CyberConnect2

開発の効率化を目指したゲームシステム

サイバーコネクトツー、15年目のポストモーテム

CEDEC2010
CESA Developers Conference

株式会社サイバーコネクトツー

開発支援室
技術開発チーフ　宇佐見公介

リード
プログラマー　相場武友

プログラマー　片桐誉裕

始めに

- introduction -

❖用語について

- なるべく北米で使われる呼び方を使用しています
  - プランナー、企画　→　ゲームデザイナー
  - デザイナー　→　アーティスト
  - モーションデザイナー　→　アニメーター
  - コンシューマー機　→　コンソール
- 会社名は敬称略を使用し、以下の略称を用います
  - 株式会社バンダイナムコゲームス様　→　NBGI
  - ソニーコンピューターエンタテインメント様　→　SCEI

❖この講義の目的

- **技術的な内容ではなく、違う視点や考え方を提供**
  - １時間で技術的な説明をしても多くを伝えることはできません。また、多くの場合、必要な時に技術書や解説書を読むほうが効率的です。
  - なので、この講義ではこれまでの私が得た経験や情報、考えている事をお話します。
- **みんなでブレインストーミングするきっかけになれば**
  - 講義を聞いて、心に感じた事を周りに伝えてください。
    - twitterやblogを使い、積極的に情報共有しましょう
  - そして、この講演をを聞いて一言ある人がいましたら、CEDECで講演して欲しいと思います。
    - そんな分析は間違っている!!、弊社ではもっと効率の良い方法を行っている!!等

第一部

- Chapter 1 -

15年目のポストモーテム

（事　後　検　証）

これまでの道のり

❖開発基盤の３つの節目

- 第一世代：PS時代(1996~1999)
  - 基盤となるシステムが無かった時代
- 第二世代：PS2時代(2000~2005)
  - 1999年11月頃から設計開始
  - PS2に実装。後にWindows Embedded、PSPに移植
- 第三世代：PS3時代(2006~)
  - 2006年04月頃から設計開始
  - 2006年07月頃からNBGIのNUライブラリをベースとして再設計
  - PS3に実装。後にXbox 360に移植

❖第一世代のゲーム制作

- 提供されているツールを使用
  - プログラマーが全てをコントロールする時代
    - ゲームの制御全般
    - エフェクトの表現、表示
    - パーティクルの表現、表示
    - カットシーンの演出、表示
    - ユーザーインターフェイスの演出、表示
  - アーティストは素材作り
    - モデル、テクスチャ
    - キャラクターアニメーション

PlayStation®

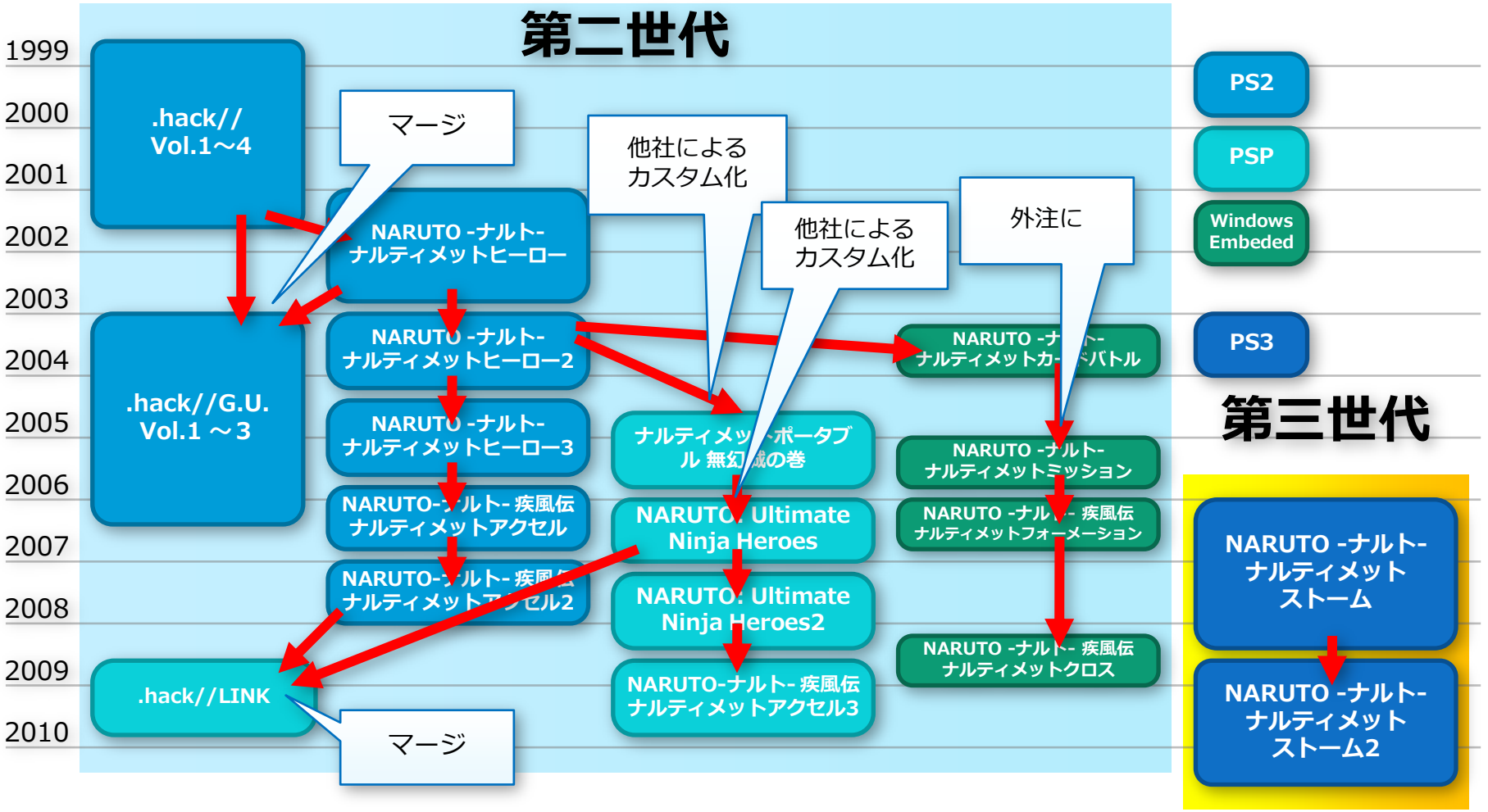
❖第二世代のゲーム制作

■ CCSと呼ばれる開発基盤の作成

- ◆ プログラマはデータの制御が主体に
  - アニメーションの切り替え制御
  - ゲームフロー制御
- ◆ アーティストはアニメーションを含む素材を全て作成
  - エフェクトの表現
  - カットシーンの演出
  - パーティクルの表現、表示
  - ユーザーインターフェイスの演出
  - モデル、テクスチャ素材
  - アニメーション全般

