ゲーム開発技術の現象学

三宅　陽一郎
yoichi-m@pk9.so-net.ne.jp

2009.2.8.(Sun) @ 東京工業大学CIC

y.m.4160@gmail.com
Twitter: miyayou

Contact Information

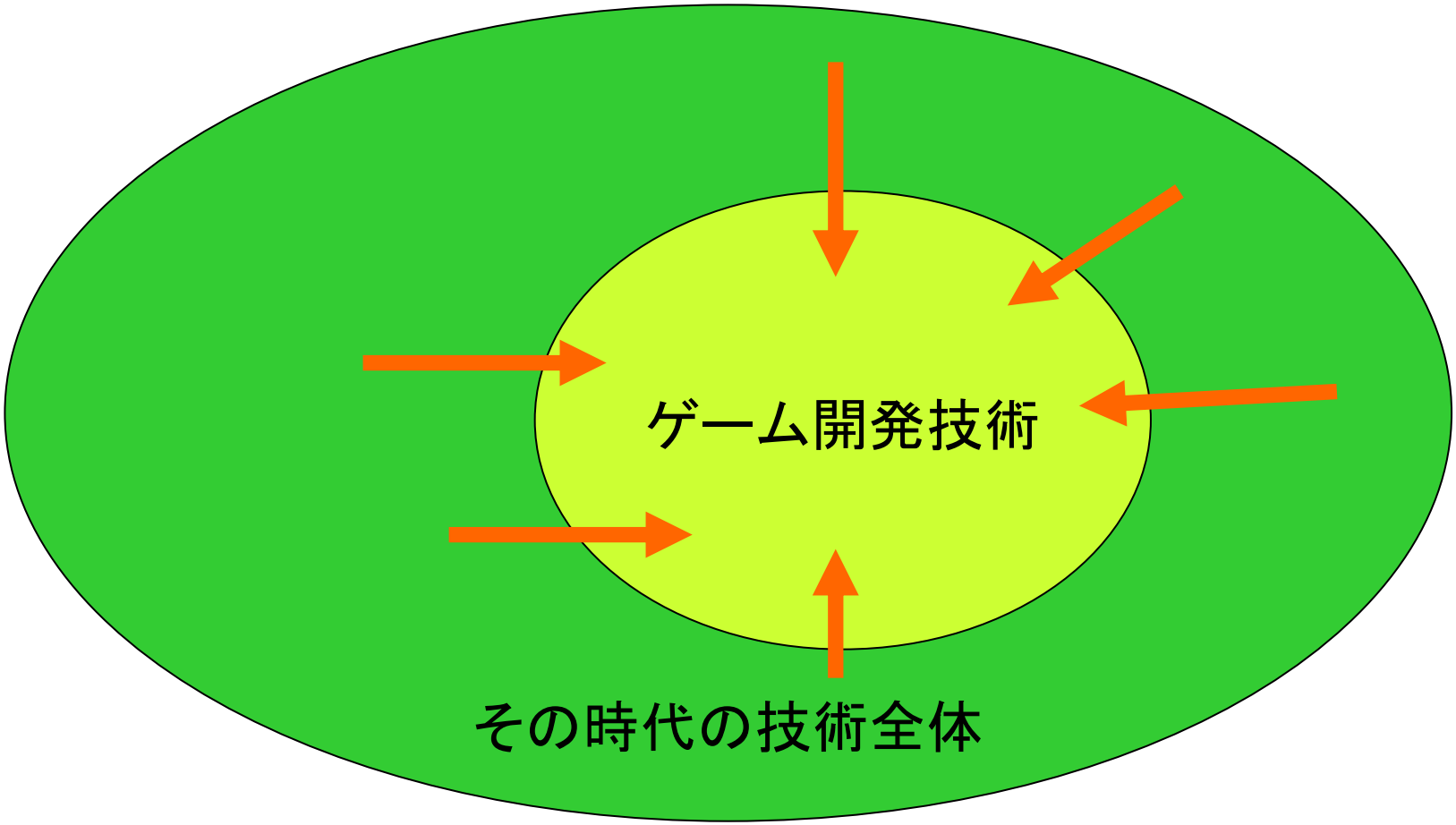
Youichiro Miyake

- Mail: y.m.4160@gmail.com
- Twitter: @miyayou
- Blog: http://blogai.igda.jp
- LinkedIn: http://www.linkedin.com/in/miyayou
- Facebook: http://www.facebook.com/youichiro.miyake

目次

第1章　ゲーム開発技術史　概要

ゲーム技術史を捉えるポイント



ゲーム開発発祥の技術はない。
ゲーム開発技術はハードウェア・ソフトウェアともに、
他分野で育まれた技術を発展・適応させて導入される。

ゲーム機の技術史　vs PC

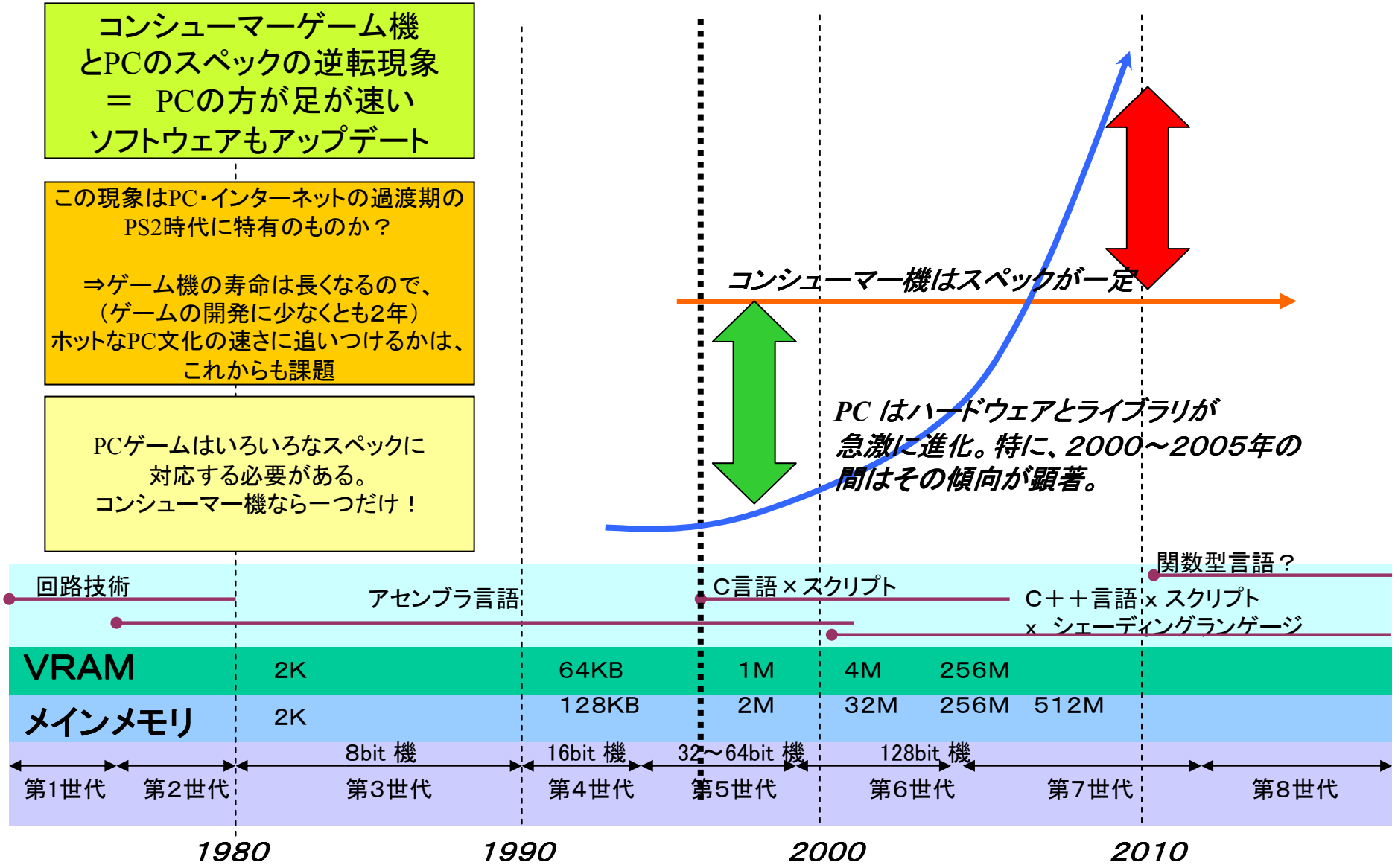
それぞれの時代でゲーム機として結晶できる技術の総体としてコンシュマーゲーム機は作られて来た

コンシューマーゲーム機
とPCのスペックの逆転現象
＝　PCの方が足が速い
ソフトウェアもアップデート

この現象はPC・インターネットの過渡期の
PS2時代に特有のものか？

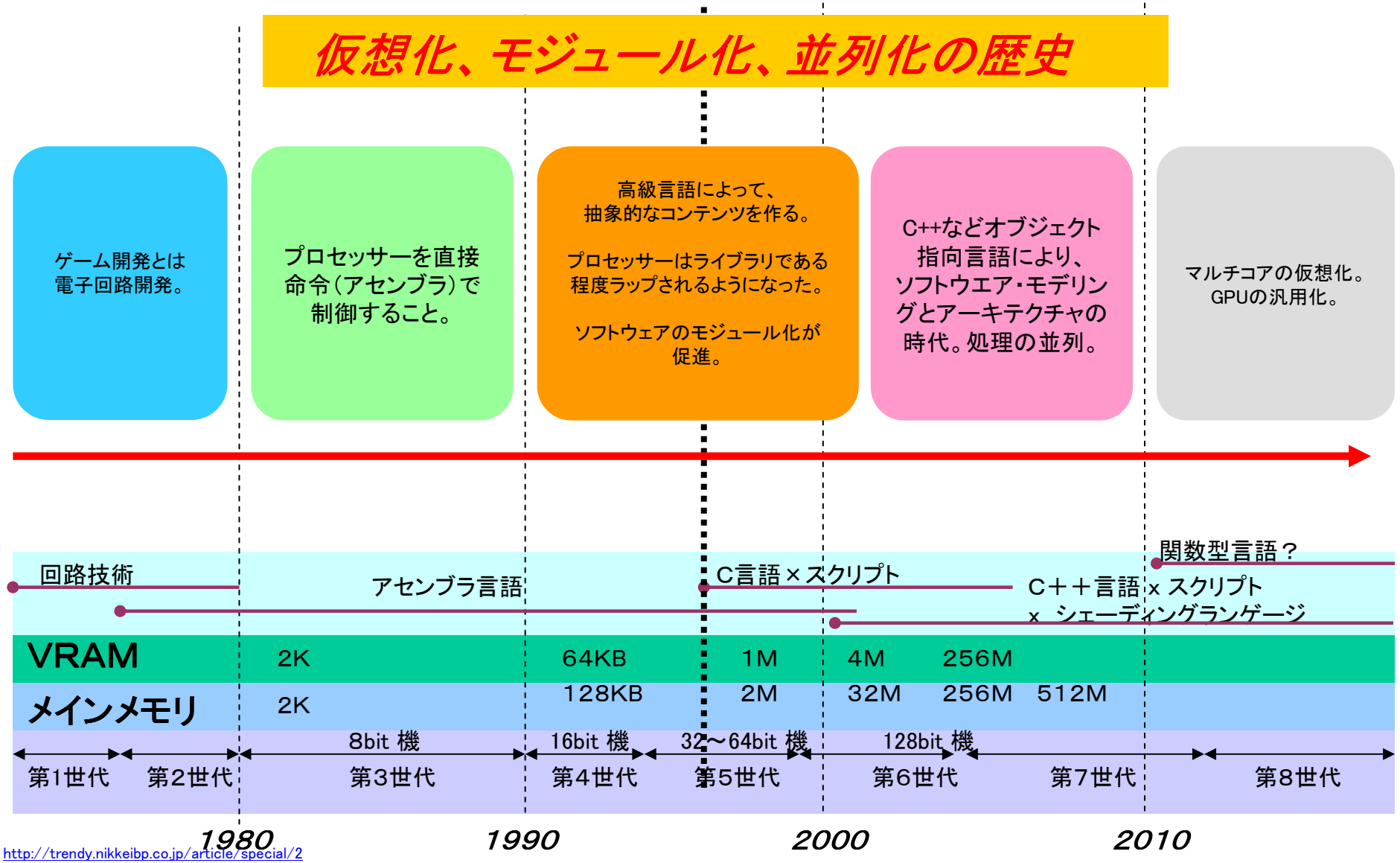
⇒ゲーム機の寿命は長くなるので、
（ゲームの開発に少なくとも2年）
ホットなPC文化の速さに追いつけるかは、
これからも課題

