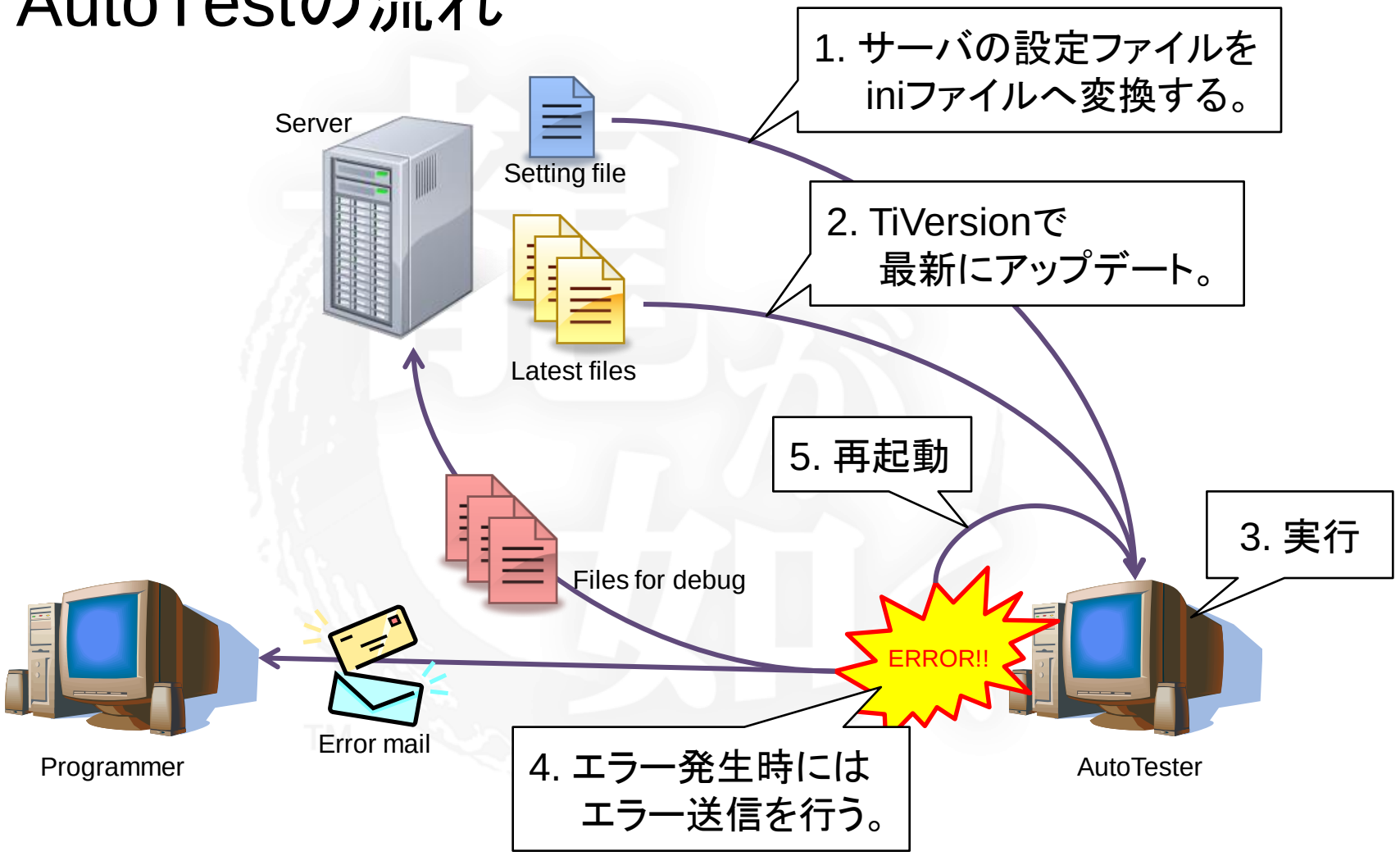


AutoTest（動的テスト）（2/4）

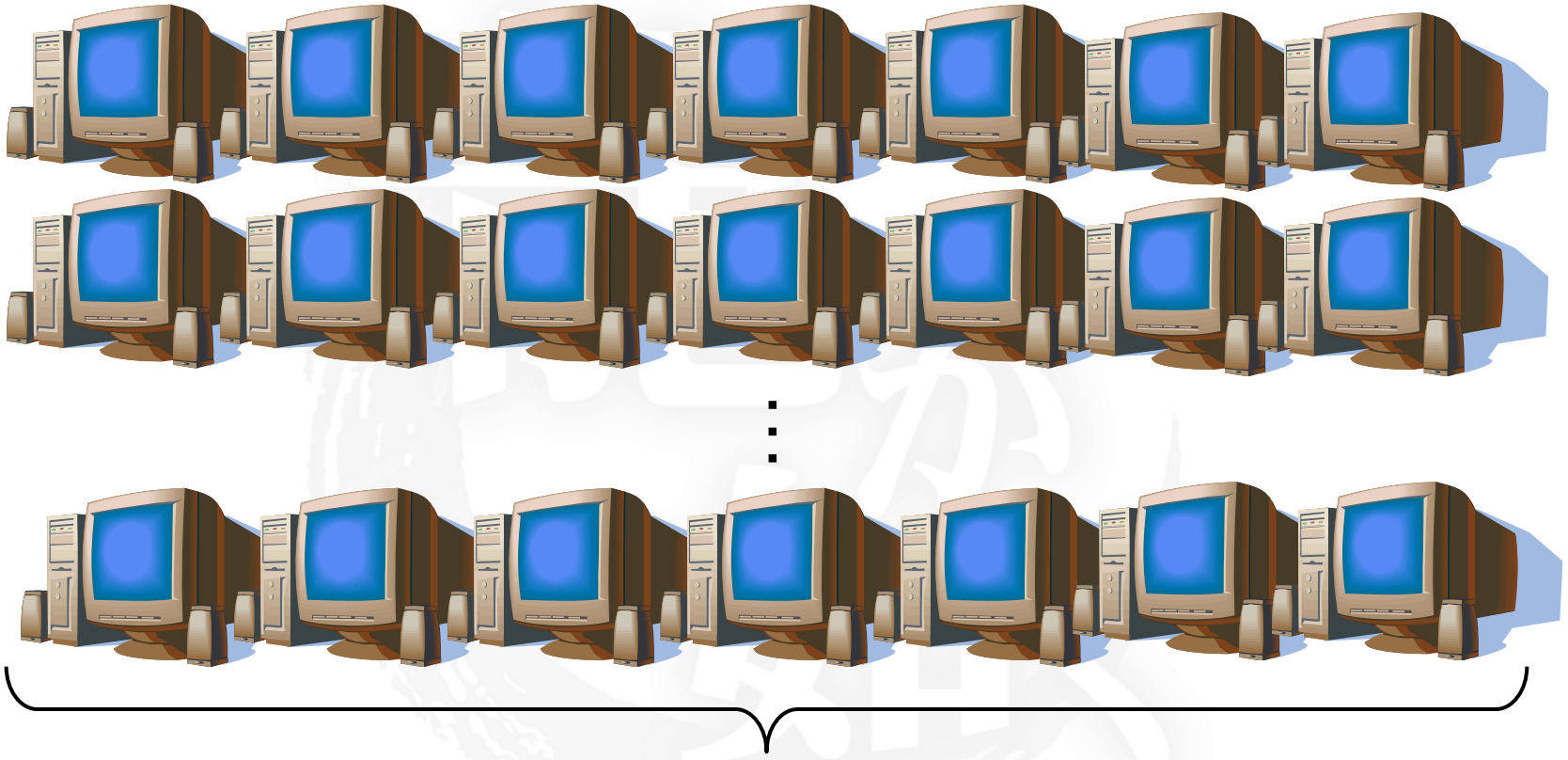
- AutoTestの流れ

AutoTest（動的テスト）（2/4）

- AutoTestの流れ



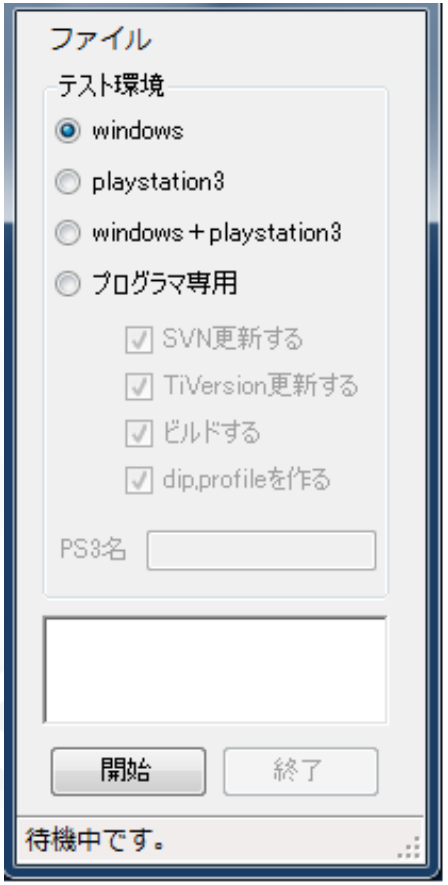
AutoTest（動的テスト）（3/4）



約80台の開発用PCで実行

AutoTest（動的テスト）（4/4）

- AutoTestクライアント
  - 操作は1クリックのみ
    - テストする人は開始ボタンを押すだけでテストすることができる。
  - 設定は不要
    - 設定はサーバで一括管理されている。
  - 主な機能
    - サーバにある設定ファイルからテスト環境の設定ファイルを準備
    - TiVersionで最新版実行環境へのアップデート
    - テスト実行中の動画録画の補助
    - プログラム例外発生時のエラー送信
    - 再起動（10回まで）



まとめ

まとめ

- 事前準備で可能な高速化は出来る限りやる
  - 予算の許す限り、高性能な機材の導入する
  - ツール・デバッグ機能の作成に手を惜しまない
- メンバーが仕事しやすい環境を整える
  - PCでゲーム機と同様の動作をする環境を構築する
  - メンバー間のコミュニケーションを密に行う
- プロジェクトを安定させる努力をする
  - プロジェクトに潜んでいる余り時間を有効活用する
  - 人手のかかる容易な作業はツールで代替してミスを防ぐ

Q&A

発表資料はCEDECのウェブサイトにて
公開予定（http://cedec.cesa.or.jp/2010/）

連絡先

Kaku_Tetsuya@sega.co.jp