❖開発タイトルによる開発基盤の派生

※開発時期、期間に関して正確ではありません

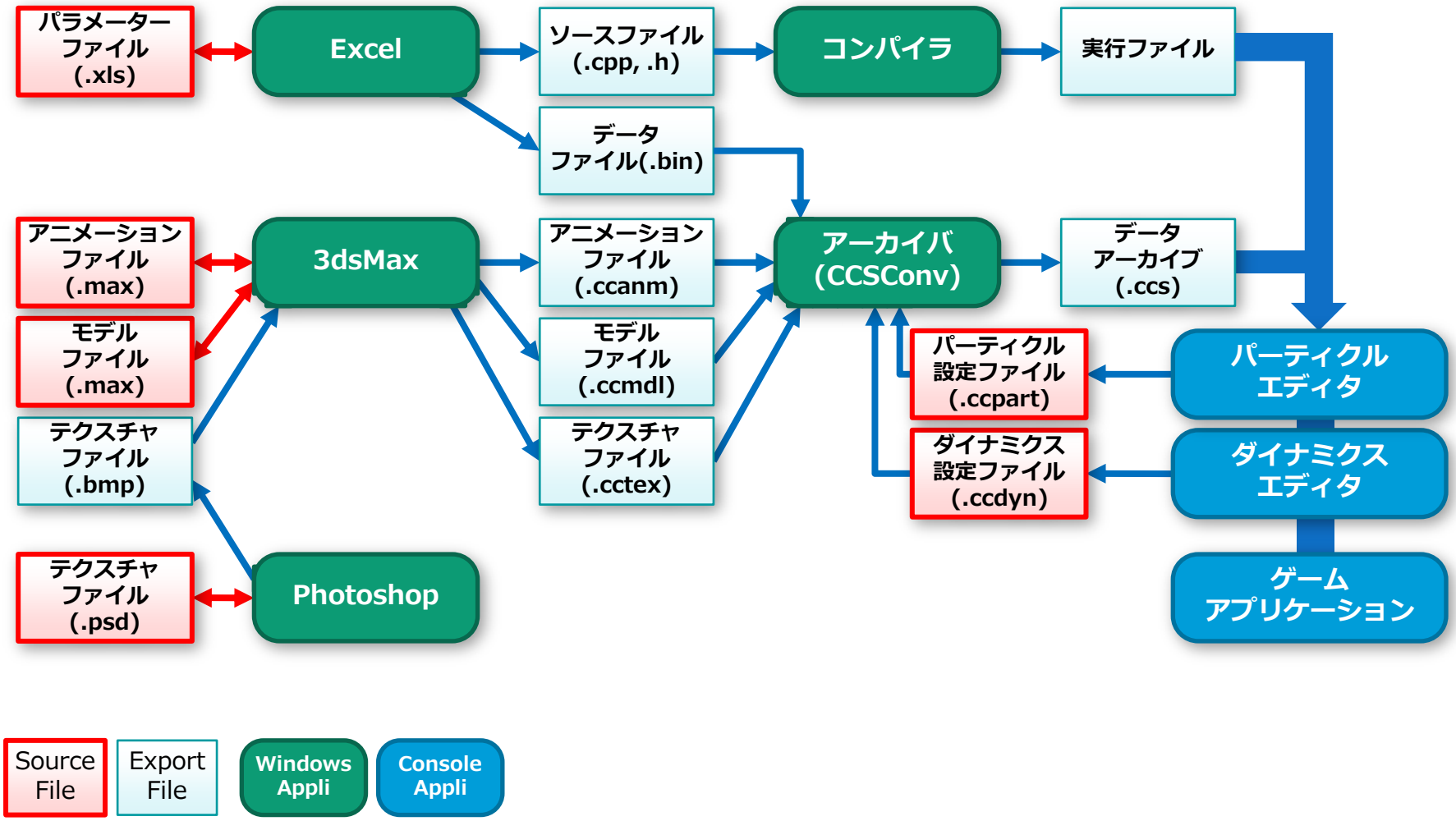


❖第三世代のゲーム制作

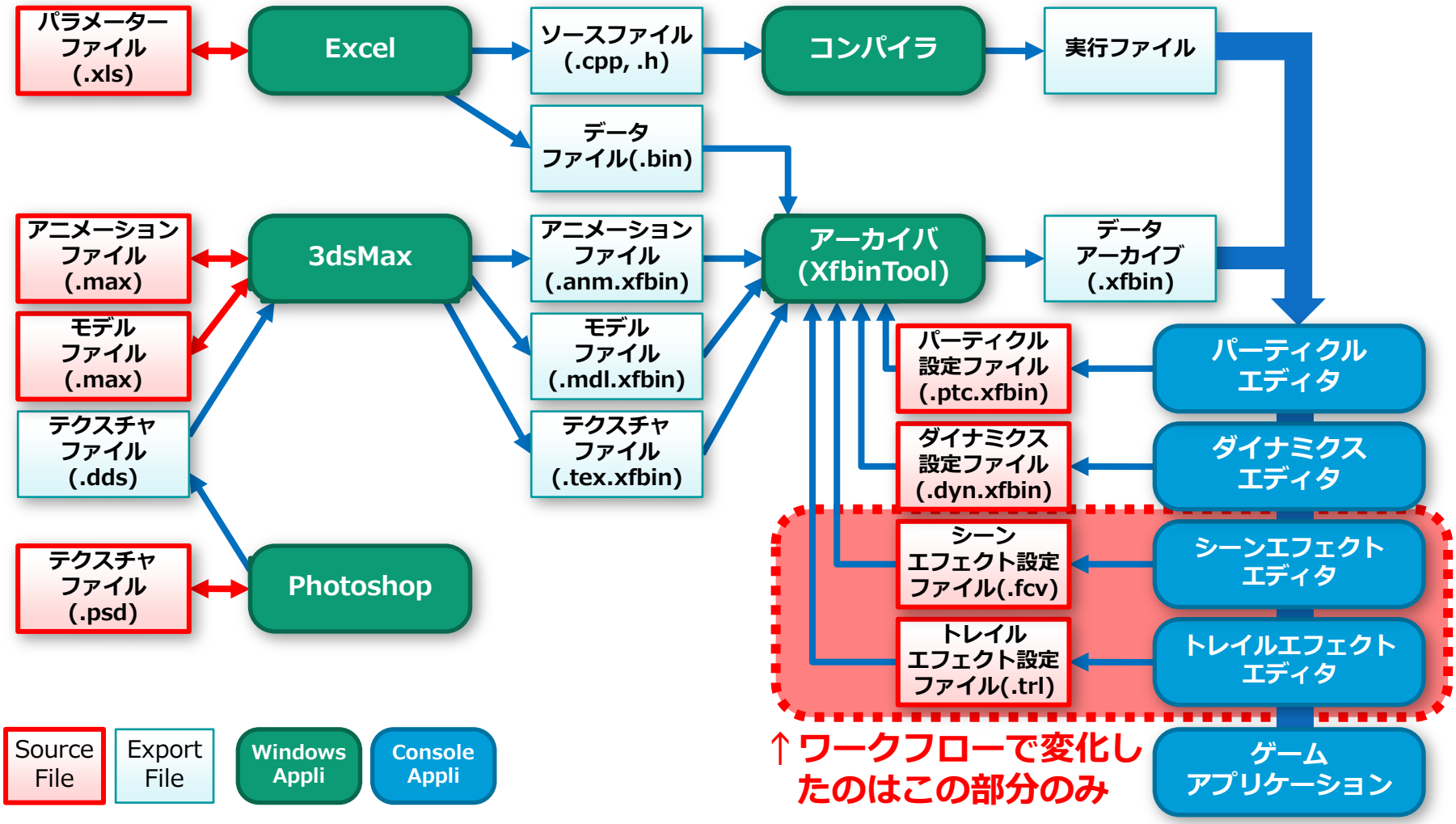
- CCSからNUCC/xfbinへ
  - NUCCライブラリ（NUライブラリを下位層にして構築）
  - xfbinフォーマット（CCSをベースにフォーマットを作り直し）
    - 精度の問題、コンバートの問題などで作り直す必要性があった
- しかし基本は変わらず
  - コンソールが切り替わったので、ツール群の作り直し
  - GUIツールキットの追加
  - スケジュールの都合上、幾つかの機能を断念
    - メタフォーマットの採用
    - レベルエディタの実装

Xbox 360®　PlayStation®3

❖第二世代のワークフロー



❖第三世代のワークフロー



❖弊社の基盤開発のまとめ

第一世代
効率化の
無かった制作

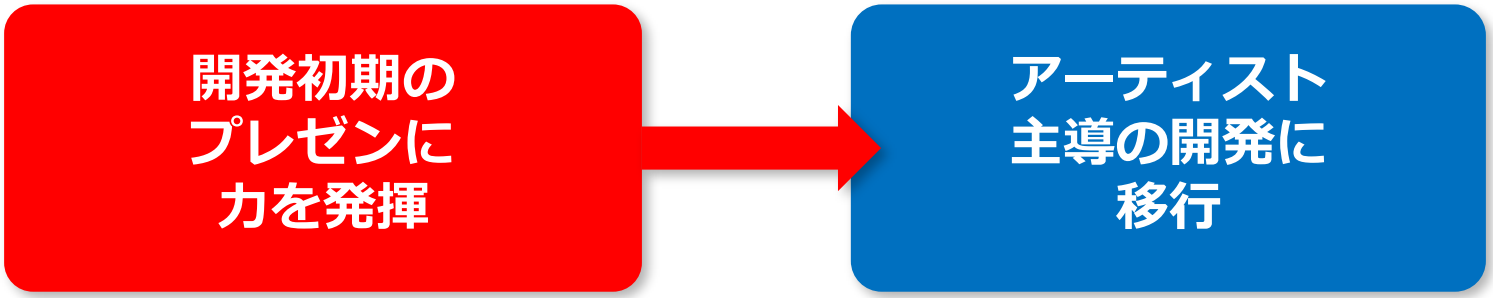
第二世代
それなりに
効率化が上手くいった

第三世代
第二世代を引き継いだ
安定期

成功した第二世代
- その機能と工夫 -

❖機能１：ストリーミング・シーン再生

- 一番最初に実装した機能
  - リアルタイムデモを長時間再生させる為の仕組み
    - CCS = Cyber Connect Streaming システム
  - 3dsMaxのシーンファイルをそのまま再生
    - シーンの表示オブジェクトをセットアップ後、
      オブジェクトに対してフレーム毎のアニメーション値を割り当て
- ゲームのイメージを実機上で再現



❖機能２：シーンベースのアニメーション表示

モデルにアニメーション
を割り当てる従来の方法

シーンベースの表示方法

全て合わせた状態で
１つのアニメーション

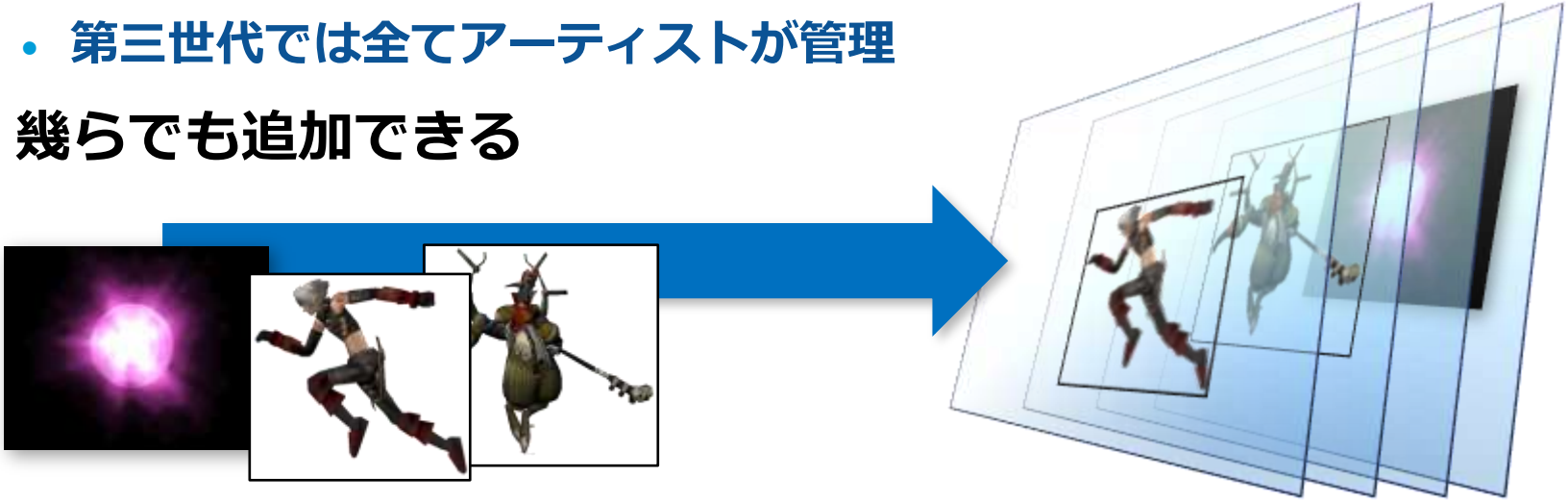
❖機能２：シーンベースのアニメーション表示

- 開発当初はプログラマーによる制御が多かったが…
  - プログラマーで制御できない、キャラクターモーションとカットシーンで使われるのを想定
  - しかし、使い勝手が良かった為、殆どの箇所で使用されるように
    - UIの表現（.hackのトップページ画面など）
    - エフェクト、パーティクル
    - その他、アニメーションを行う箇所は殆ど使用