PCゲームはいろいろなスペックに
対応する必要がある。
コンシューマー機なら一つだけ！



プログラマーから見たゲーム機の歴史

それぞれの時代でゲーム機として結晶できる技術の総体としてコンシュマーゲーム機は作られて来た



http://trendy.nikkeibp.co.jp/article/special/20080318/1008218/?P=3

ゲーム・プログラム発展の3つのキーワード

(1)　仮想化 …　ハードウェアの物理的構造を
論理的な構造に見せること。

(2)　モジュール化　…　機能ごとに分かつこと。

(3)　並列化　…　複数の処理を同時実行すること。

…とは言っても、プログラミングの技術の発展そのものでもある。
（むしろ、導入としてはゲーム分野は最も遅い）

第1章　ゲーム開発技術史　概要

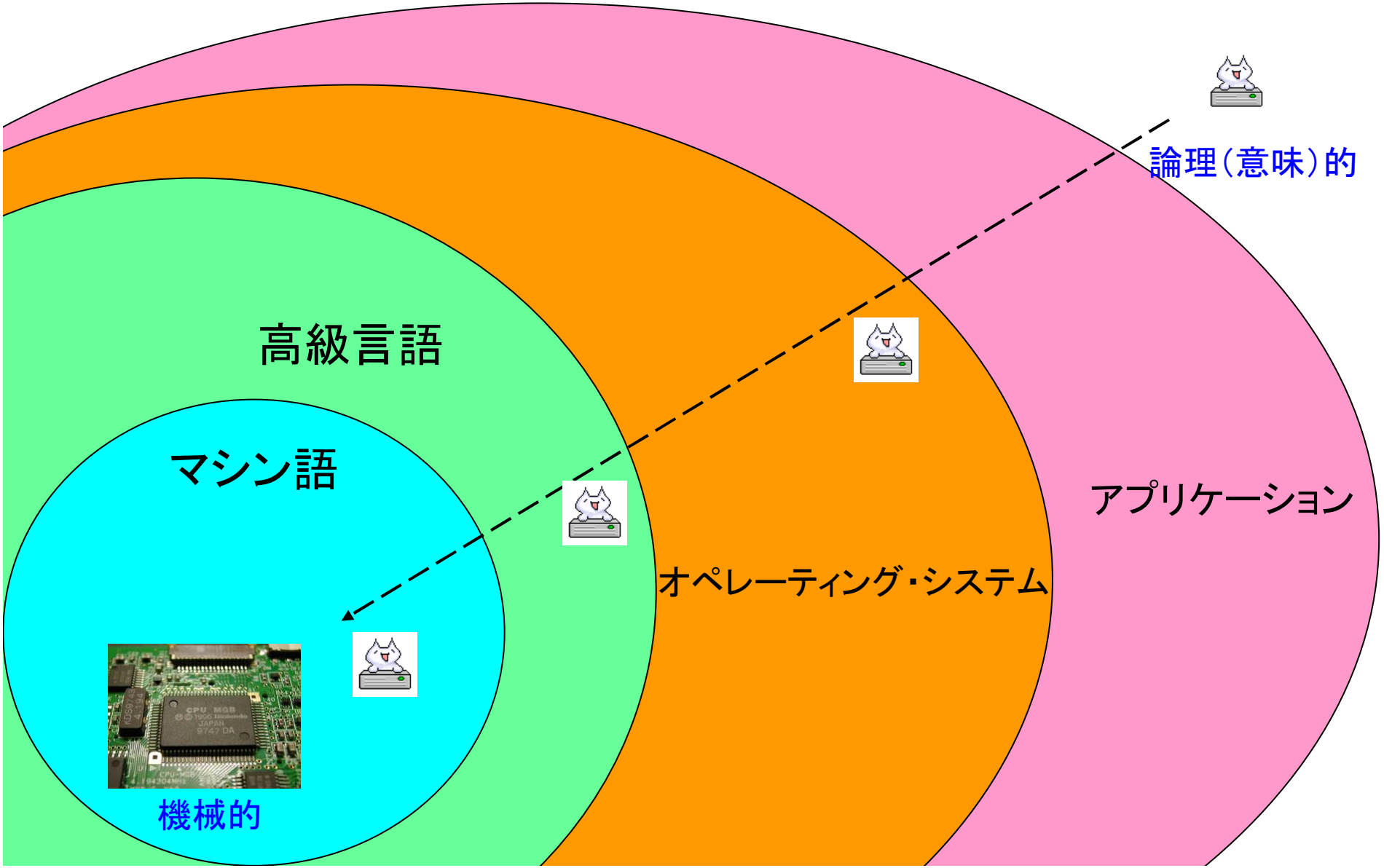
(1) 仮想化

(2) モジュール化

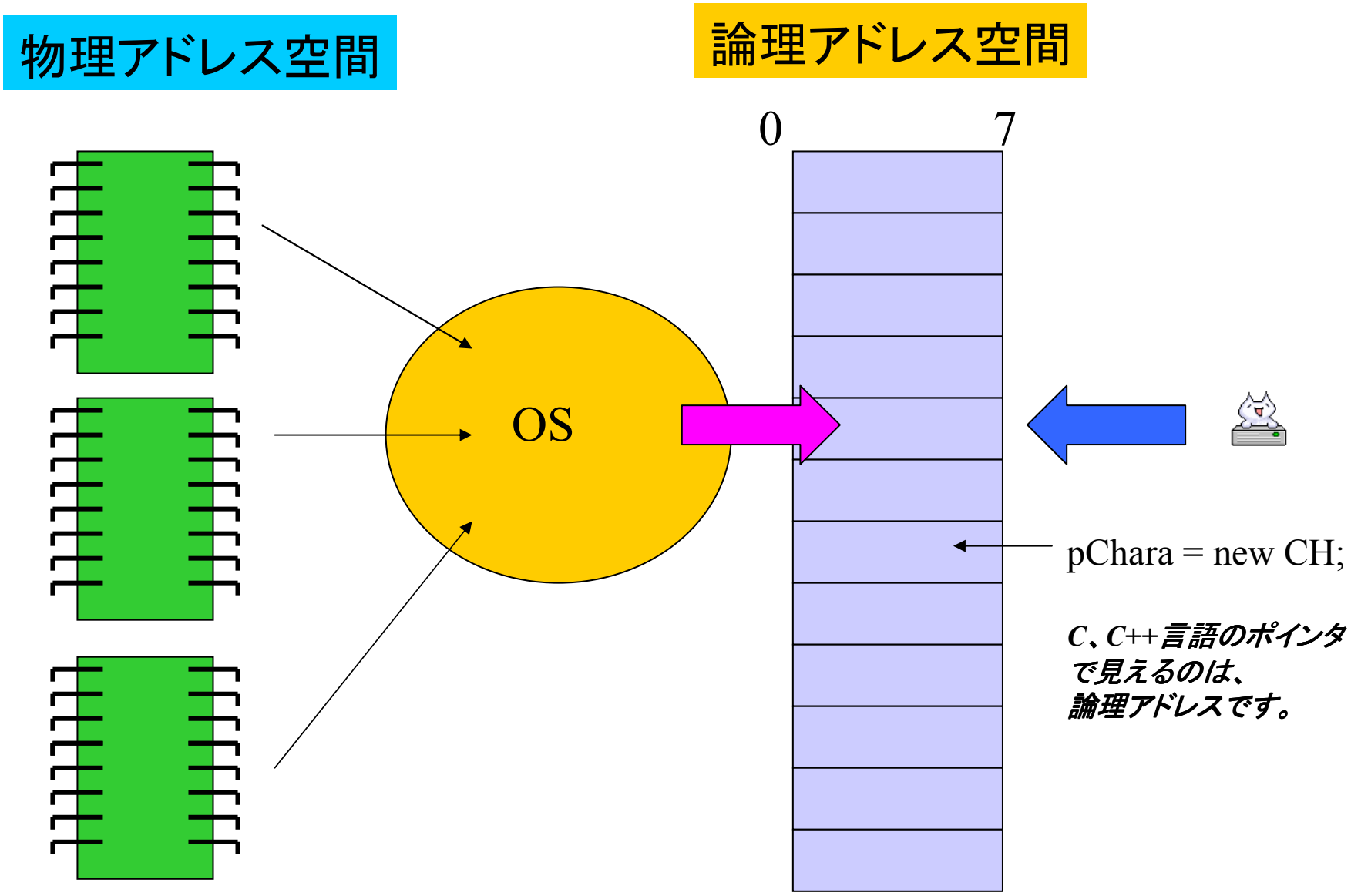
(3) 並列化

仮想化って何ですか？

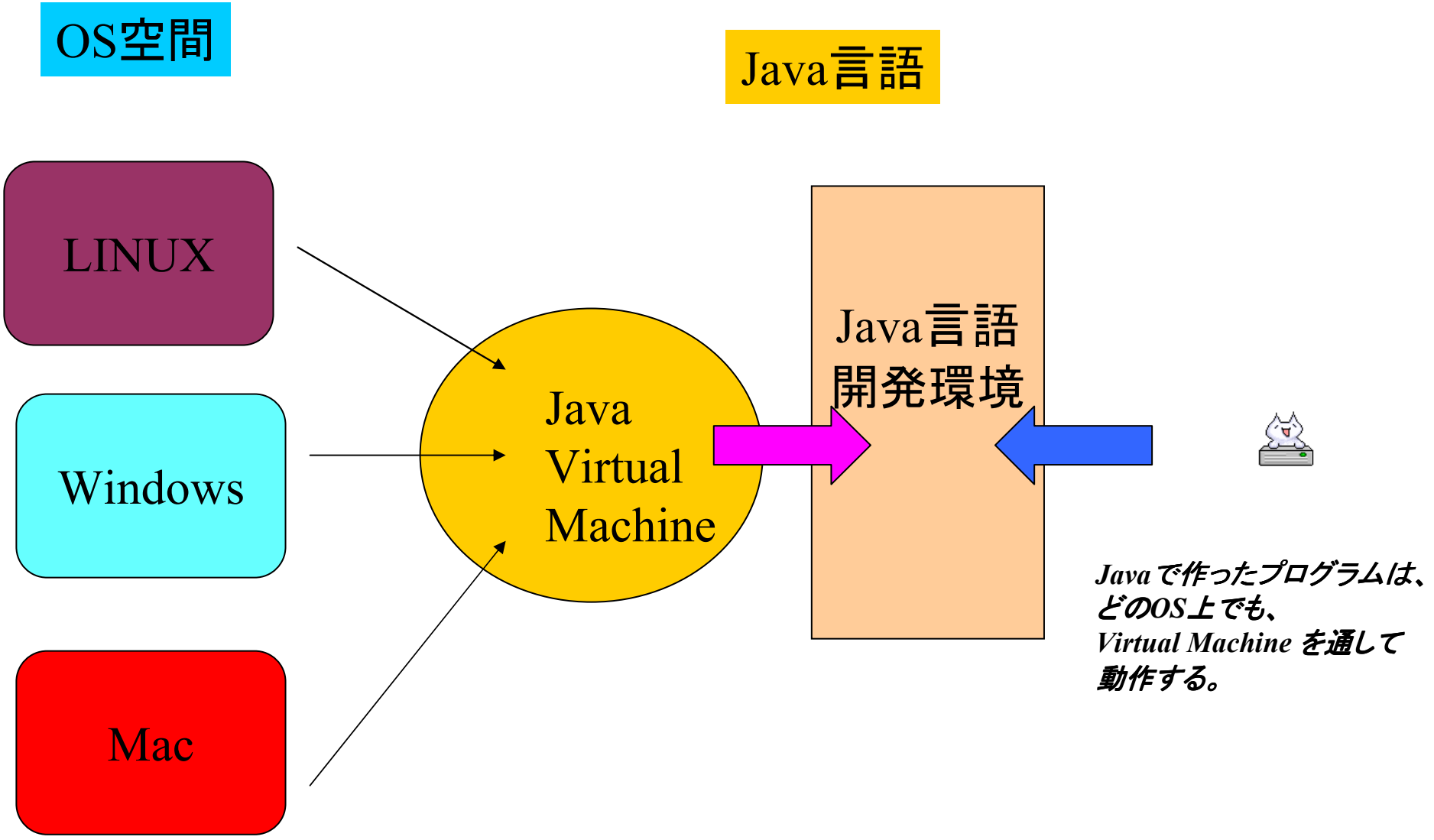
コンピューターの物理的特性を論理的に抽象化すること



（例）メモリの仮想化

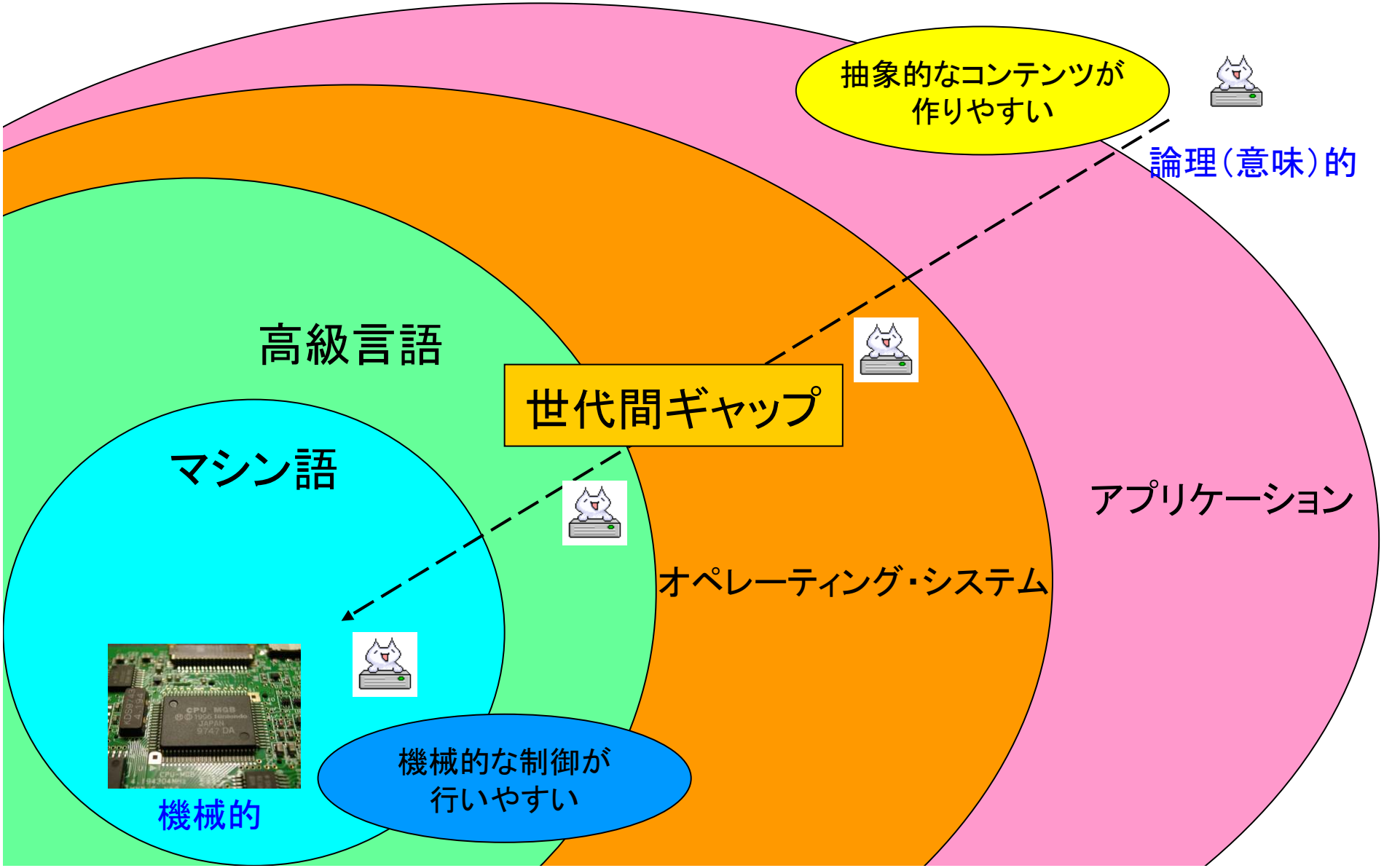

物理アドレス空間
論理アドレス空間
0
7
OS
pChara = new CH;
C、C++言語のポインタ
で見えるのは、
論理アドレスです。

（例）JavaによるOSの仮想化



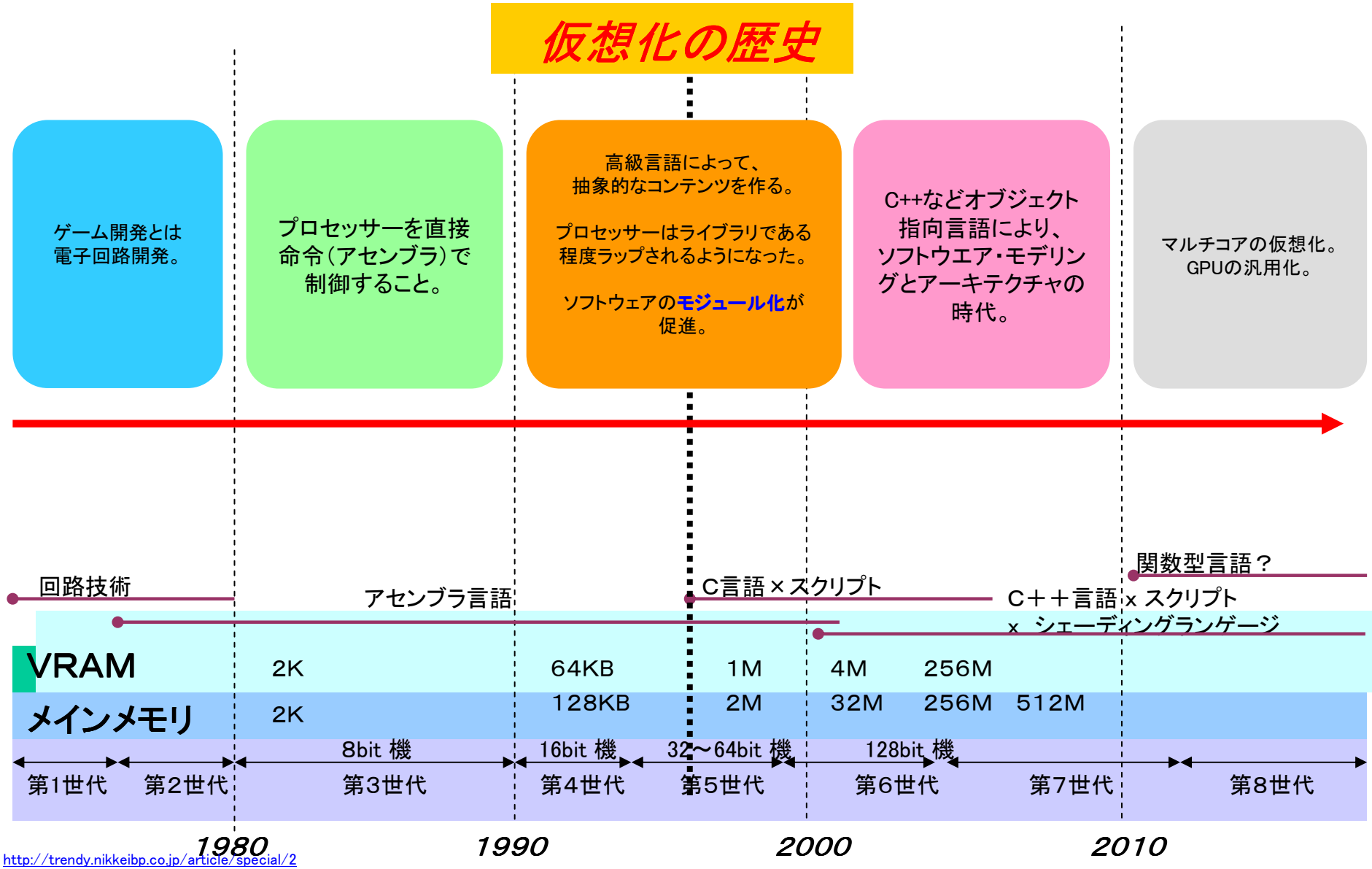
仮想化って何ですか？

コンピューターの物理的特性を論理的に抽象化すること



プログラマーから見たゲーム機の歴史

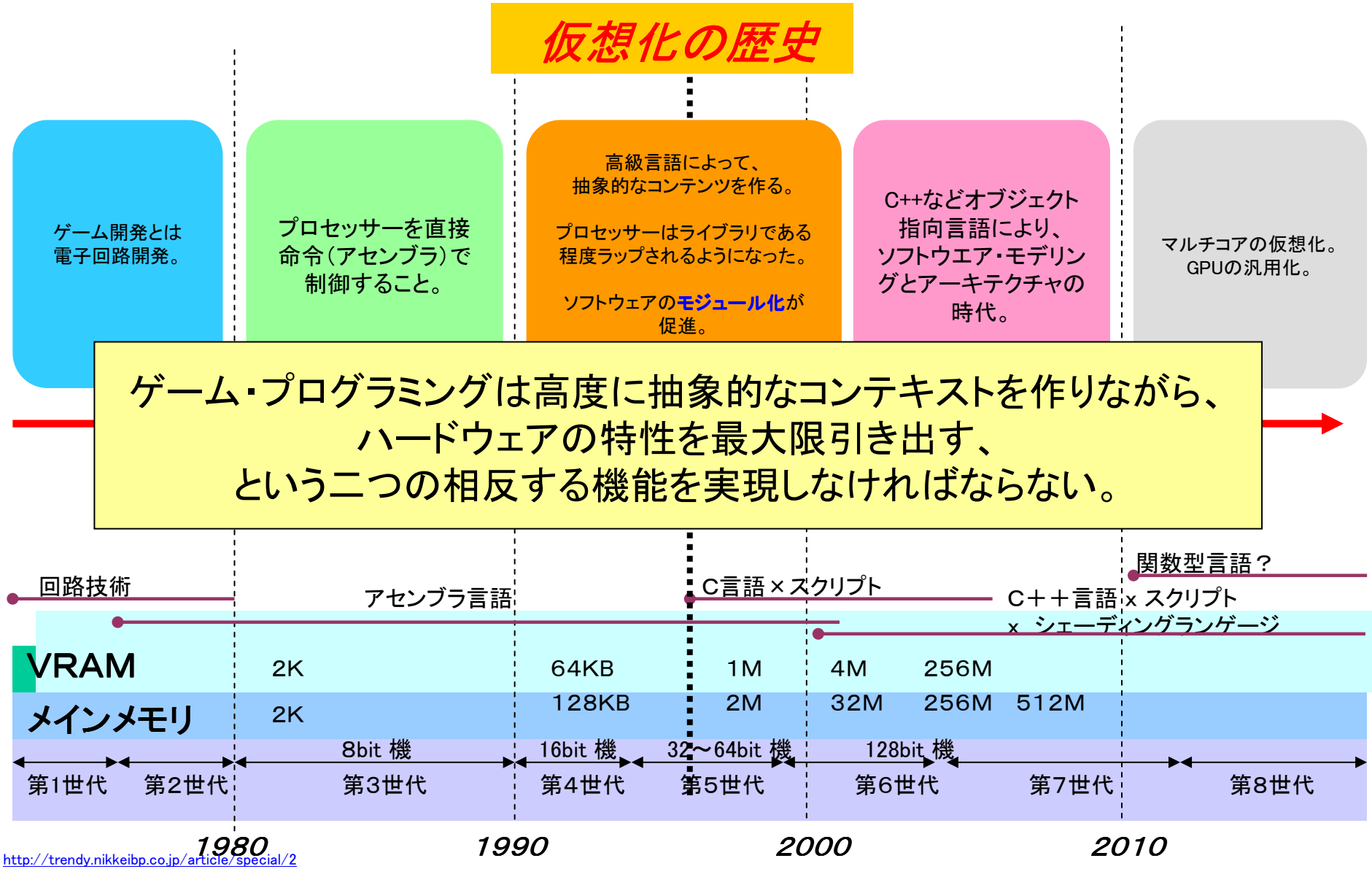
それぞれの時代でゲーム機として結晶できる技術の総体としてコンシュマーゲーム機は作られて来た



http://trendy.nikkeibp.co.jp/article/special/20080318/1008218/?P=3

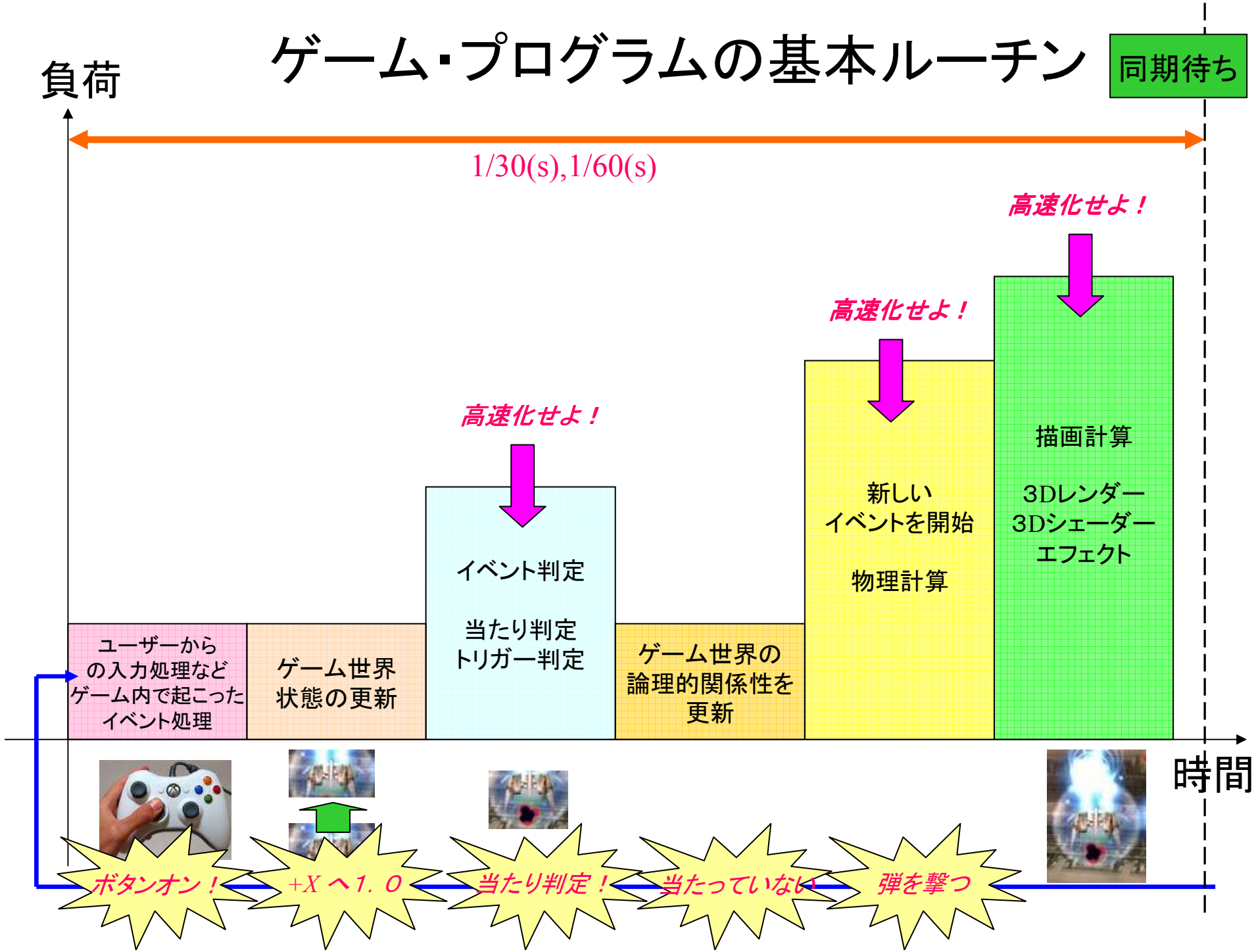
プログラマーから見たゲーム機の歴史

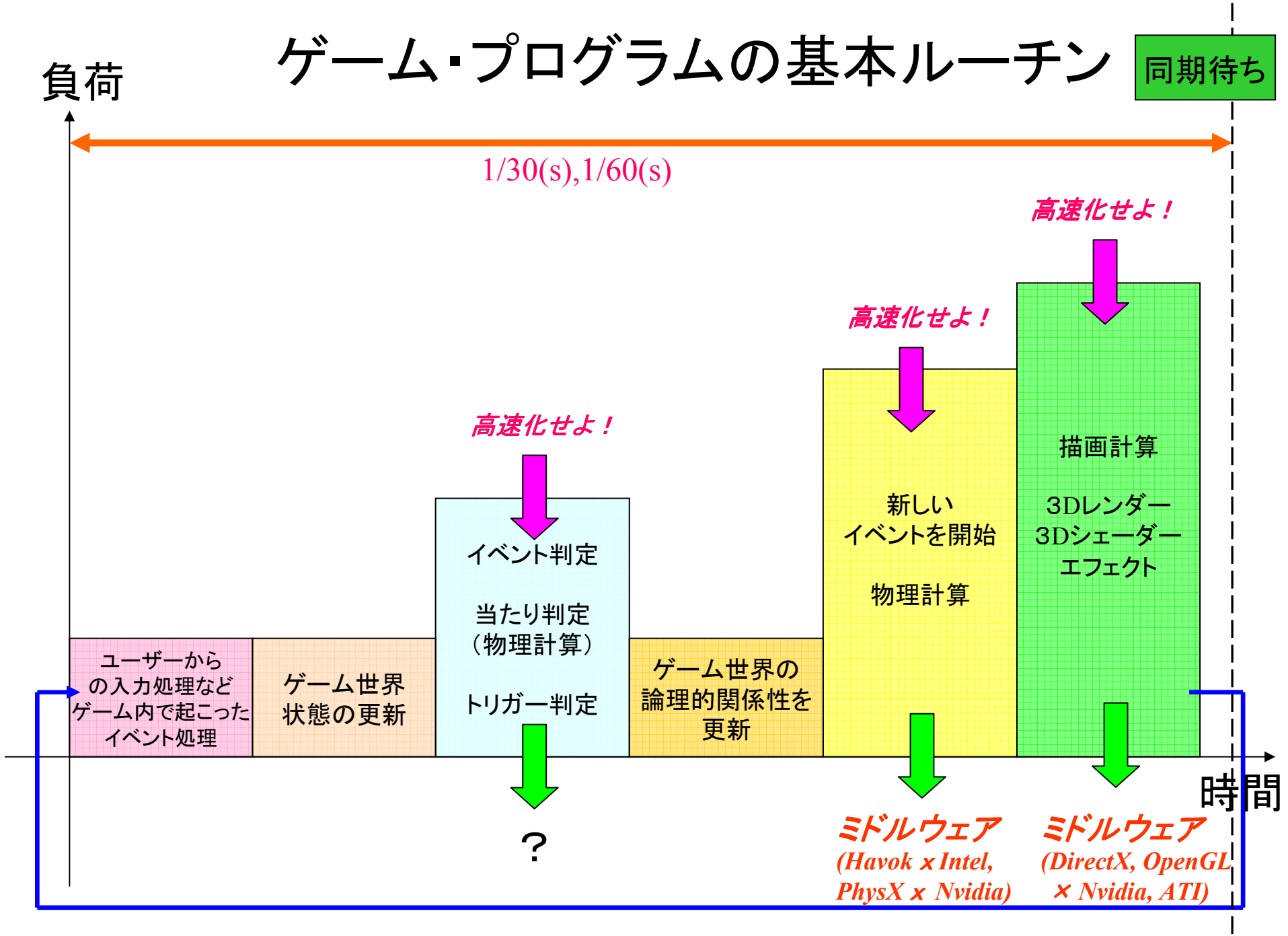
それぞれの時代でゲーム機として結晶できる技術の総体としてコンシュマーゲーム機は作られて来た