❖機能３：レイヤー機能

■ 描画優先を管理するクラス

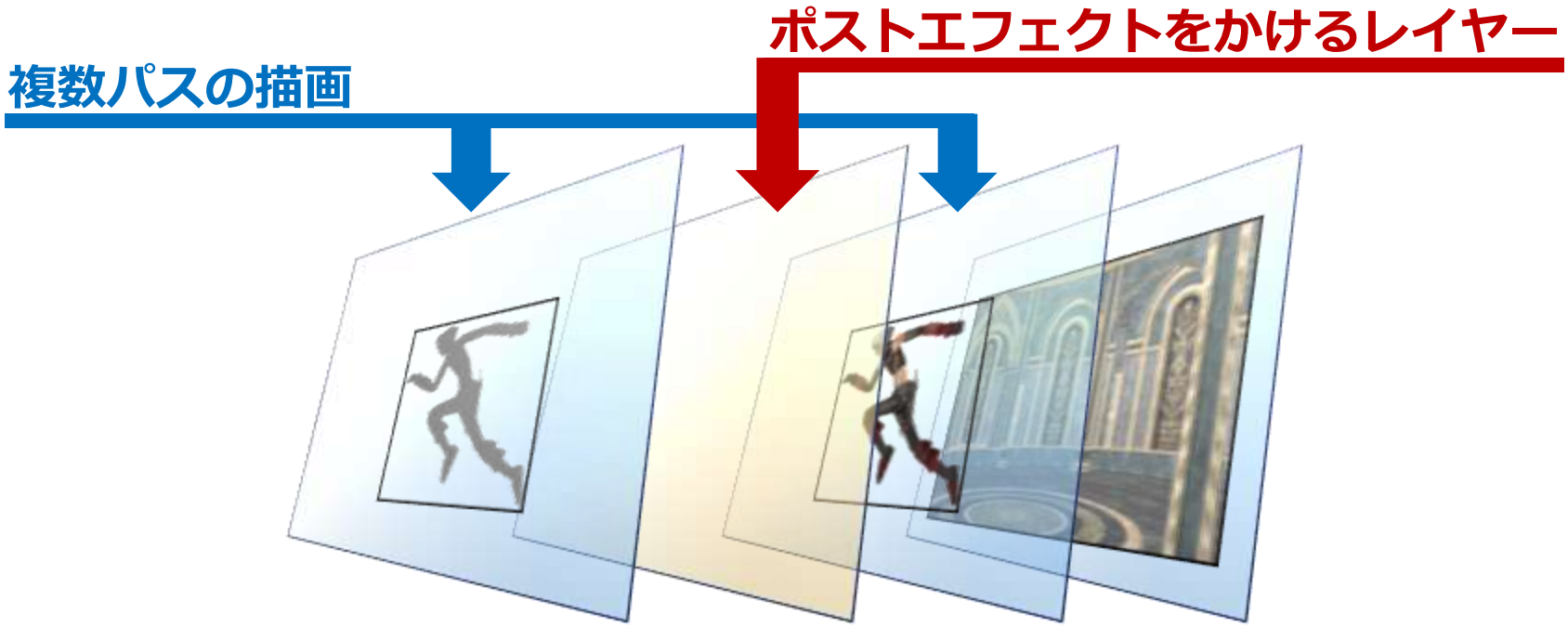
- ◆ どのレイヤーに描画するかを指定して外部から優先を制御
  - シーン再生の為に、全ての表示オブジェクトを描画優先を指定する仕組みが必要だった。
- ◆ どのレイヤーに描画するかをアーティストが指定
  - 第二世代ではシーン再生のみアーティストが管理
  - 第三世代では全てアーティストが管理
- ◆ 幾らでも追加できる



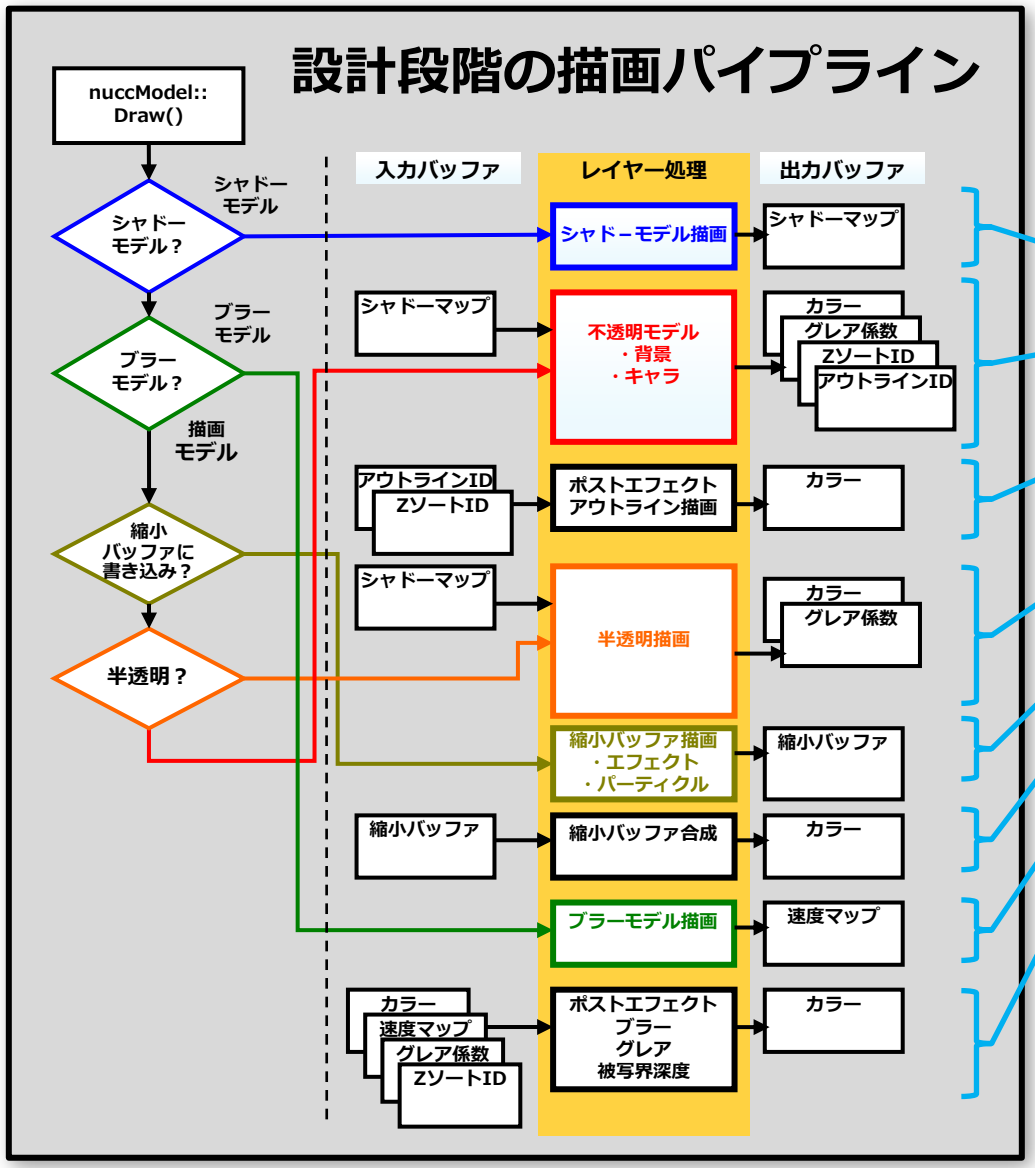
❖機能３：レイヤー機能

■ シェーダーの描画パスの管理にも利用

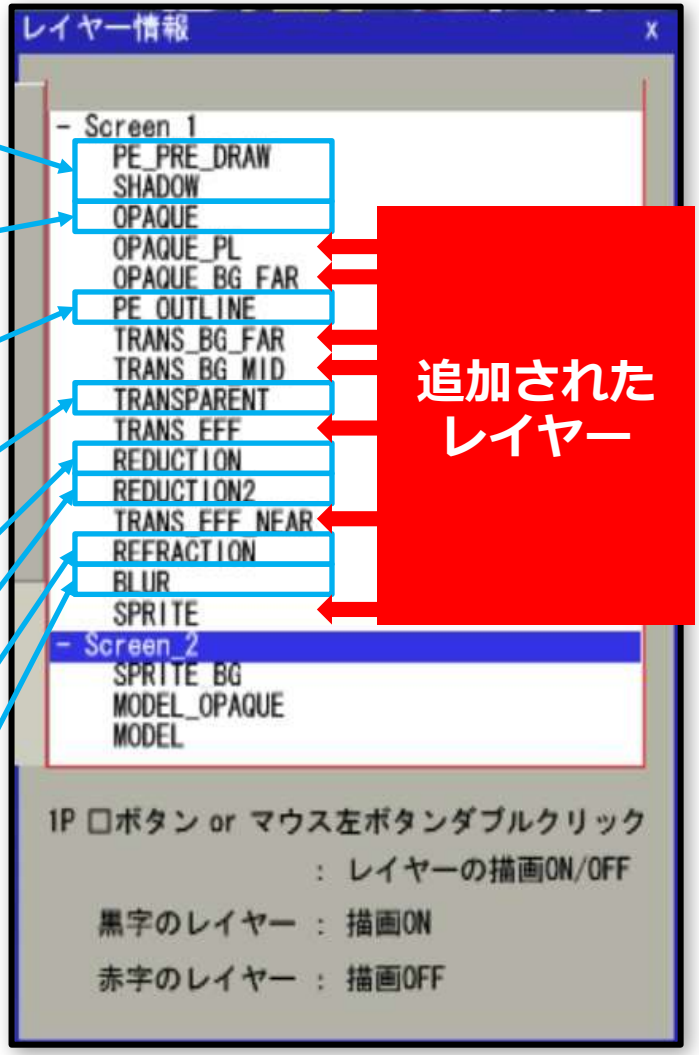
- ◆ 複数パスのシェーダーの順序を管理
- ◆ ポストエフェクトを何処までかけるのかをレイヤーで管理