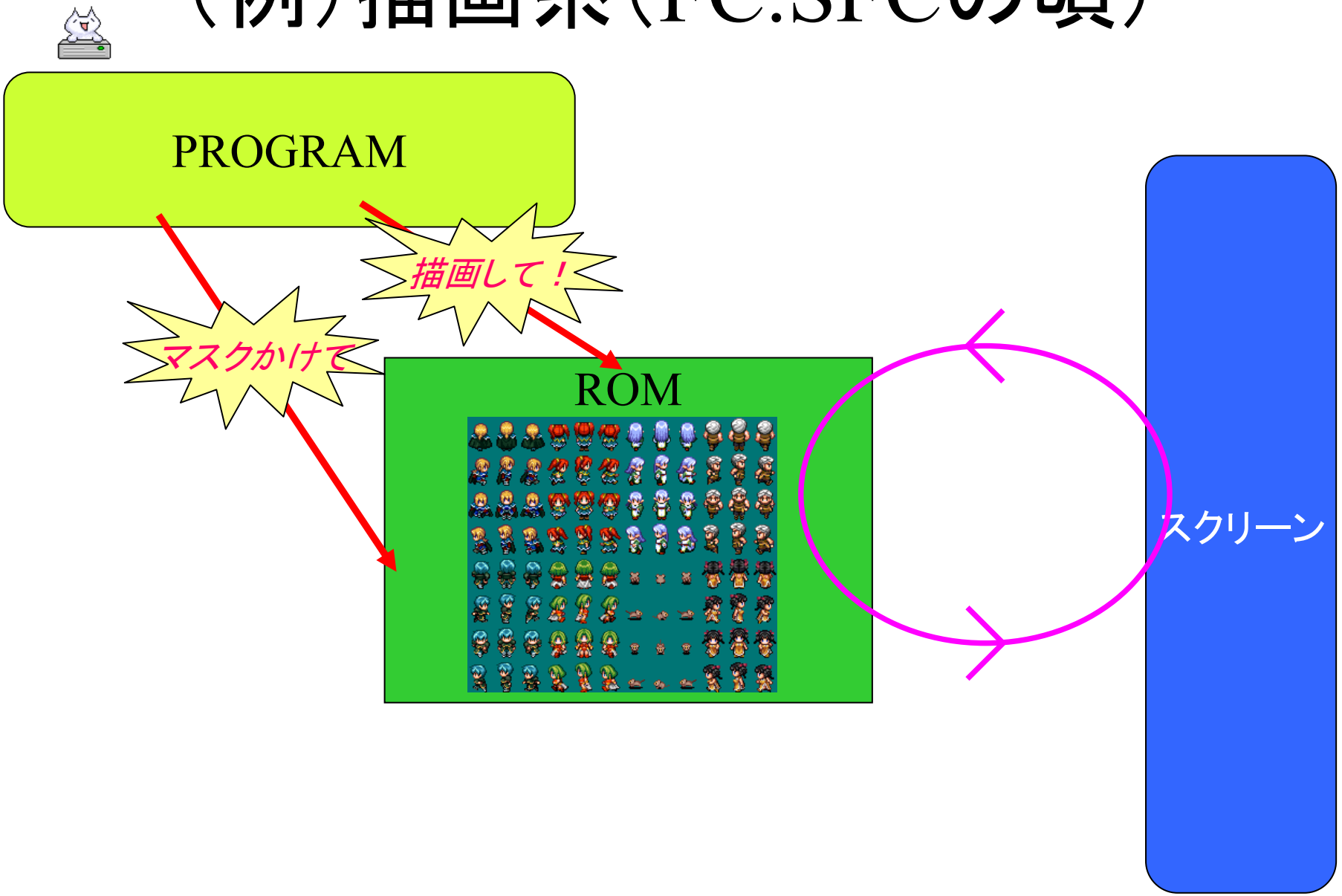
http://trendy.nikkeibp.co.jp/article/special/20080318/1008218/?P=3

ゲーム･プログラムの基本ルーチン

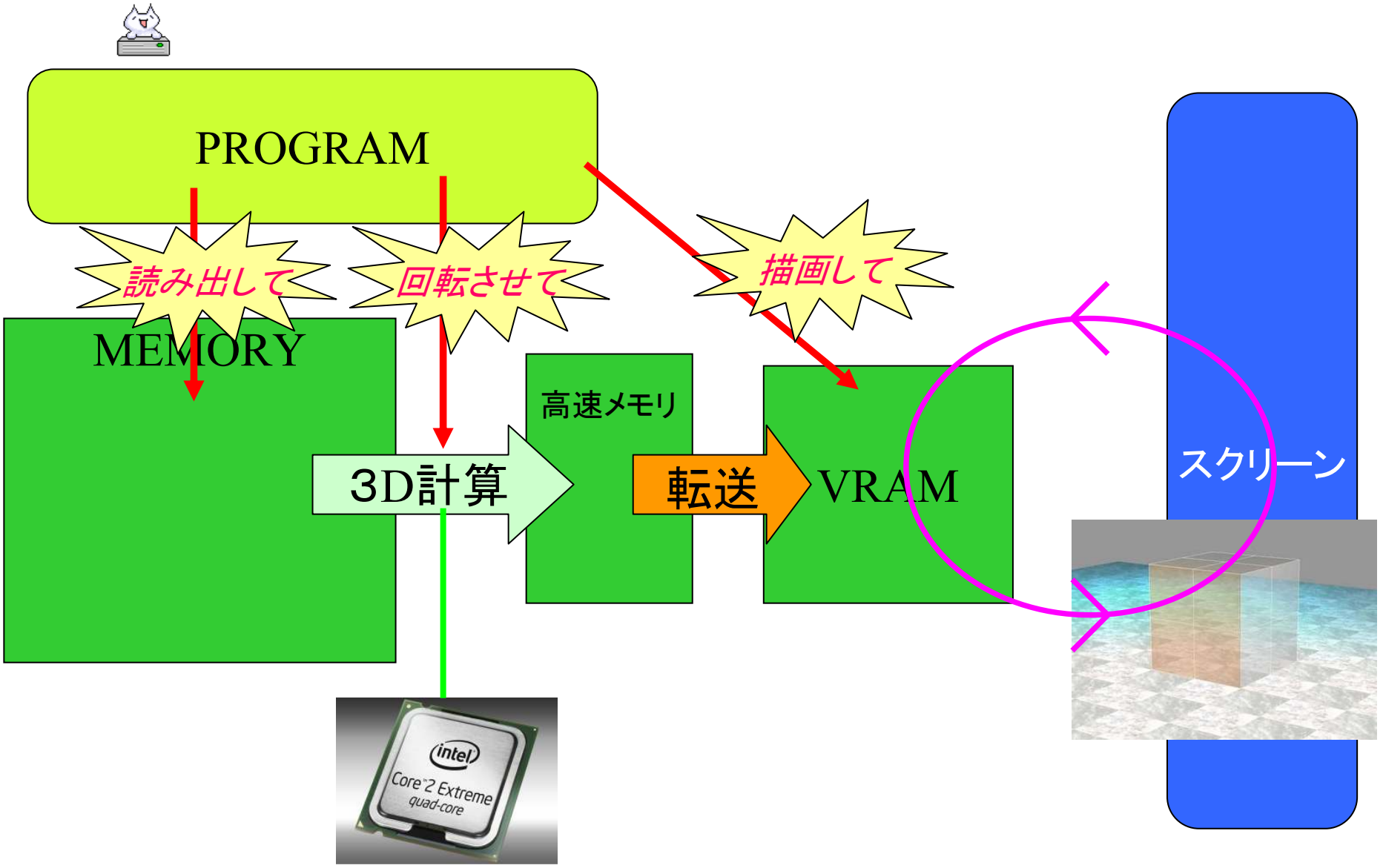



ゲーム･プログラムの基本ルーチン
同期待ち
負荷
1/30(s),1/60(s)
高速化せよ！
高速化せよ！
高速化せよ！
ユーザーから
の入力処理など
ゲーム内で起こった
イベント処理
ゲーム世界
状態の更新
イベント判定
当たり判定
（物理計算）
トリガー判定
ゲーム世界の
論理的関係性を
更新
新しい
イベントを開始
物理計算
描画計算
3Dレンダー
3Dシェーダー
エフェクト
？
ミドルウェア
(Havok x Intel,
PhysX x Nvidia)
ミドルウェア
(DirectX, OpenGL
x Nvidia, ATI)
時間

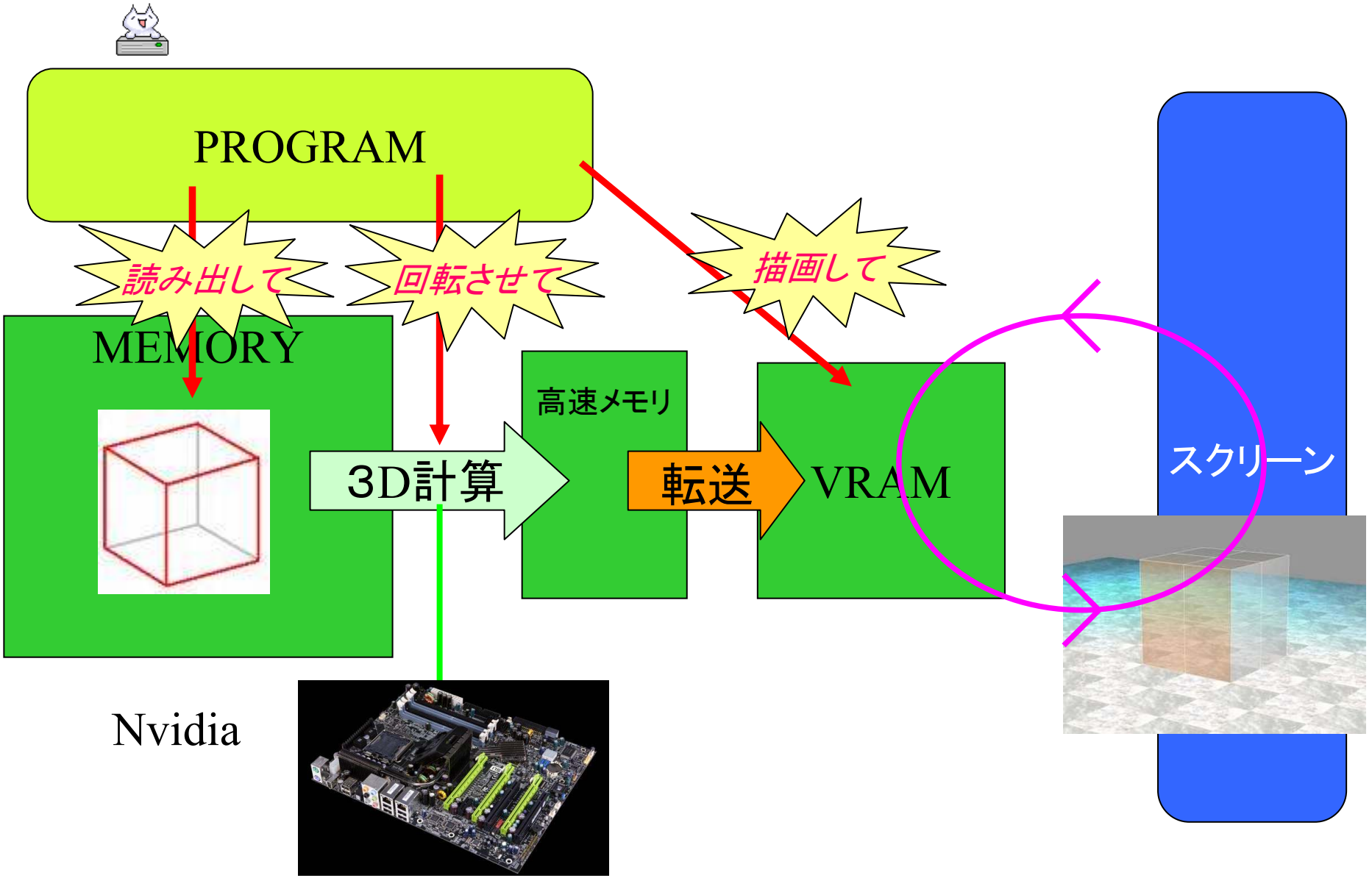
（例）描画系（FC.SFCの頃）



（例）描画系（ソフトウェア処理）



（例）描画系（ハードウェア処理）



3Dライブラリ（例）

$$\begin{bmatrix} \mathrm{Cos}\,\alpha\,\mathrm{Cos}\,\beta & -\mathrm{Sin}\,\alpha\,\mathrm{Cos}\,\beta & 0 \\ \mathrm{Sin}\,\alpha\,\mathrm{Cos}\,\beta & \mathrm{Cos}\,\alpha\,\mathrm{Cos}\,\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathrm{Sin}\,\beta$$

回転関数（こういう関数がたくさん）

$$\mathrm{Rot}(\alpha)\cdot\mathrm{Rot}(\beta)$$

回転操作（ライブラリなし）

記述が煩雑
最適化が必要
個人差が出る

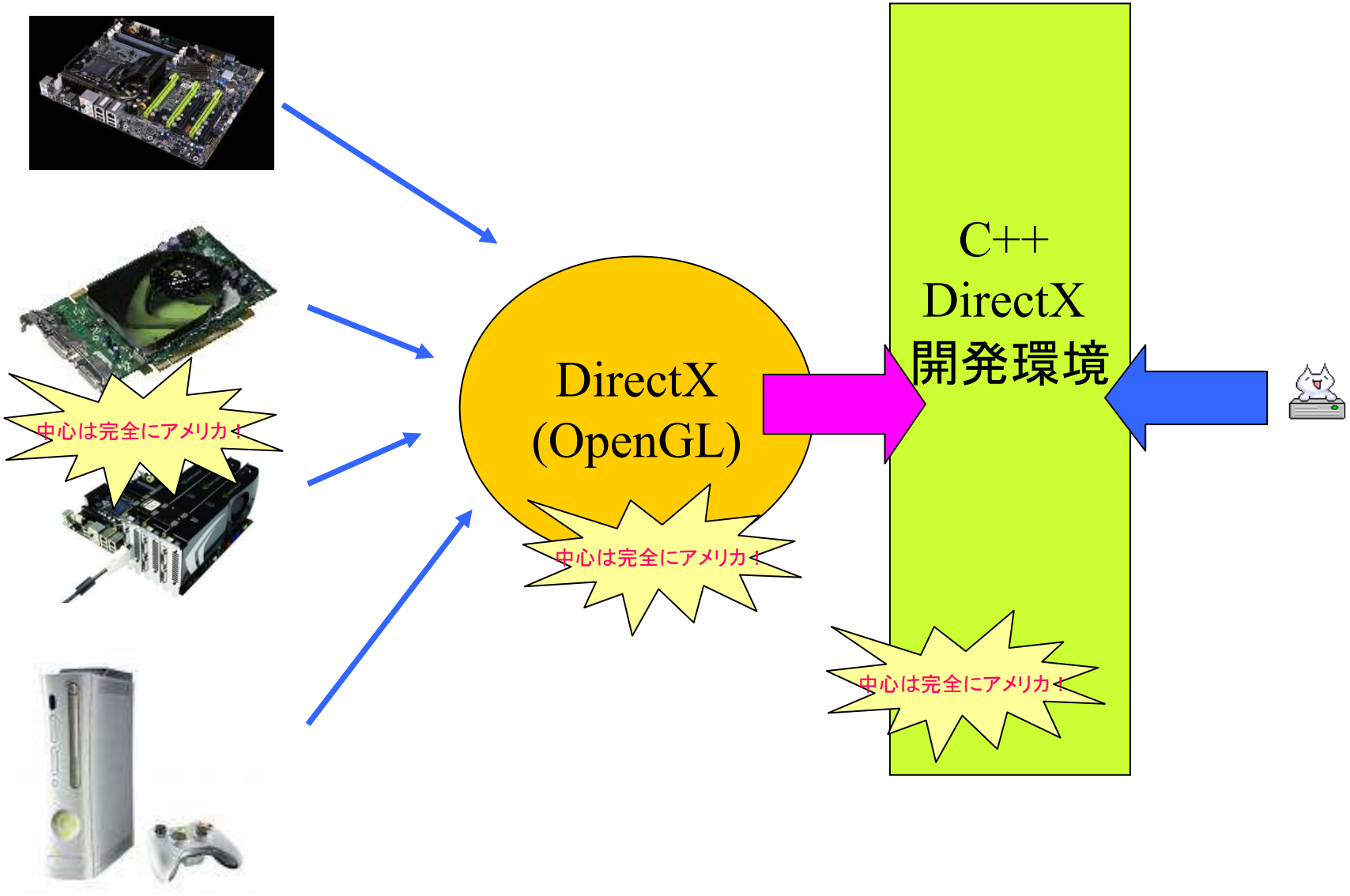
回転操作（ライブラリあり）

記述が美しい
最適化はライブラリがしてくれる
記述が統一

＋

ハードウェア（でなくてもある程度速い）

（例）3D計算のライブラリによる仮想化



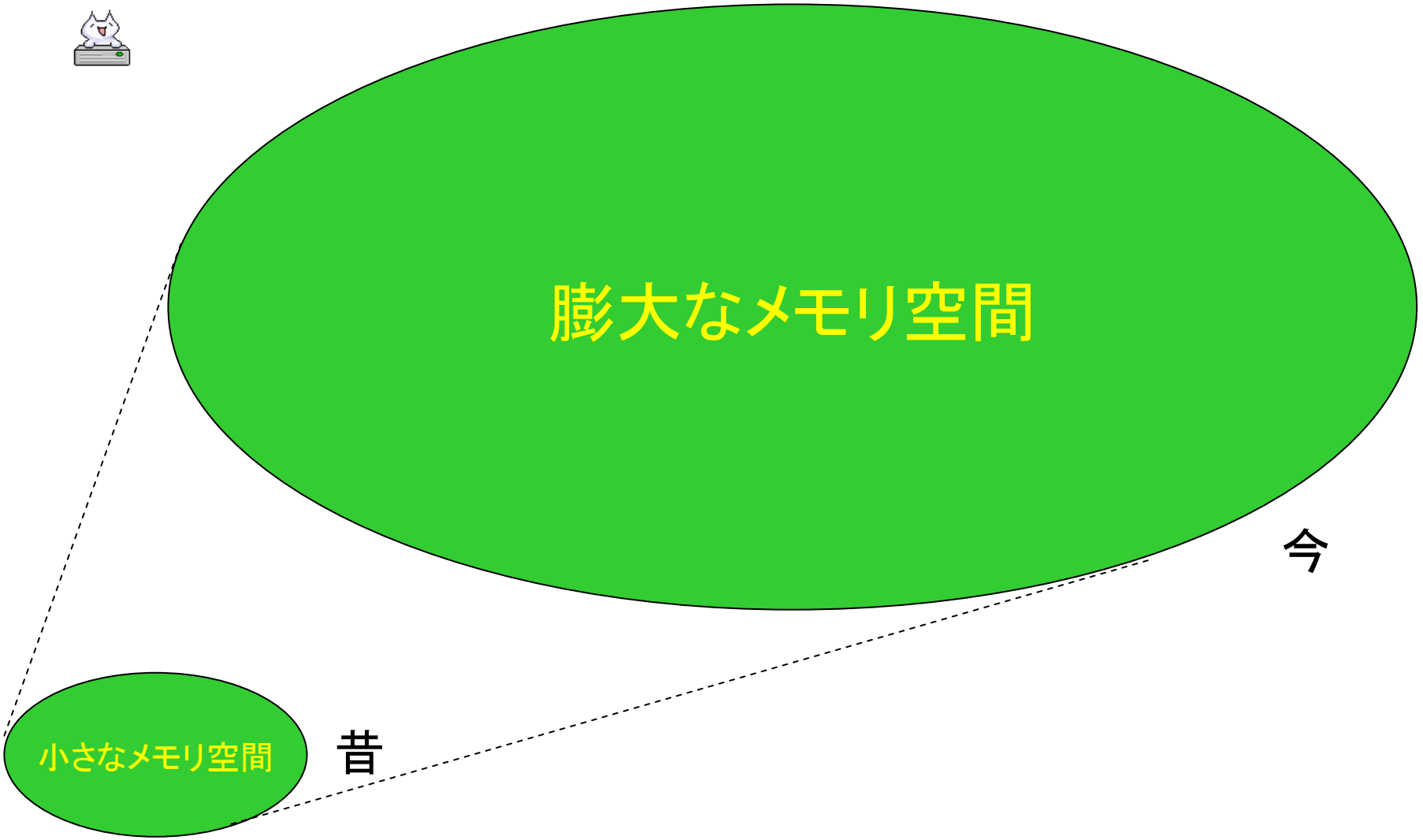
第1章　ゲーム開発技術史　概要

(1) 仮想化

(2) モジュール化

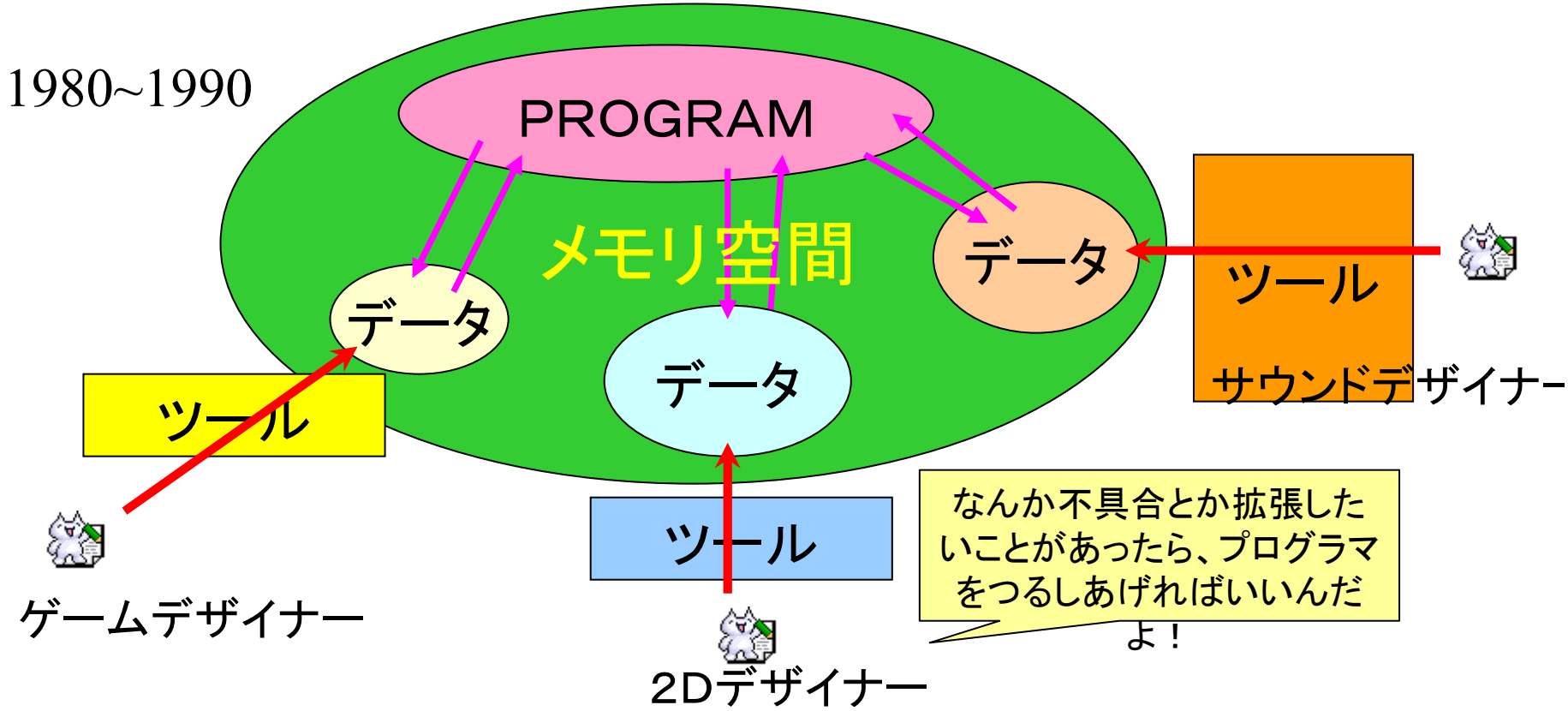
(3) 並列化

プログラマから見える風景



プログラマから見える風景

1人のプログラマがゲームプログラムからツール（CADやサウンドツールなど）まで作る。
そのツールを介して、デザイナーがデジタル空間にアクセスして創作する。

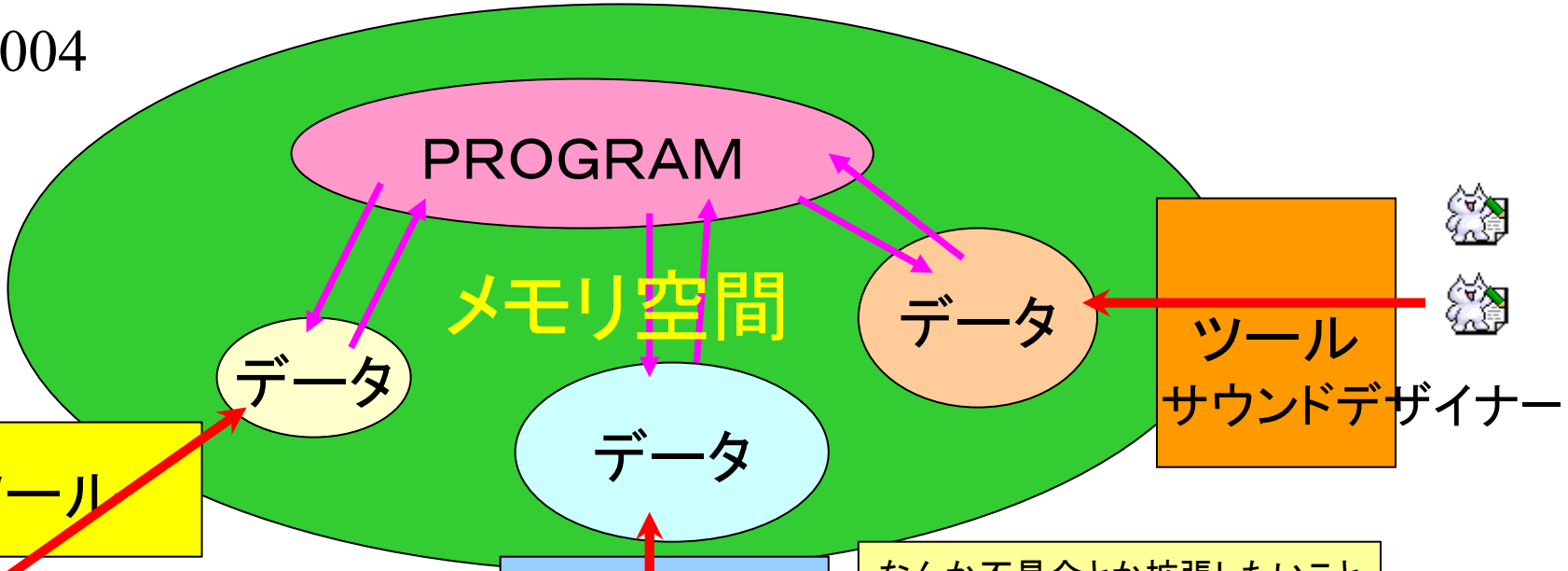


ツールはプログラマとデザイナーの仕事を明確に区別する、という機能を持つ。
しかし、実際はツールを巡って、デザイナーとプログラマが議論し合うことになる。