設計段階の描画パイプライン



実際のレイヤー構成



データ管理の工夫

- １ソースファイル、１出力ファイル
  - DCCツール上で扱う素材のファイル（テクスチャ、モデル、アニメーション）１つに付き、１ファイル出力される
  - アーティストが理解しやすく、ミスしにくい
- データ内にデータソースのファイルパス情報を埋め込み
  - エラー時のデータ特定がしやすい
- メモリ確保サイズが先にわかるようにフォーマットを作成
  - ストリーミング読み込みとあわせて、無駄なメモリを消費しない工夫

安定期の第三世代
- 引き継がれた問題点 -

❖データフォーマットの問題

- プロジェクトで様々な拡張データが必要
  - データフォーマットに柔軟性がなく、追加に多くの手間
    - フォーマット作成
    - 編集ツールの作成（入力UI、データ保存、出力処理等）
    - コンソール上のパーサー、データ処理
  - プロジェクトで様々な拡張データが必要
    - 担当者ごとに独自フォーマット
    - ツールの扱いに統一性がない状態に

❖シェーダー対応の問題

第三世代からシェーダー対応

- シェーダーによって、入力されるデータは様々
  - シェーダーの汎用性のため、データフォーマットも汎用性が必要に!!
  - 本来なら、メタフォーマットを採用するべき
- しかし、時間的な問題から汎用的なデータフォーマットの実装を断念
  - その為、シェーダーを新たに作成するごとに、ゲームアプリケーションとコンバーターに対して固有のコードを追加することに
  - 新たなシェーダーを実装する手間がふえ、人的アサインが難しくなった

❖拡張性のないシステムの問題

- ツールの実装は担当者任せ
  - ツールごとに操作や完成度もバラバラ
    - アンドゥができない
    - よく落ちる
    - エクスポートはできるが、インポートができない
  - メンテナンスが後回し
    - プロジェクトのスケジュールが最優先
    - 担当者が変わったりして、段々と直しにくい状態に
  - 結果、ツールの完成度を上げることが出来ない
    - 継続して使われない理由
    - 担当者が入れ替わると作り直しになりやすい

❖GUI Toolkit の実装の問題

■ ゲームアプリケーション側に実装

- ◆ Visual Studioのリソースエディタを使用
  - ある程度は効率化
- ◆ しかし問題も多い
  - 使用時に開発機が必要
  - パッド操作による扱いにくい操作
  - コンソールに依存しているので、他のコンソールへの移行が難しい

VSリソースエディタ上　　コンソール上

❖問題点のまとめ

- データフォーマットの問題
- シェーダー対応の問題
- 拡張性のないシステムの問題
- GUIツールキットの実装

拡張性
ツールの制作
が重要に

問題の要因をさぐる

❖ゲームプロジェクト依存の問題

- スケジュールの問題から設計が十分に行われない
- プロジェクトの初期段階で基盤構築
  - スケジュールによって最低限必要な分だけ実装
  - ツールは、その担当者がスケジュールに…以下同文
    - PS2は開発当初の人数が多かったがPS3では少なく、
      結果として断念した箇所が多かった
- プロジェクトの後半は機能追加とメンテナンス
  - 機能充実よりも、ゲーム開発優先
    - 必要最低限の機能のみ、使い勝手が置き去りに
    - マニュアルの不足

❖ゲームプロジェクト依存の問題

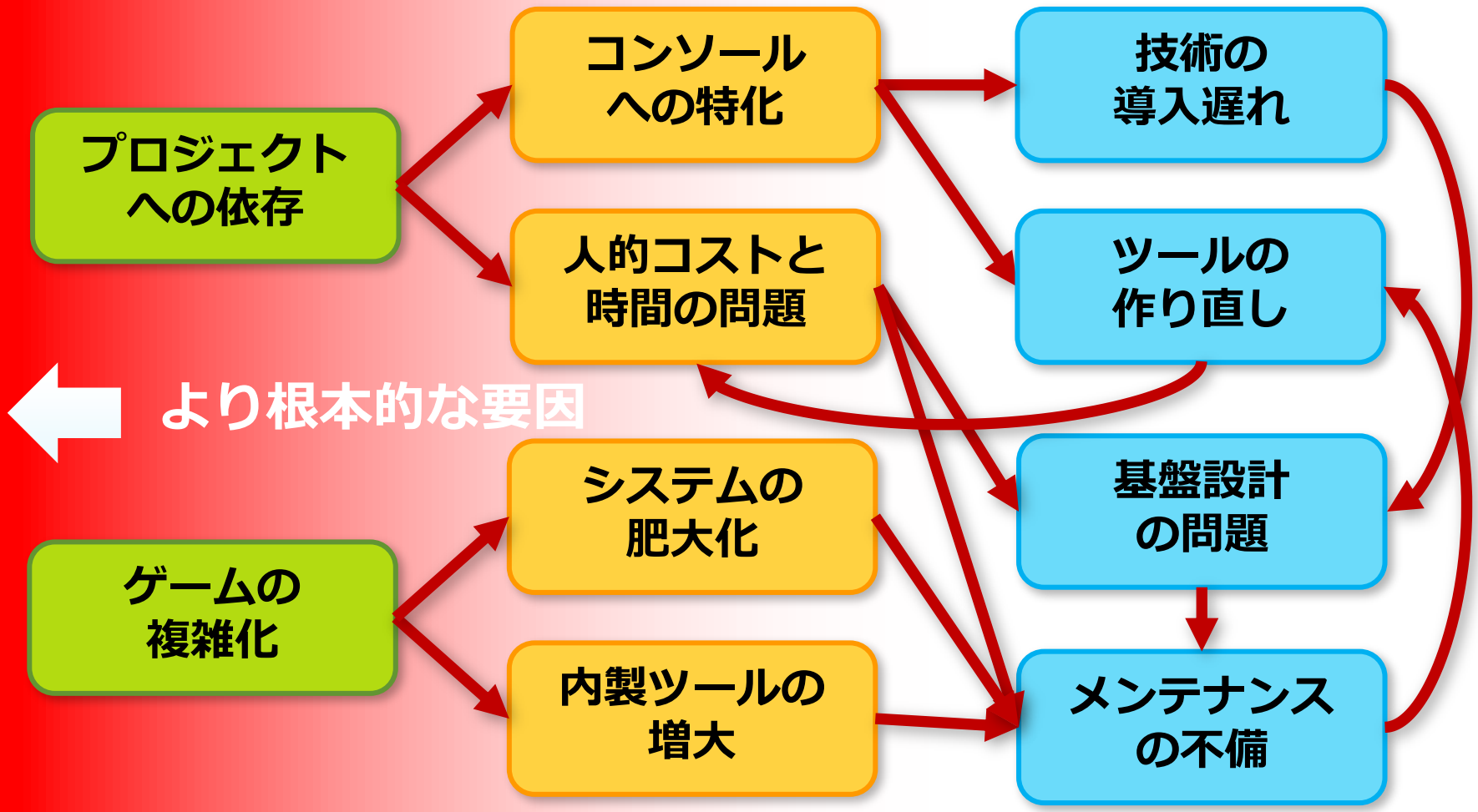
■ コンソールに依存の問題

- ◆ 新しい技術の導入が遅れる
  - PS2 → PS3への移り変わりで問題に
  - 世代の移り変わりに対応しにくくなる
- ◆ 設計がメインコンソールに依存
  - 可搬性が低く、世代の切り替えで多くの作り直し作業が発生

❖ゲームの複雑化による問題

- ツールの増大
  - 使いやすさの犠牲
  - 機能が不十分
- システムの肥大化
  - 各プロジェクトの必要機能を追加しながら肥大化
  - 肥大化しメンテナンスが行きとどかなくなる
    - 去年の弊社CEDEC講演でも、今後の課題として上げていた
  - 処理速度にも問題が出てくる

❖問題点のまとめ


プロジェクト
への依存
コンソール
への特化
人的コストと
時間の問題
技術の
導入遅れ
ツールの
作り直し
より根本的な要因
ゲームの
複雑化
システムの
肥大化
内製ツールの
増大
基盤設計
の問題
メンテナンス
の不備