プログラマから見える風景

数人のプログラマがゲームプログラムからツールまで作る。
ミドルウェア製品も購入している。
そのツールを介して、デザイナーがデジタル空間にアクセスして創作する。

1990~2004



ゲームデザイナー

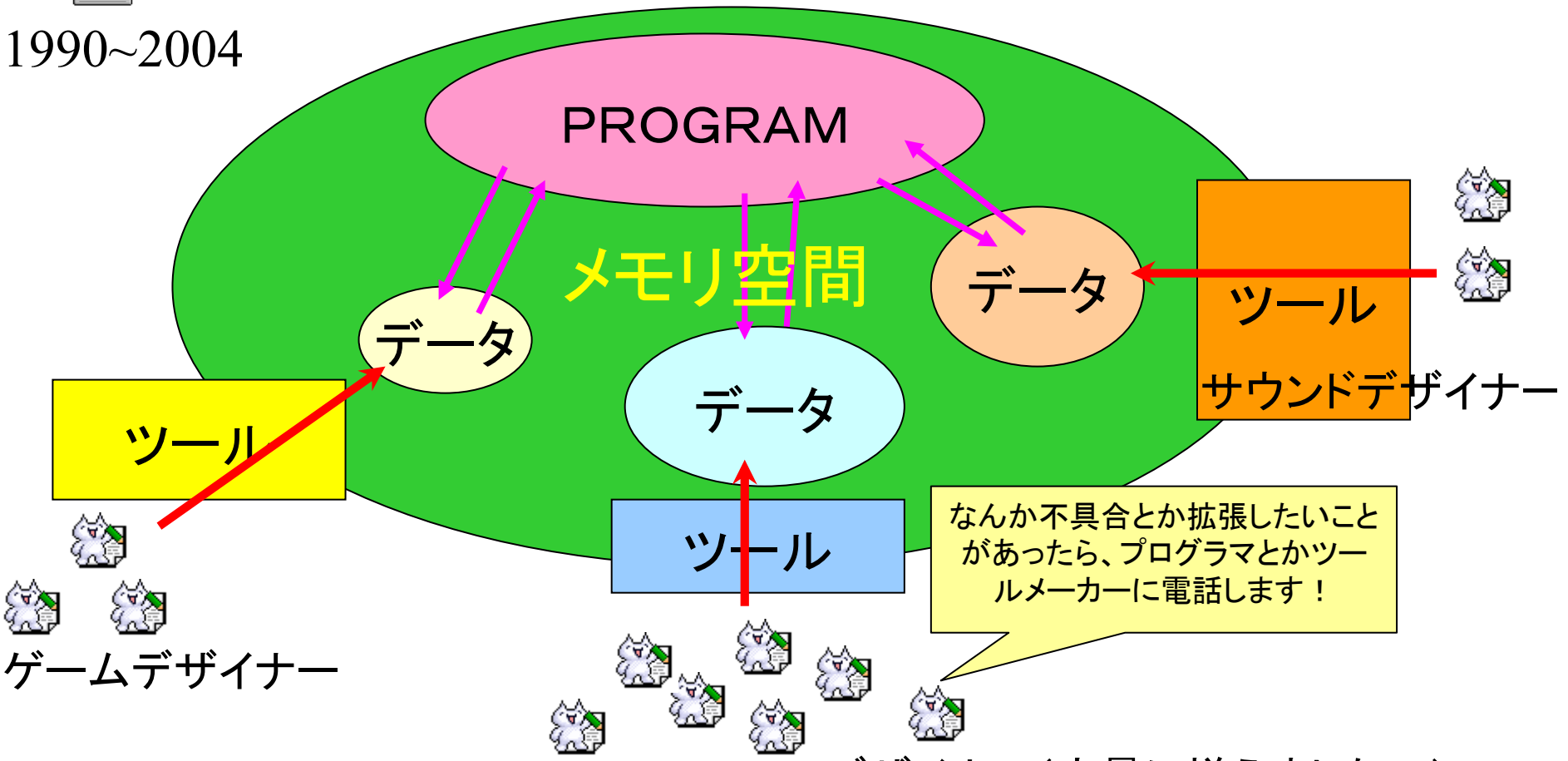
なんか不具合とか拡張したいことがあったら、プログラマとかツールメーカーに電話します！

2D、3Dデザイナー（大量に増えました！）

プログラマから見える風景

数人のプログラマがゲームプログラムからツールまで作る。
ミドルウェア製品も購入している。
そのツールを介して、デザイナーがデジタル空間にアクセスして創作する。

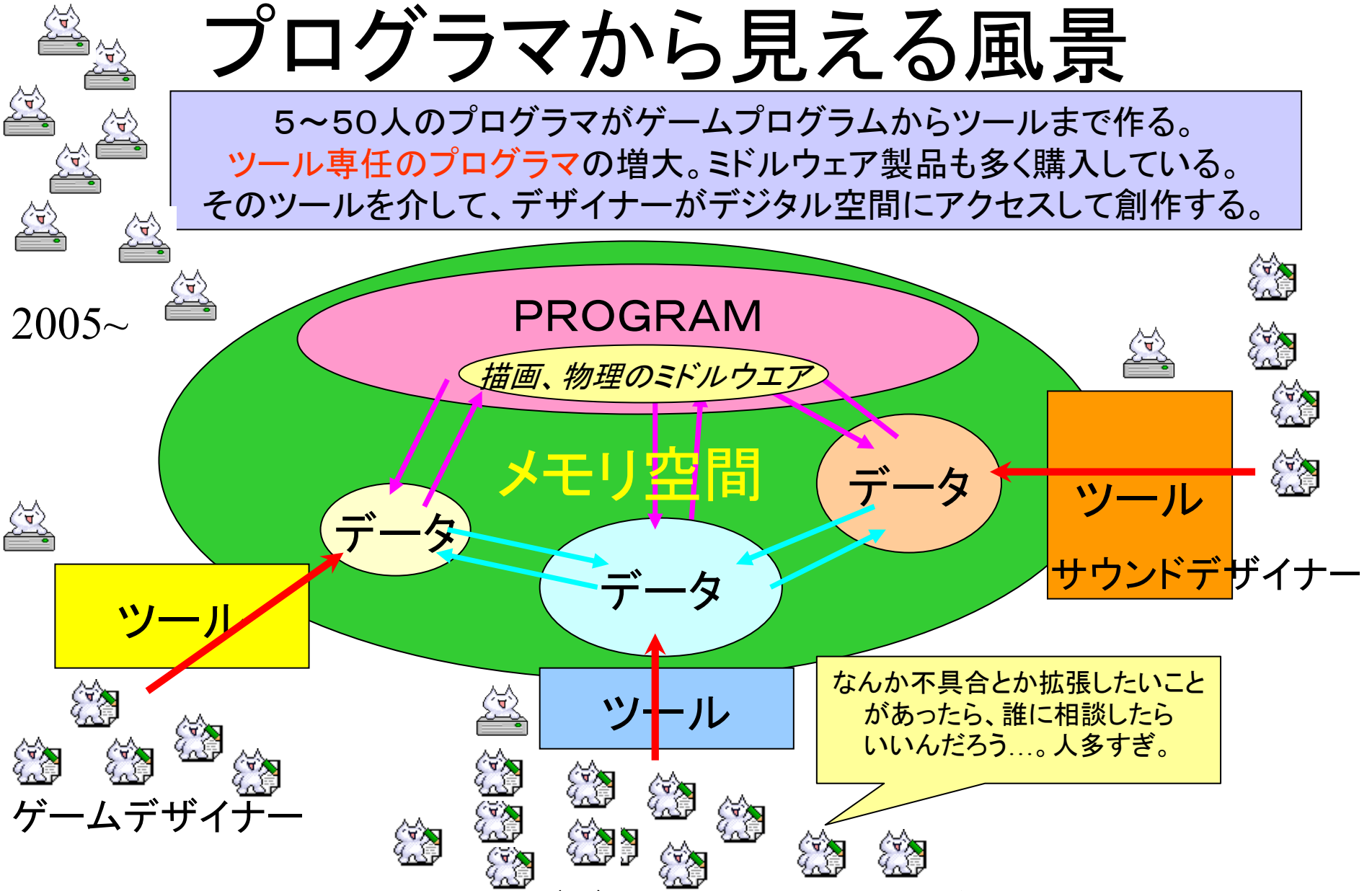
1990~2004



なんか人が増えて来たけど、まだぎりぎり人材管理しなくていいかなあ・・・

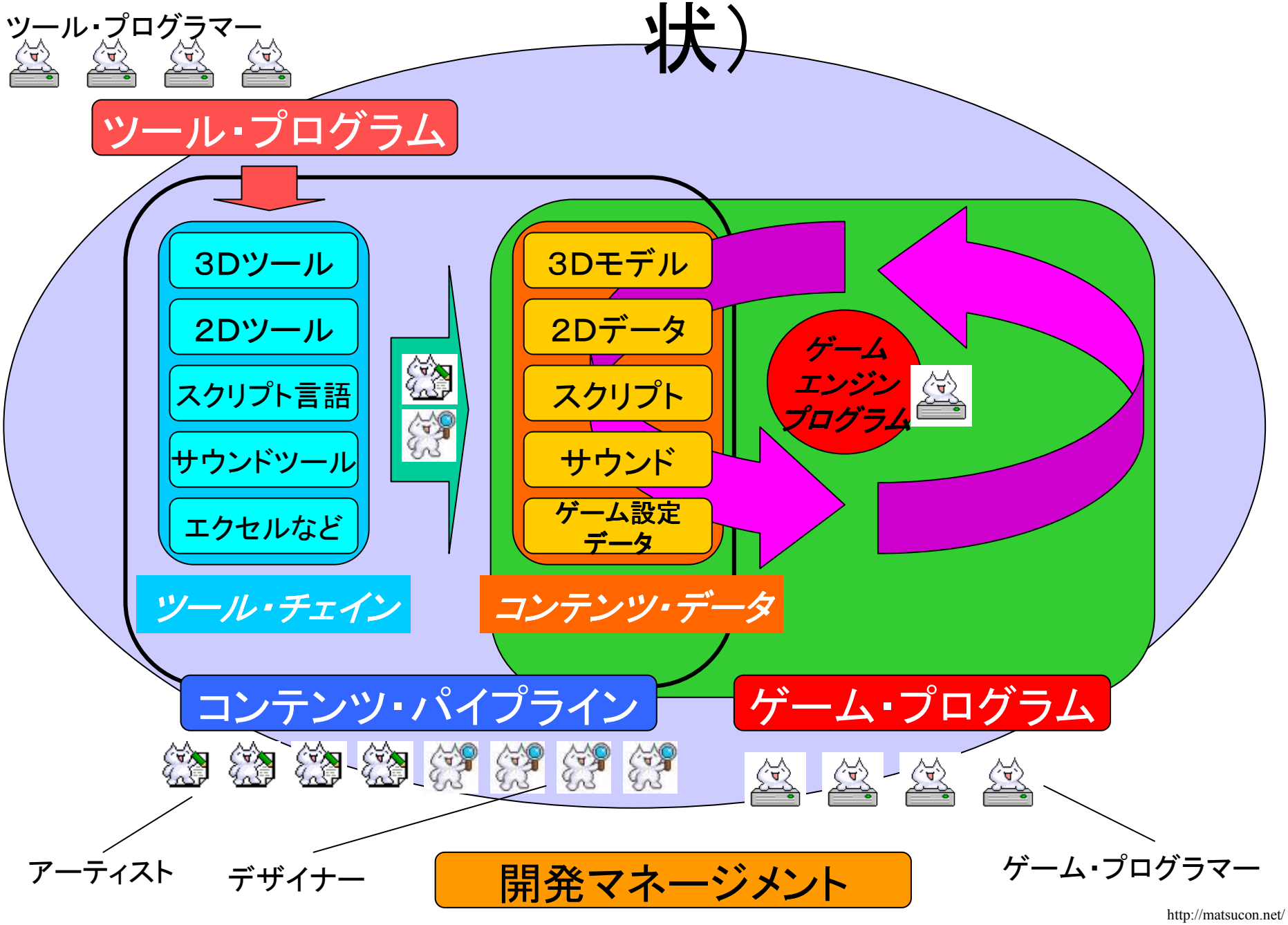
プログラマから見える風景

5～50人のプログラマがゲームプログラムからツールまで作る。
ツール専任のプログラマの増大。ミドルウェア製品も多く購入している。
そのツールを介して、デザイナーがデジタル空間にアクセスして創作する。