第二部

- Chapter 2 -

新しい開発環境に向けて

❖問題点に対処する為に

プロジェクト
への依存

→ プロジェクトから
独立した部門の設立

ゲームの
複雑化

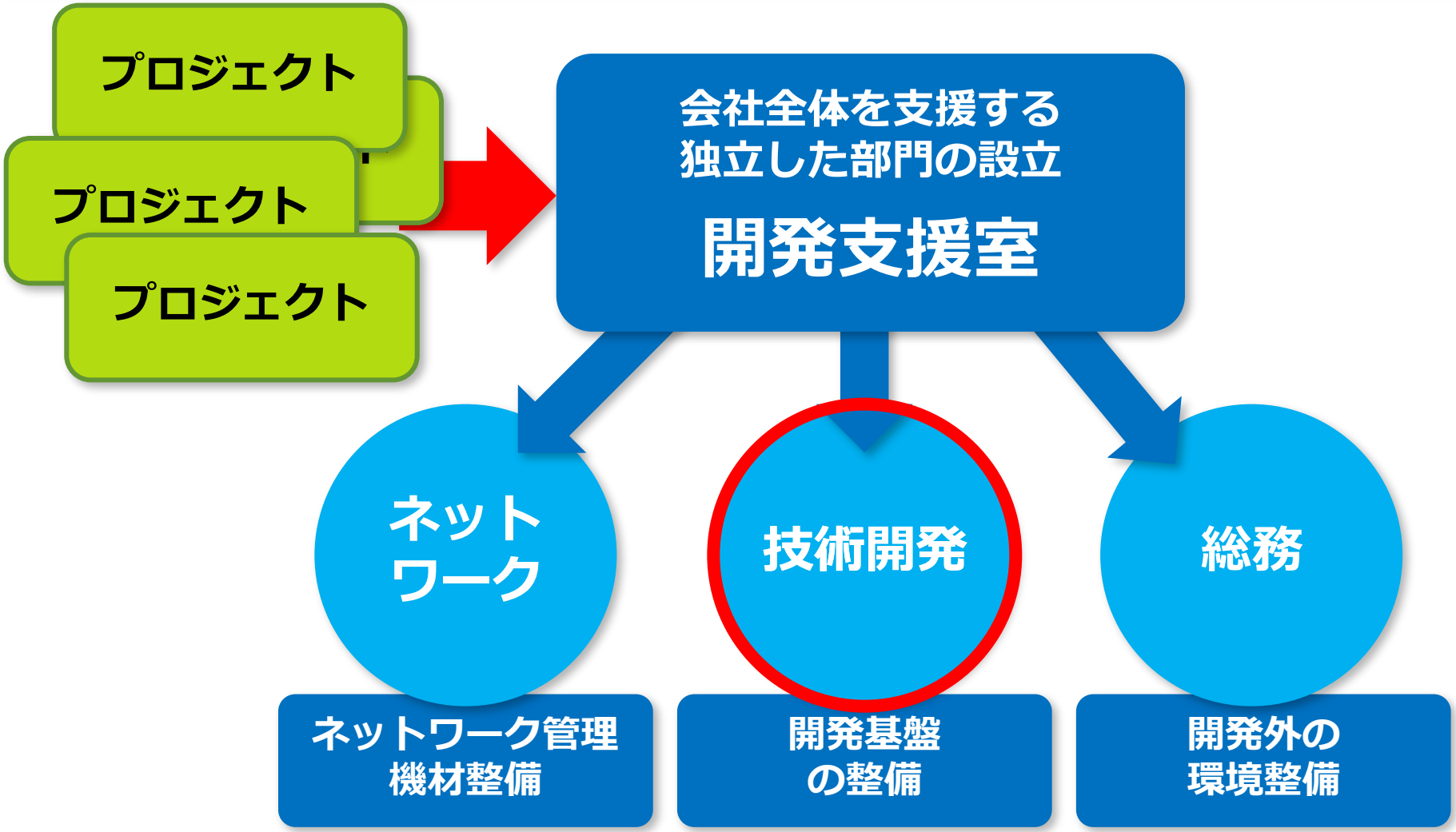
しかし、市場の要求なので
ゲームを簡略化すれば
解決する訳ではない

内製ツールの
増大

システムの
肥大化

→ 新しい開発基盤
の構築

❖プロジェクトからの独立をした部門の設立



❖真にプロジェクトからの独立を目指すために

- 独立したものの？
  - プロジェクトはどこも人手が足りない
  - なんだかんだで、プロジェクトに引っ張られる現状
- 明確な目標設定
  - 会社に何が足りなくて、何が必要なのか？
- 意思を伝える努力
  - 会社、上司への説得が必要
  - 営業活動も必要

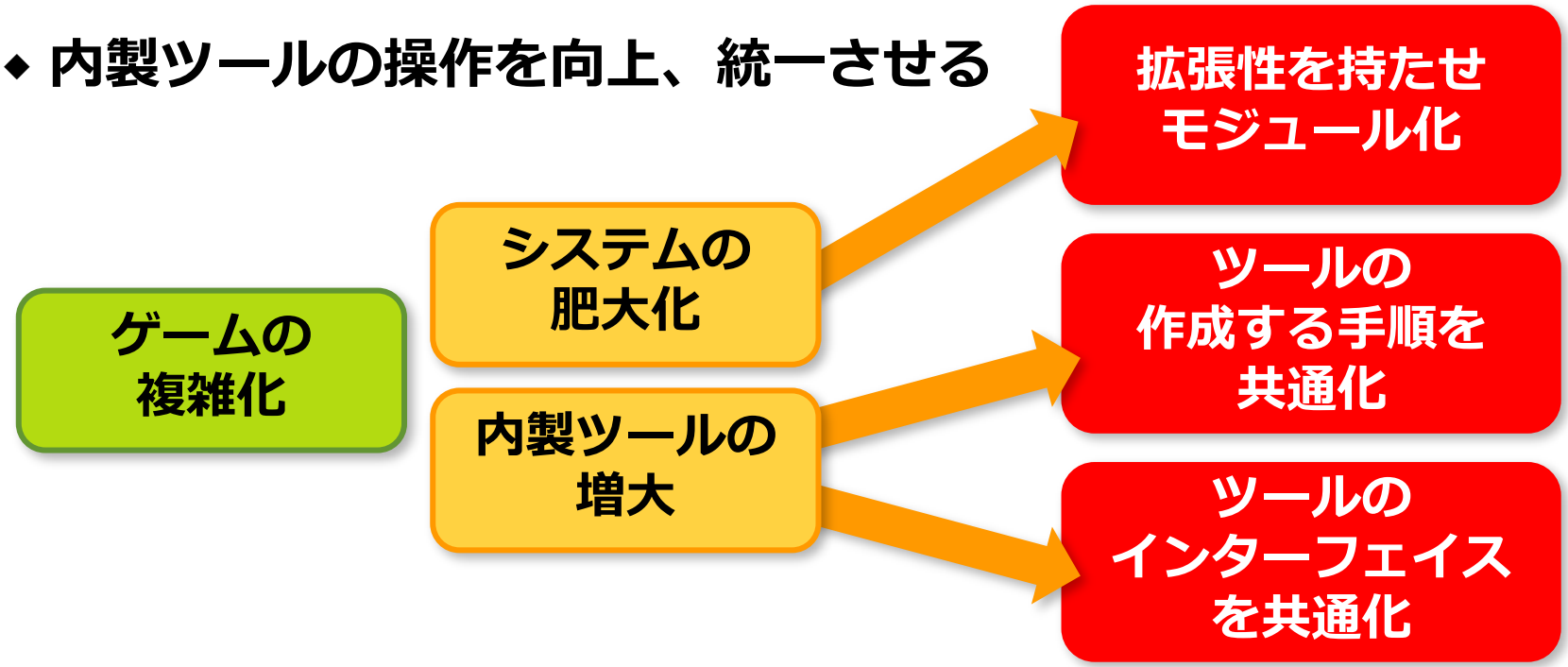
❖段階的な目標設定

- まずは小規模の目標から始める
  - 大規模ではリスクも高い。コスト意識も重要
    - ゲームエンジンを目標とすると、完成までに時間が掛かり過ぎる
    - 小さな実装から導入を出来る仕組み
- 段階的な目標設定
  - 導入がうまくいけば、フレームワーク開発へ
  - フレームワークが形になればエンジン開発へ

❖第一目標

■ 内製ツールの開発環境を整える

- ゲームで使われるデータ全てを管理出来るデータフォーマット
- それを使用する内製ツールの開発環境
- 内製ツールの操作を向上、統一させる



❖内製ツールの増大を解消するために

データの扱いを
共通化

→ 共通データフォーマット
&
データ操作API

ツール制作の手順
ツールの操作手順
を共通化

→ ビジュアルプログラミングAPI
&
Windows GUI Toolkit

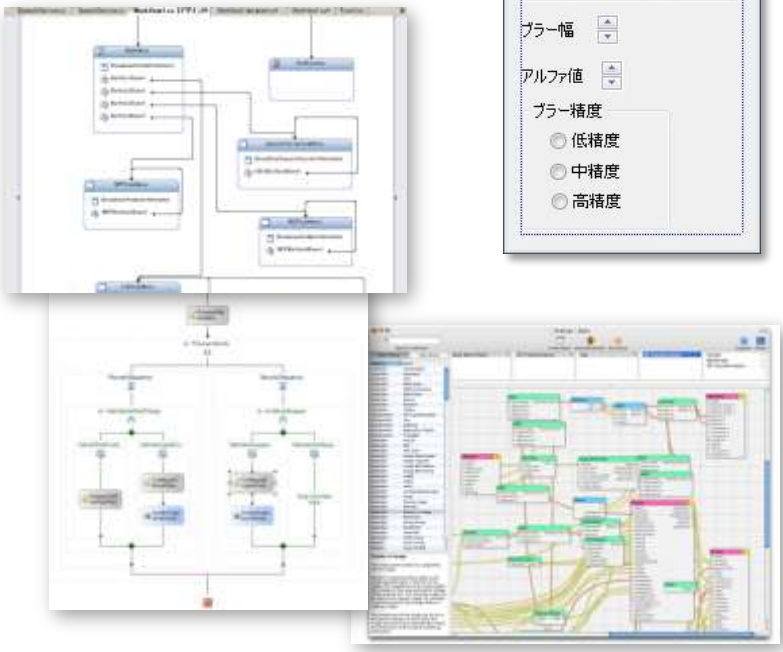
❖共通データフォーマット&データ操作APIについて

■ データ定義と生成、操作手順を共通化

- ◆ データからクラスを生成
  - あらゆるクラスをシリアライズ、デシリアライズする共通した手順
- ◆ 全てのデータを保存、編集可能に
- ◆ 全てのデータをアニメーション可能に
- ◆ エクセルシートとの相互変換機能
- ◆ ヒストリー機能
  - 全てのデータをUndo出来る様に。ノード間の接続も履歴をとる
  - ヒストリー自体も保存可能に

GUI Toolkitとビジュアルプログラミング API

- 共通データフォーマットをGUIで編集
  - 第三世代で実装したGUI Toolkitを
    コンソール側ではなくWindows側で実現
- ビジュアルプログラミングAPI
  - 制御フローの視覚化
    - フローチャート、アクティビティ図、
      状態推移図などをサポート
    - ゲームフロー制御、シナリオ制御に利用
  - データフローの視覚化
    - シェーダープログラミング
    - エフェクト、パーティクル設定
    - レンダリングパイプライン設定



❖ライブラリをそろえるだけでは使ってもらえない

データの扱いを
共通化

→ 共通データフォーマット
&
データ操作API

ツール制作の手順
ツールの操作手順
を共通化

→ ビジュアルプログラミングAPI
&
Windows GUI Toolkit

↓

ゲームデータ・エディタ

❖ゲームデータエディタの機能

- **データの構造を規定**
  - 初期値
  - 読み書き属性
  - 編集用のGUIの指定
  - データ内容の説明文
- **データエディタ**
  - データの編集
  - デバッグ情報も埋め込む
- **データ最適化**
  - デバッグ情報の削除

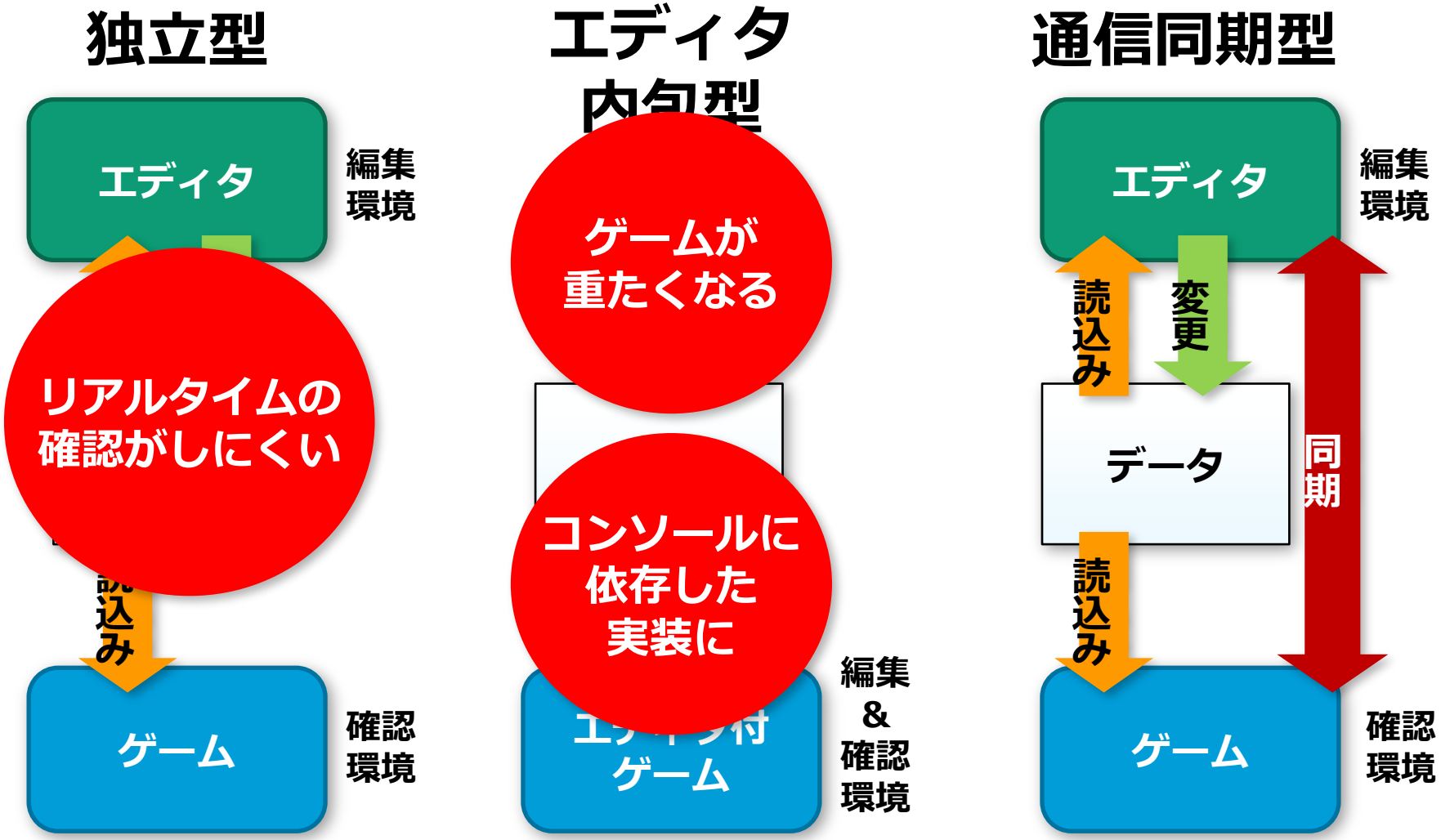
❖データ構造の記述をどうするか

- 多くはテキストベースの記述
  - C++でデータ構造を記述
    - コンバーターでXMLで出力
    - クラス以外の情報が設定しにくい
  - XML等でデータ構造を記述
    - コンバーターでC++言語ヘッダ＆ソースを出力
    - XMLの記述が面倒