人と開発工程をきちんと管理しないと、予算が膨らんでたまりません！

プログラマから見える風景（現状）



まとめ

(1) ゲームデザイナー、グラフィッカー、サウンドデザイナーは、それぞれの分野で創作を行う。

(2) 創作の結果をツールを通して、ゲーム空間にコンバートする。

(3) プログラマはそういった素材を集めて、全体のゲームをリアルタイムに奏でるプログラムを作る。

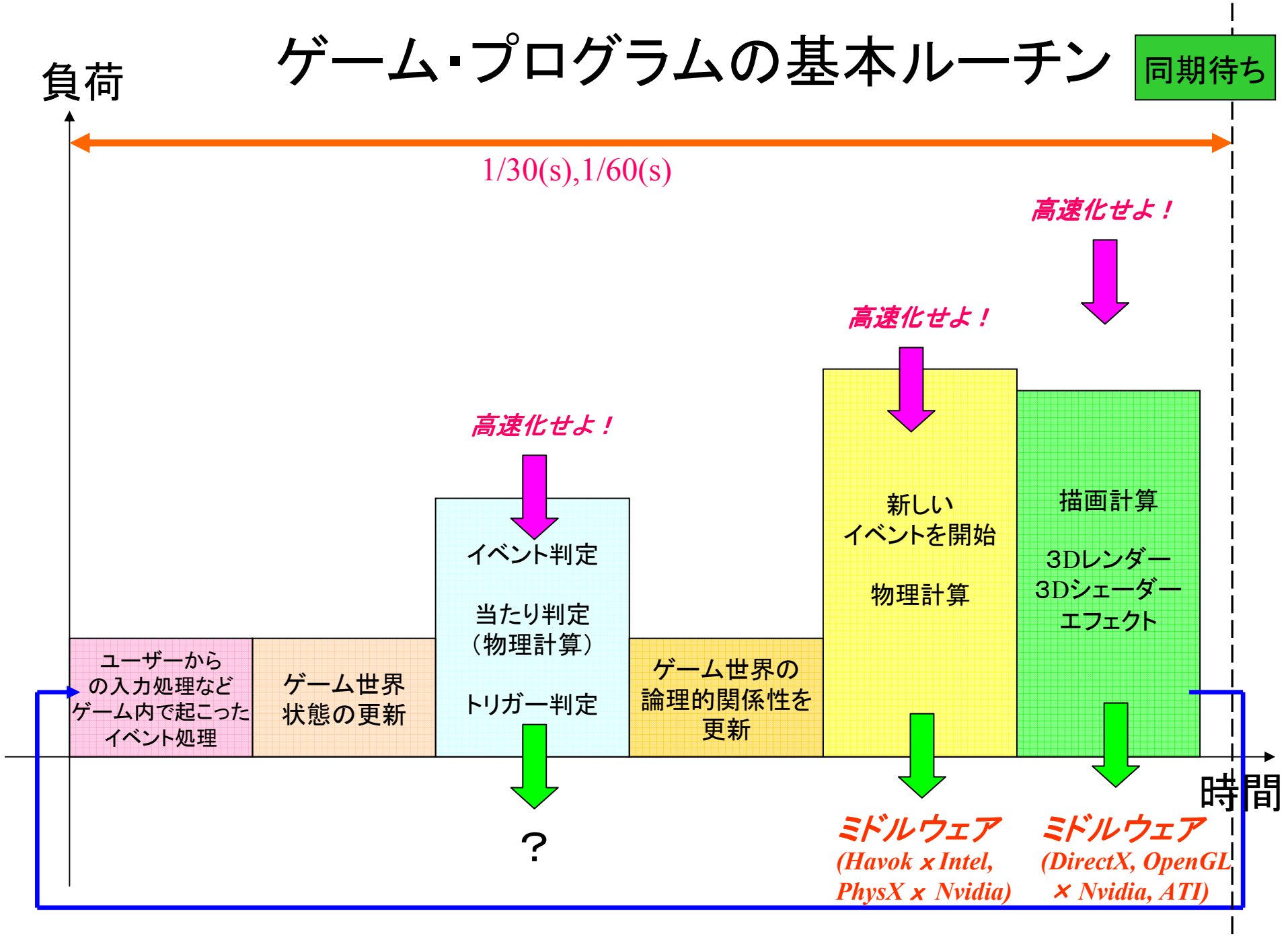
ツールは作業のモジュール化を築く土台である。
逆にツールの確立していない分野は、分業体制が
未熟である（AI分野など）。

第1章　ゲーム開発技術史　概要

(1) 仮想化

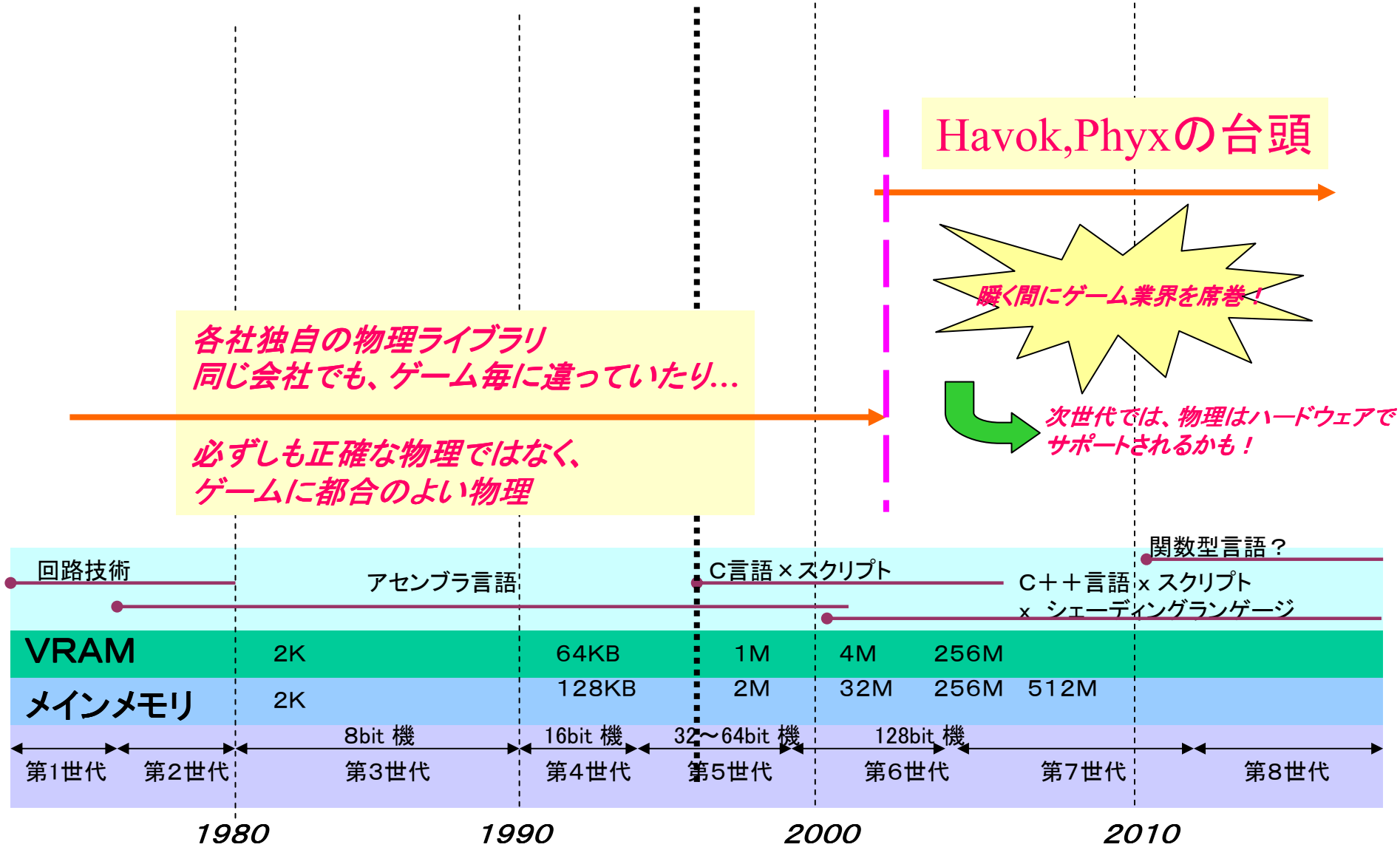
(2) モジュール化

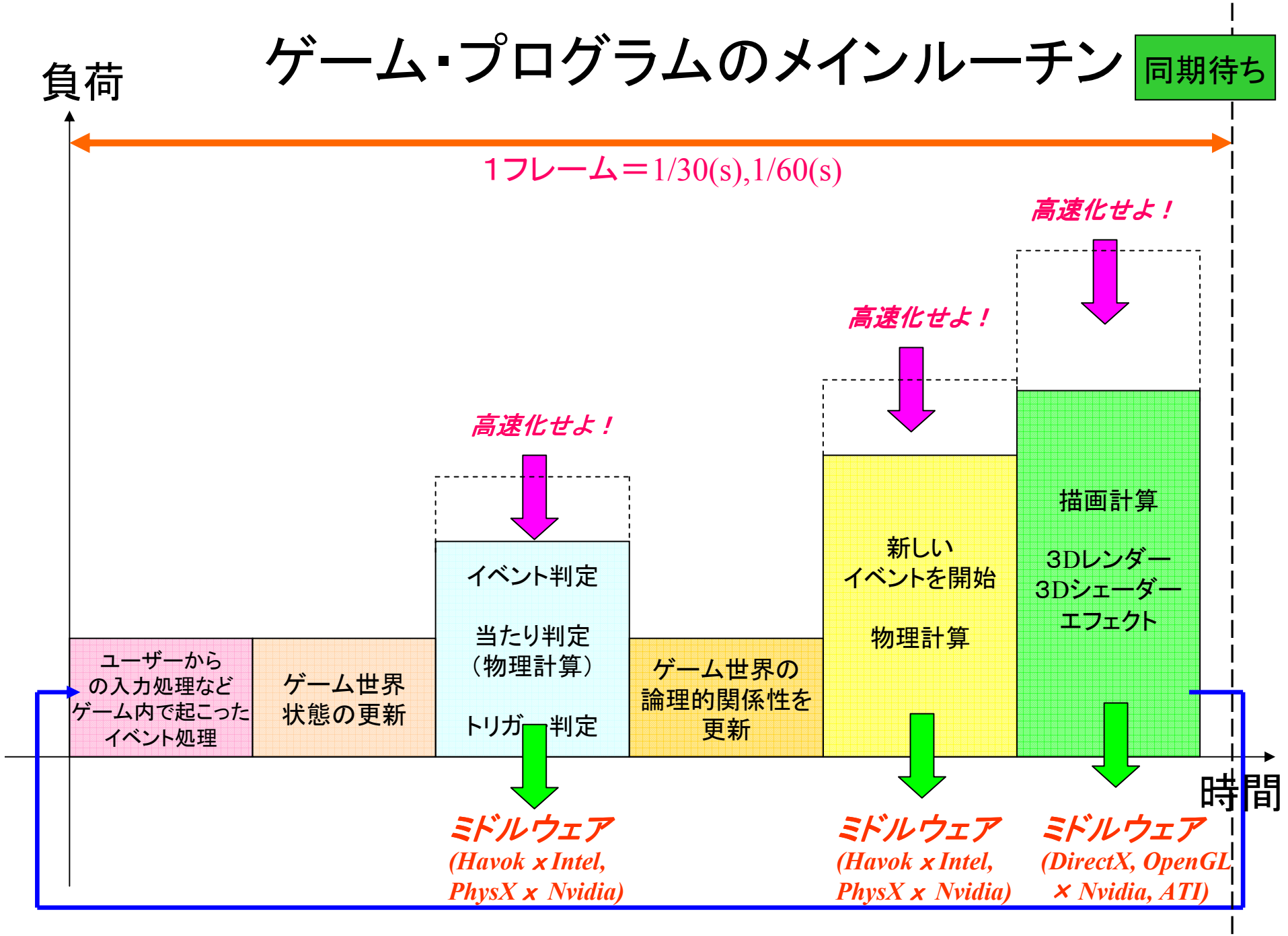
(3) 並列化


ゲーム･プログラムの基本ルーチン
同期待ち
負荷
1/30(s),1/60(s)
高速化せよ！
高速化せよ！
高速化せよ！
ユーザーから
の入力処理など
ゲーム内で起こった
イベント処理
ゲーム世界
状態の更新
イベント判定
当たり判定
（物理計算）
トリガー判定
ゲーム世界の
論理的関係性を
更新
新しい
イベントを開始
物理計算
描画計算
3Dレンダー
3Dシェーダー
エフェクト
？
ミドルウェア
(Havok x Intel,
PhysX x Nvidia)
ミドルウェア
(DirectX, OpenGL
x Nvidia, ATI)
時間

物理から見たゲーム機の歴史

それぞれの時代でゲーム機として結晶できる技術の総体としてコンシュマーゲーム機は作られて来た




ゲーム・プログラムのメインルーチン
同期待ち
負荷
1フレーム＝1/30(s),1/60(s)
高速化せよ！
高速化せよ！
高速化せよ！
ユーザーから
の入力処理など
ゲーム内で起こった
イベント処理
ゲーム世界
状態の更新
イベント判定
当たり判定
（物理計算）
トリガー判定
ゲーム世界の
論理的関係性を
更新
新しい
イベントを開始
物理計算
描画計算
3Dレンダー
3Dシェーダー
エフェクト
時間
ミドルウェア
(Havok x Intel,
PhysX x Nvidia)
ミドルウェア
(Havok x Intel,
PhysX x Nvidia)
ミドルウェア
(DirectX, OpenGL
x Nvidia, ATI)