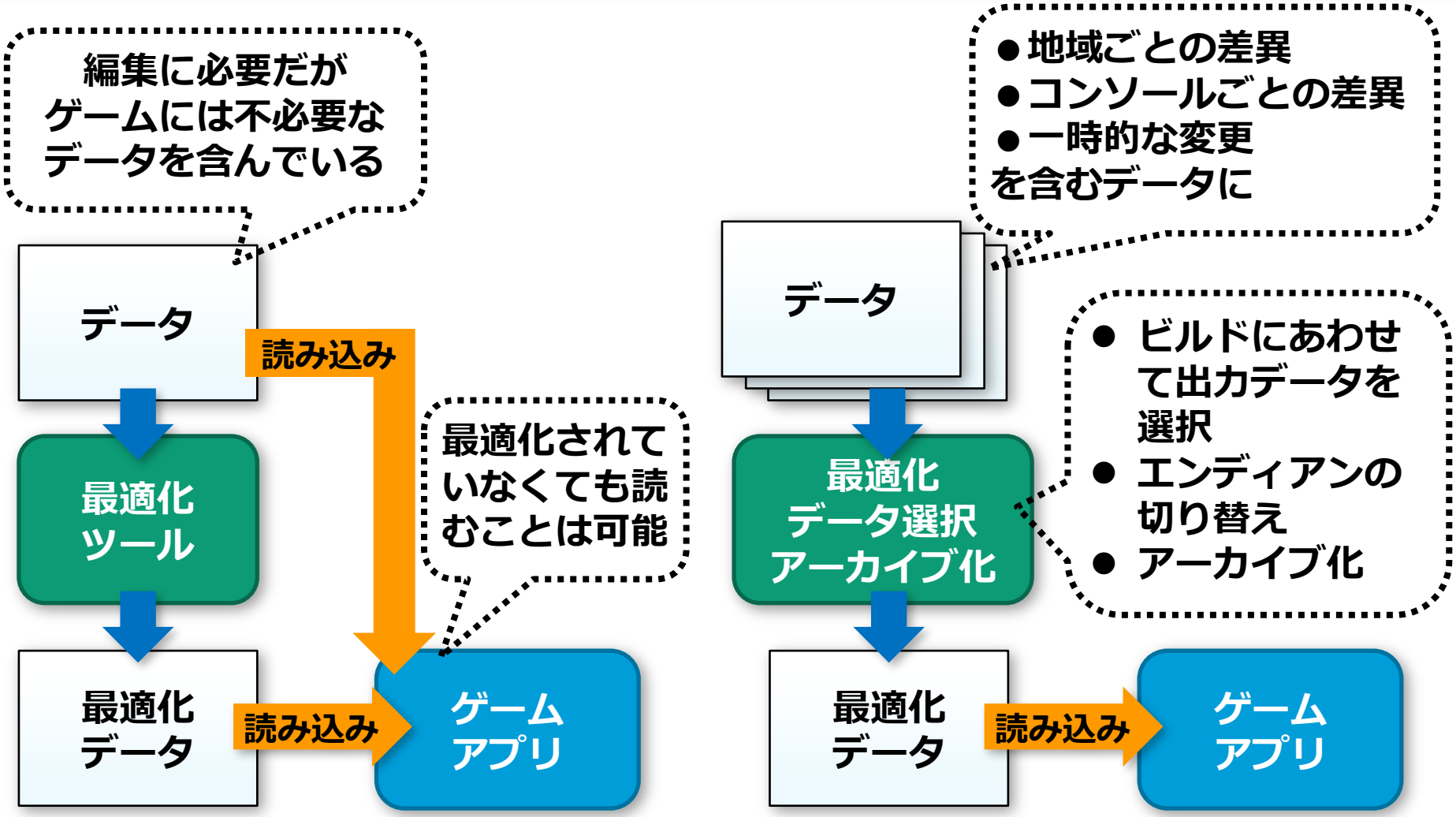
エディタによる記述

分かりやすい、無駄が少ない
XML、C++言語のヘッダー＆ソースを出力

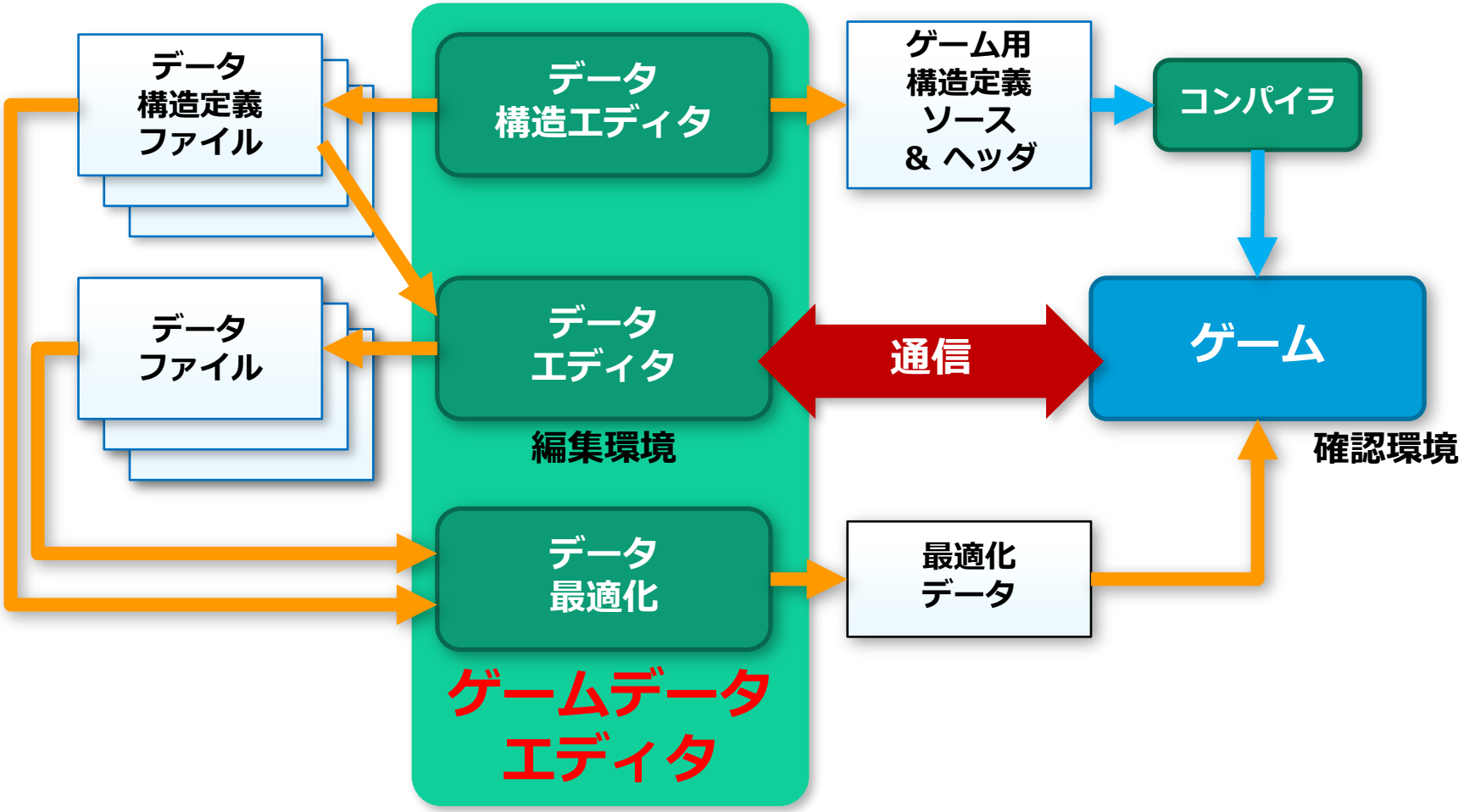
❖データを編集する仕組みをどうするか？



❖編集したデータの扱いをどうするか？



❖データエディタの構成

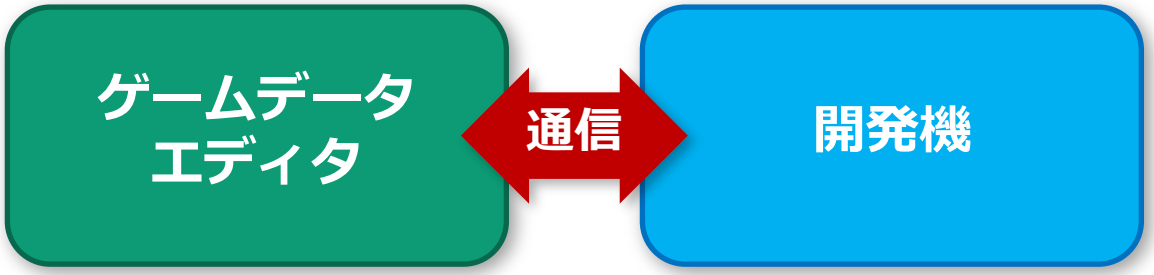

データ
構造定義
ファイル
データ
ファイル
データ
構造エディタ
データ
エディタ
編集環境
データ
最適化
ゲームデータ
エディタ
ゲーム用
構造定義
ソース
& ヘッダ
コンパイラ
通信
ゲーム
確認環境
最適化
データ

❖ゲームエディタの具体的な利用（１）

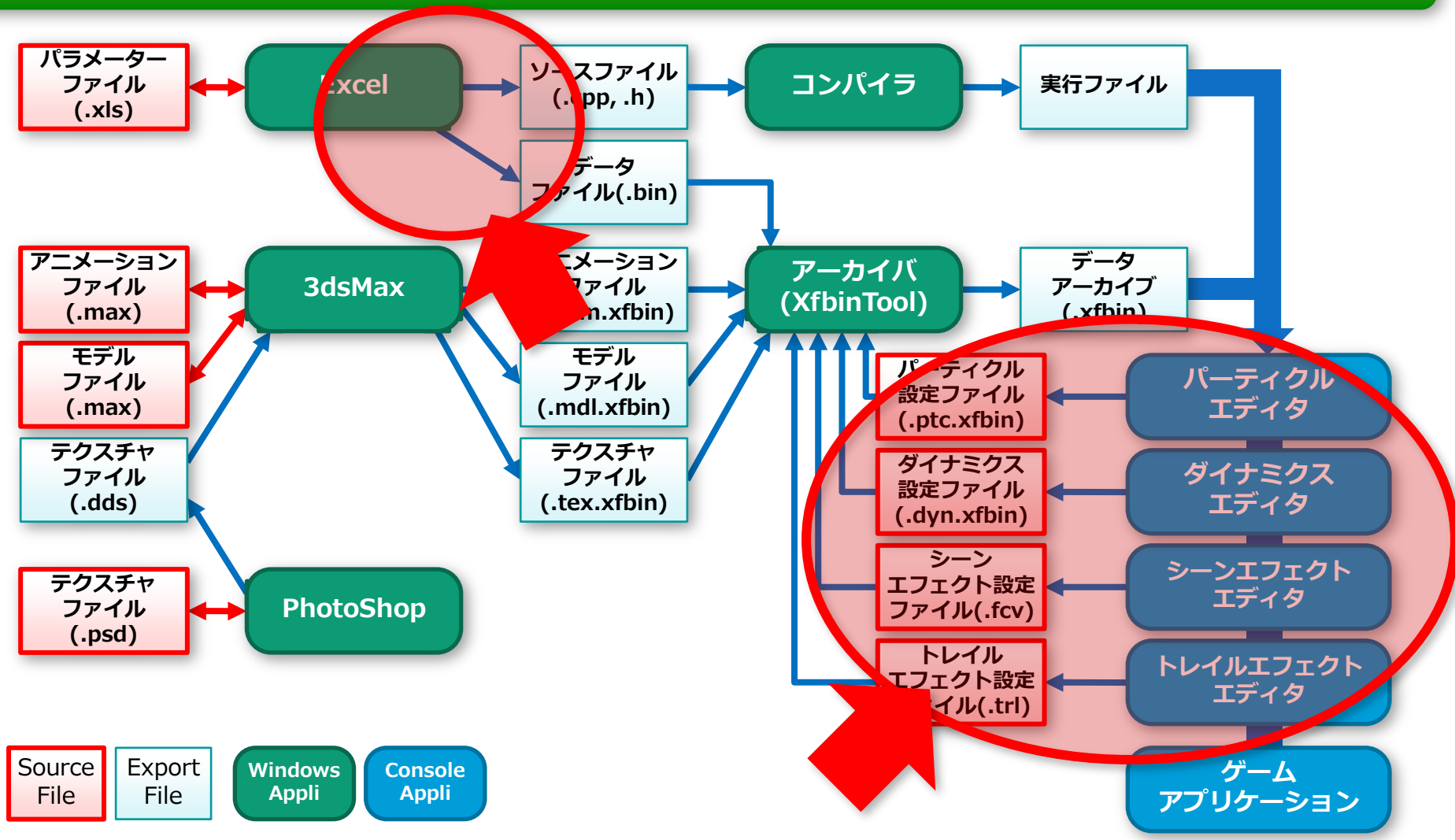
- エクセルシートの内容をGUIで調整したい!!
  - データエディタを介してデータを更新する
  - リアルタイムに確認できる等

❖ゲームエディタの具体的な利用（２）

- ゲーム側のデバッグ用のメニューが複雑!!
- ゲーム側のエディタでは使いにくい
  - Windows上のGUIエディタで設定
  - Undoなども可能

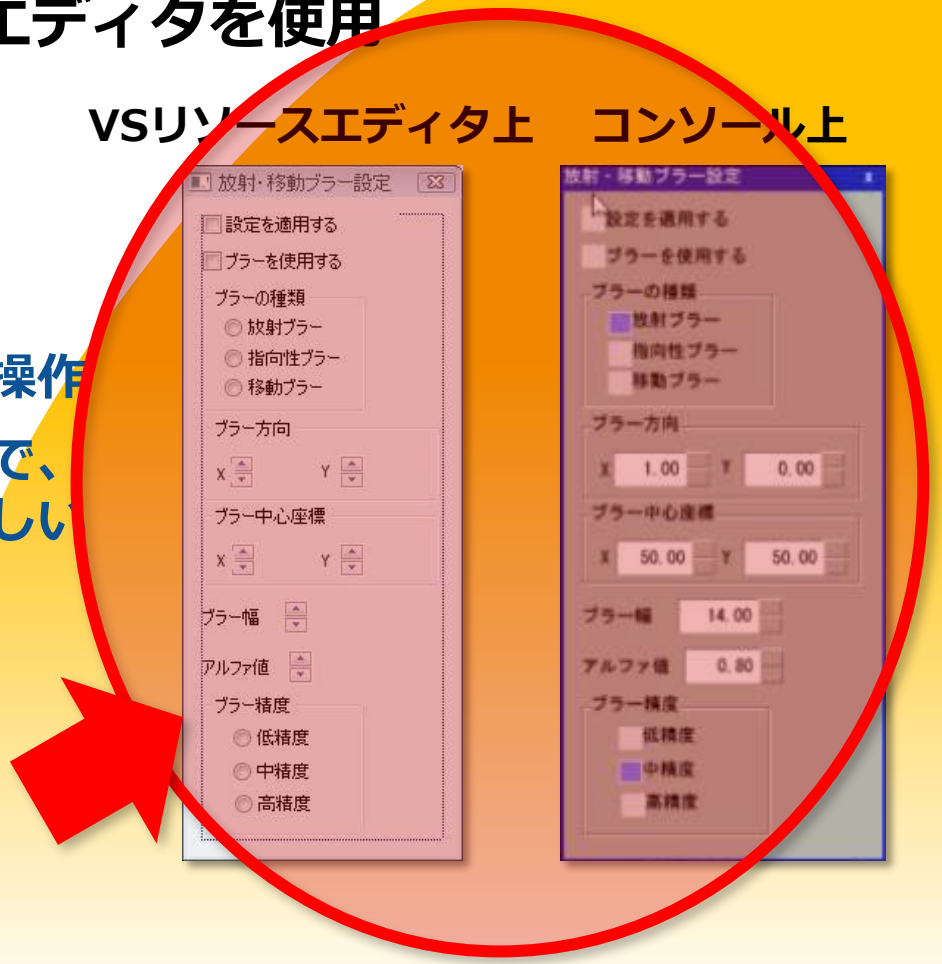


❖第三世代のワークフロー



❖GUI Toolkit の実装の問題

- ゲームアプリケーション側に実装
  - Visual Studioのリソースエディタを使用
    - ある程度は効率化
  - しかし問題も多い
    - 使用時に開発機が必要
    - パッド操作による扱いにくい操作
    - コンソールに依存しているので、他のコンソールへの移行が難しい



将来への展望

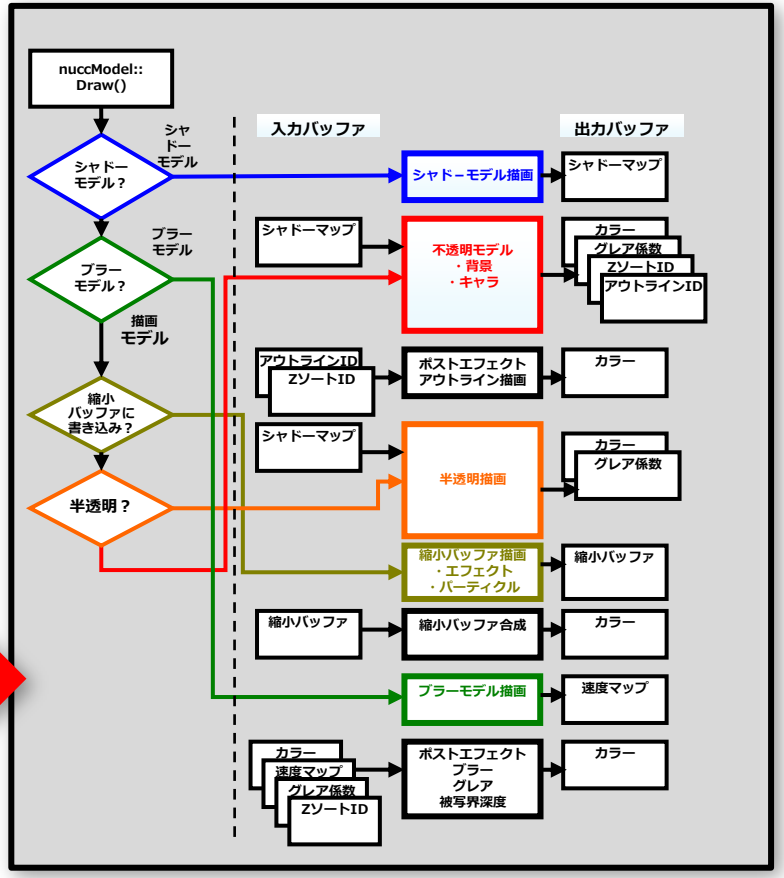
- The Prospects for the future -

❖レンダリングパイプライン エディタ

■ レンダリングパイプラインの問題

- メインコンソールから他のコンソールへ移植を行った際にレンダリングパイプラインを変更、調整する必要があった
- また、プロジェクトによっても最適化の為に同様の要求が

**ならば、
パイプラインを
つなげることで
変更が出来れば**



❖レンダリングパイプラインエディタ

- レイヤーの仕組みを応用
  - レイヤー単位でモジュール化
    - ステータス変更
    - 描画優先、ソート
    - エフェクト(ポストエフェクト)
    - 並列化指定
    - 使用可能シェーダー
  - レイヤーに入力バッファ、出力バッファを接続
  - アーティストは、オブジェクトに対して「レイヤー」と「シェーダー」を選択

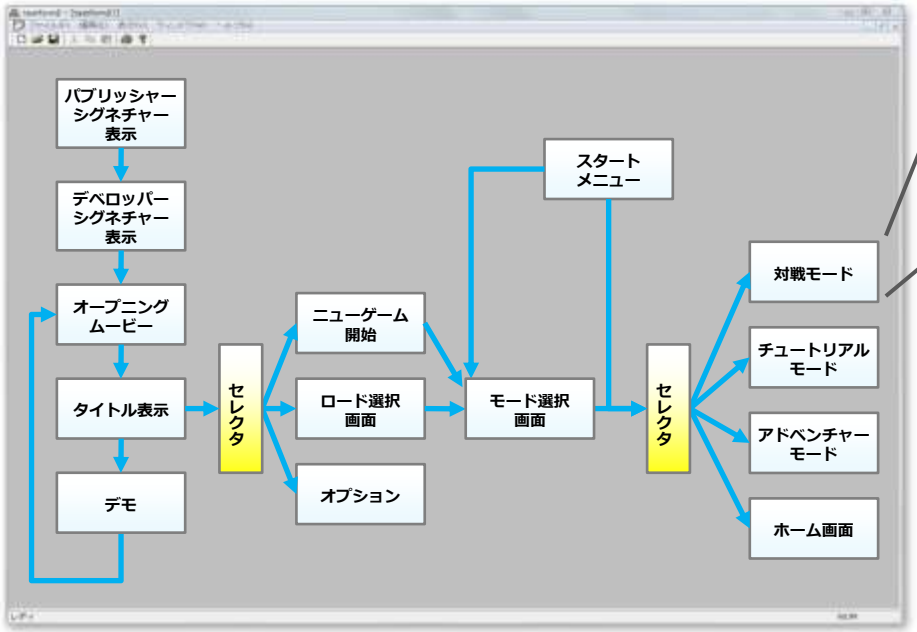
❖ゲームフロー エディタ

■ 仕様書＝ゲーム制作の実現

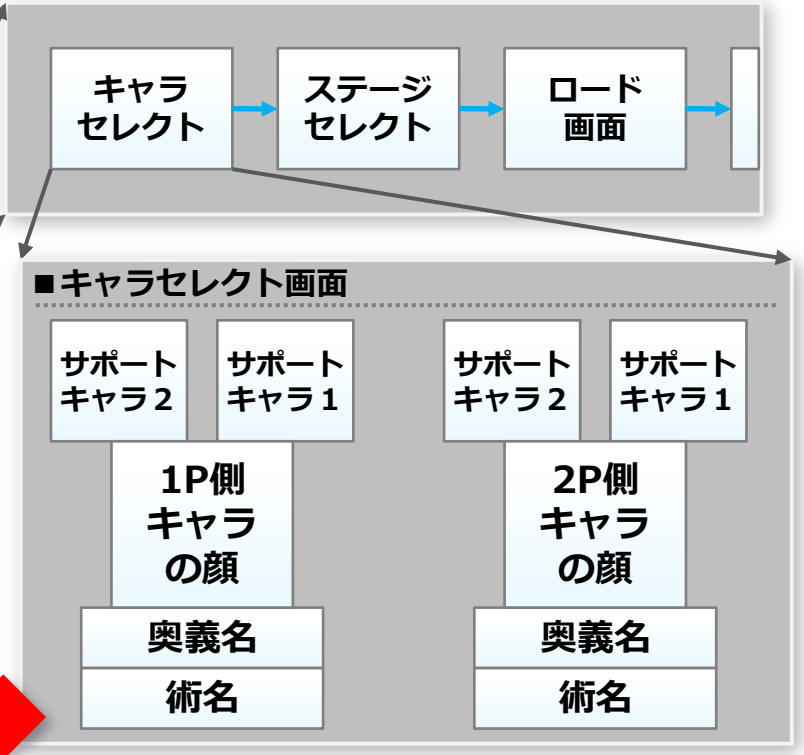
- ◆ どういうゲームのモードがあって、どういう流れがあって、その場面で何を表示すべきか？はゲームデザイナーが考える。
  - それを結局、プログラマーが同じ形で実装をする。
- ◆ ゲームフローの制御はほとんどがswitch～case文と関数
  - 短調かつ、面倒なだけのプログラム
  - プログラムだと、全体の見通しも悪い
  - ゲームモードの結合にバグが出やすい

だったら、
ゲームデザイナーがGUIで書いたものが
そのまま実行できる仕組みがあれば良いのでは？

❖ゲームフローエディタ



ゲームデザイナーが
フローチャート形式で記述



「何を表示するべきか」を記入

**アーティストはゲーム画面に対して
シーンを構築していく**

❖最終的な目標

今まで

アーティストの作る
シーンのイメージが中心となって
ゲームの制作を行う

→ アーティスト
ドリブン

これから

担当者それぞれが
ゲームを「デザイン」して
ゲームの制作を行う

→ デザイナー
ドリブン

まとめ
- Summary -

開発環境は大きく様変わりして
ゲームエンジンやミドルウェア主体の開発へ
移り変わっています

おくれていると言われる、日本の開発も

そういう場合の為にも、
社内での開発基盤を整える必要が
あると考えます

プロジェクト内容に応じて選択すべきだと思います

プロジェクトの規模やジャンルによって、
当然合わないものもあります

そのためにも、会社の問題点を洗い出し

1．体制を見直そう

効率化は技術的な問題だけではない
むしろ、体制や運用方法が重要

2．ポイントを絞ろう

会社にとって効率化のポイントはどこか？

3．小さいことから始める

大きい設計はリスクも多い
小さなを広げていく事が重要

開発の効率化は出来るはず!!