もし、これで余裕があるとしたら、もっとたくさんのコンテンツを載せようとするのが、ゲーム業界…

さらなる高速化

極限までハードウェアを使いこなすことで、
他者より一歩でも抜きん出ることを目指す

並列化による高速化

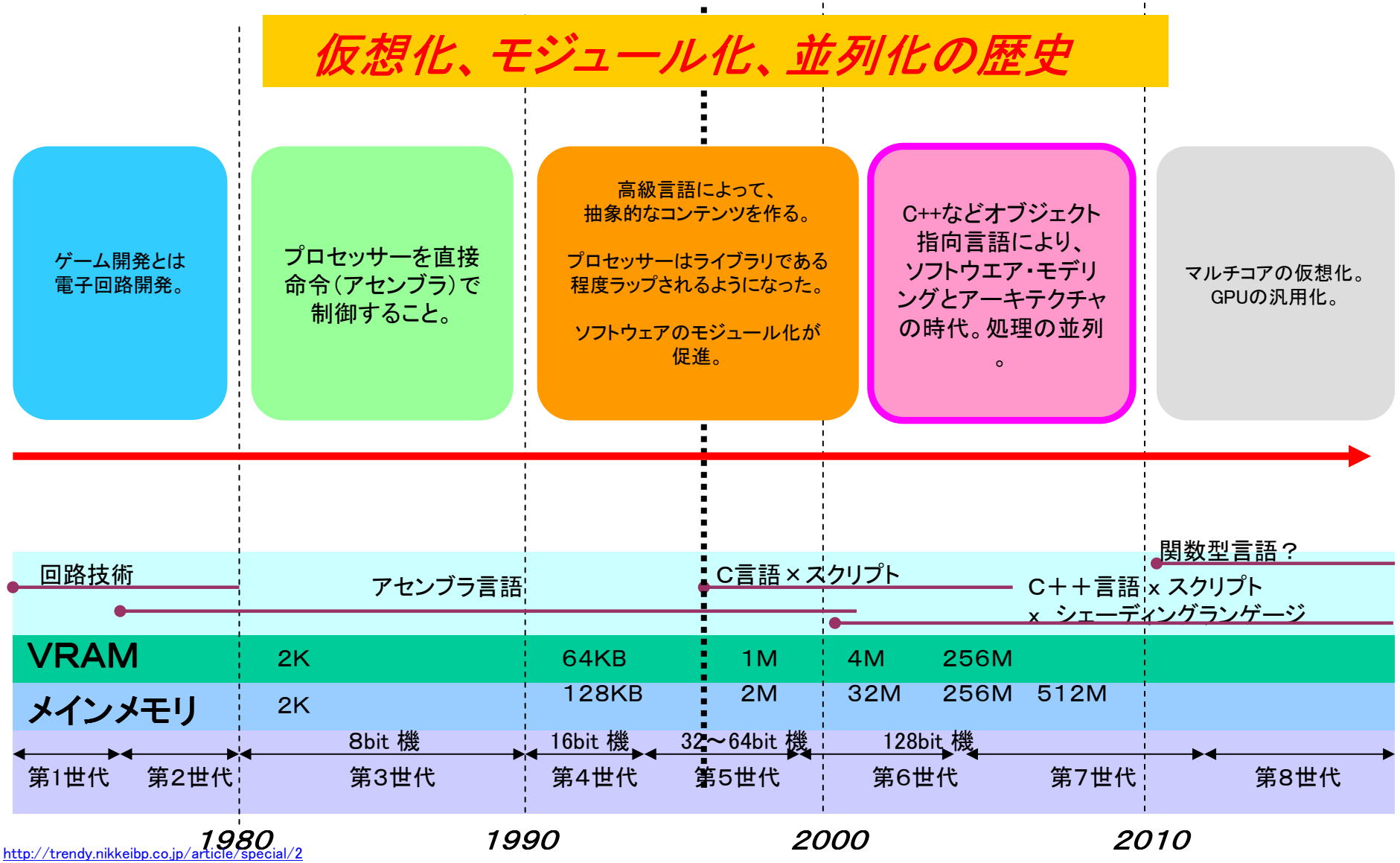
(1) Xbox360 は、３CPU６コア

(2) PS3 は、SPU x 7

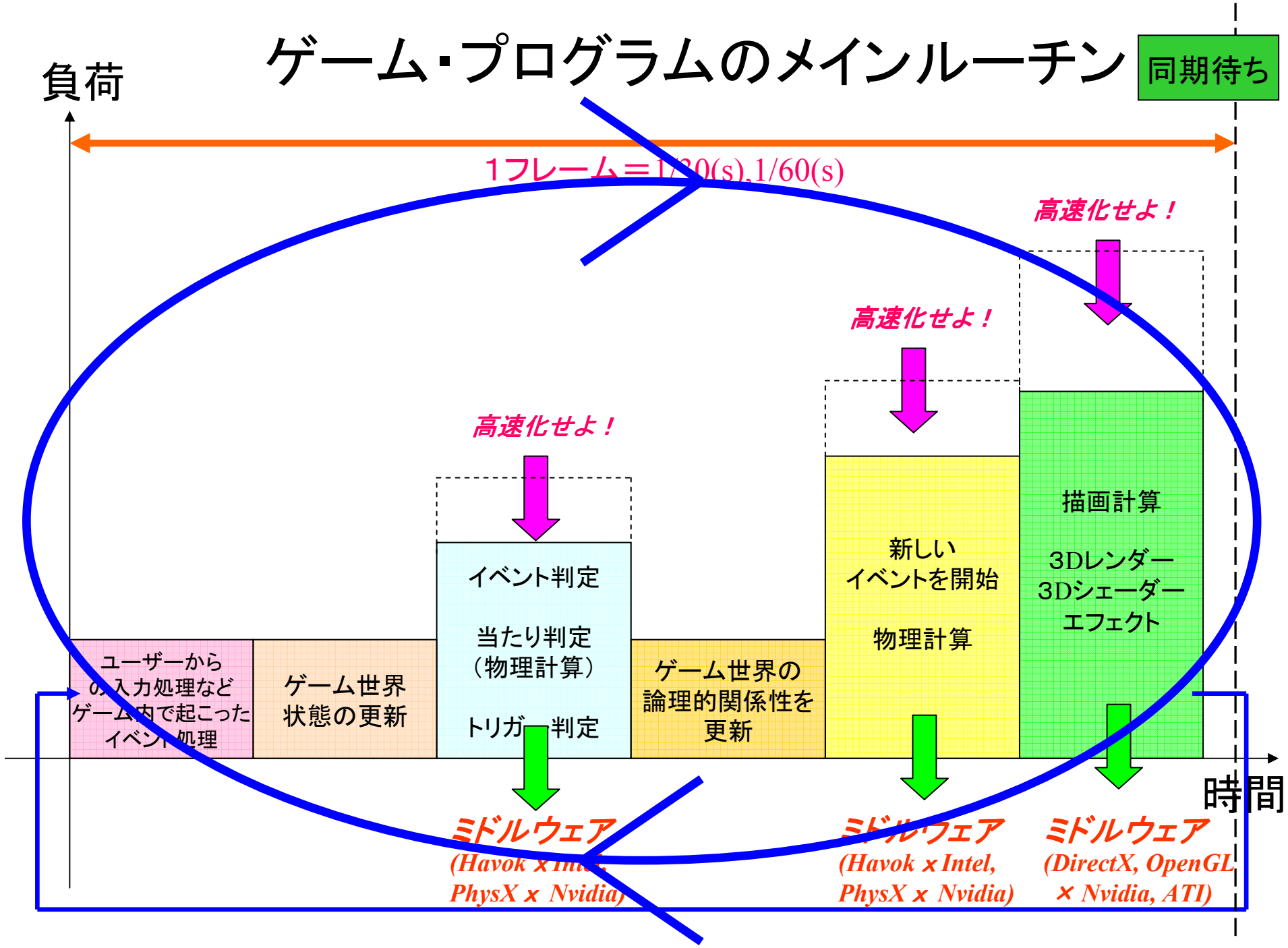
ゲーム・プログラムを如何に並列化して高速化するか？

プログラマーから見たゲーム機の歴史

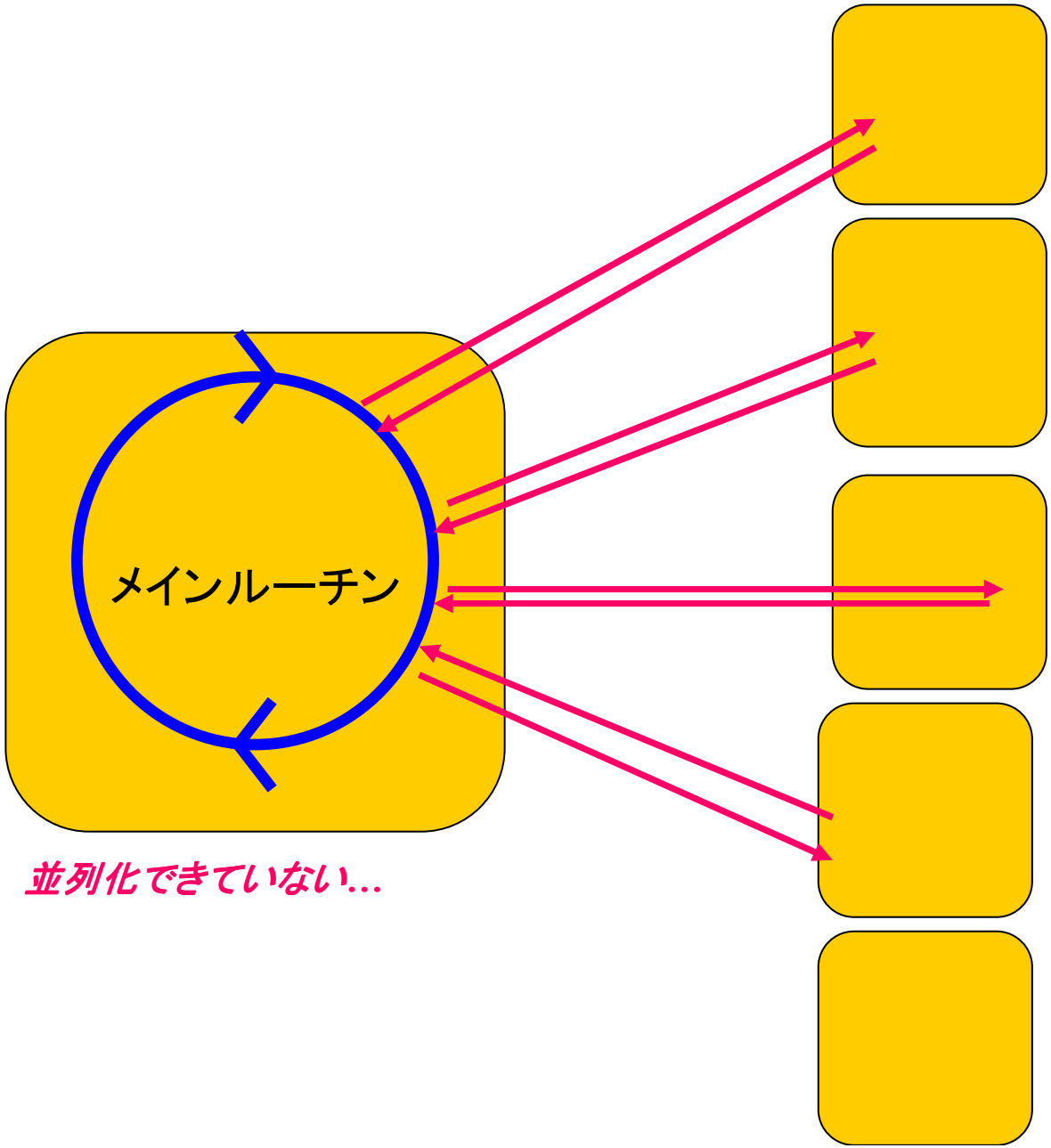
それぞれの時代でゲーム機として結晶できる技術の総体としてコンシュマーゲーム機は作られて来た



http://trendy.nikkeibp.co.jp/article/special/20080318/1008218/?P=3

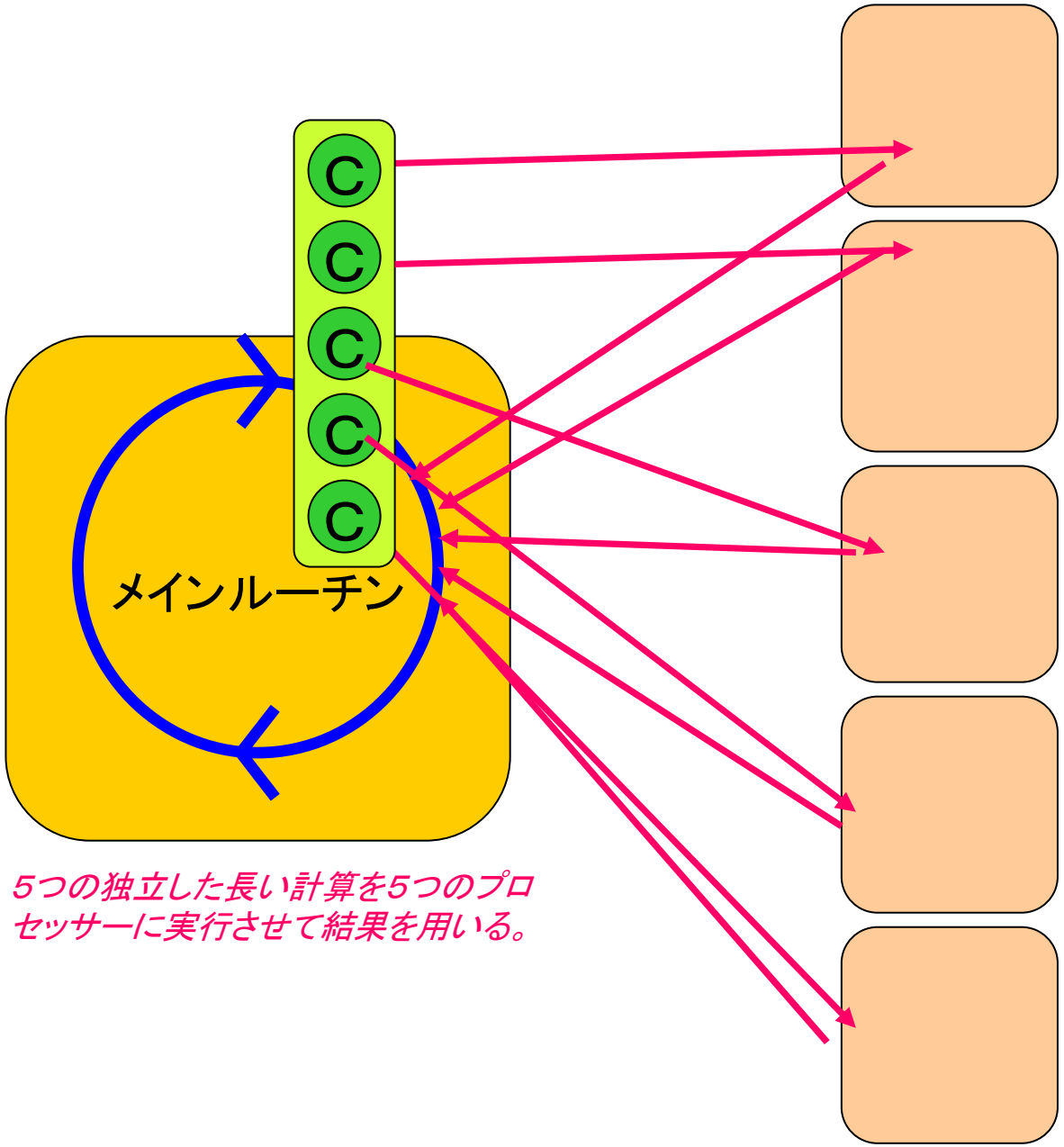

ゲーム・プログラムのメインルーチン
同期待ち
負荷
1フレーム＝1/30(s),1/60(s)
高速化せよ！
高速化せよ！
高速化せよ！
ユーザーから
の入力処理など
ゲーム内で起こった
イベント処理
ゲーム世界
状態の更新
イベント判定
当たり判定
（物理計算）
トリガー判定
ゲーム世界の
論理的関係性を
更新
新しい
イベントを開始
物理計算
描画計算
3Dレンダー
3Dシェーダー
エフェクト
時間
ミドルウェア
(Havok x Intel,
PhysX x Nvidia)
ミドルウェア
(Havok x Intel,
PhysX x Nvidia)
ミドルウェア
(DirectX, OpenGL
x Nvidia, ATI)

５つの「かまど」を使いこなす



並列化できていない…

５つの「かまど」を使いこなす



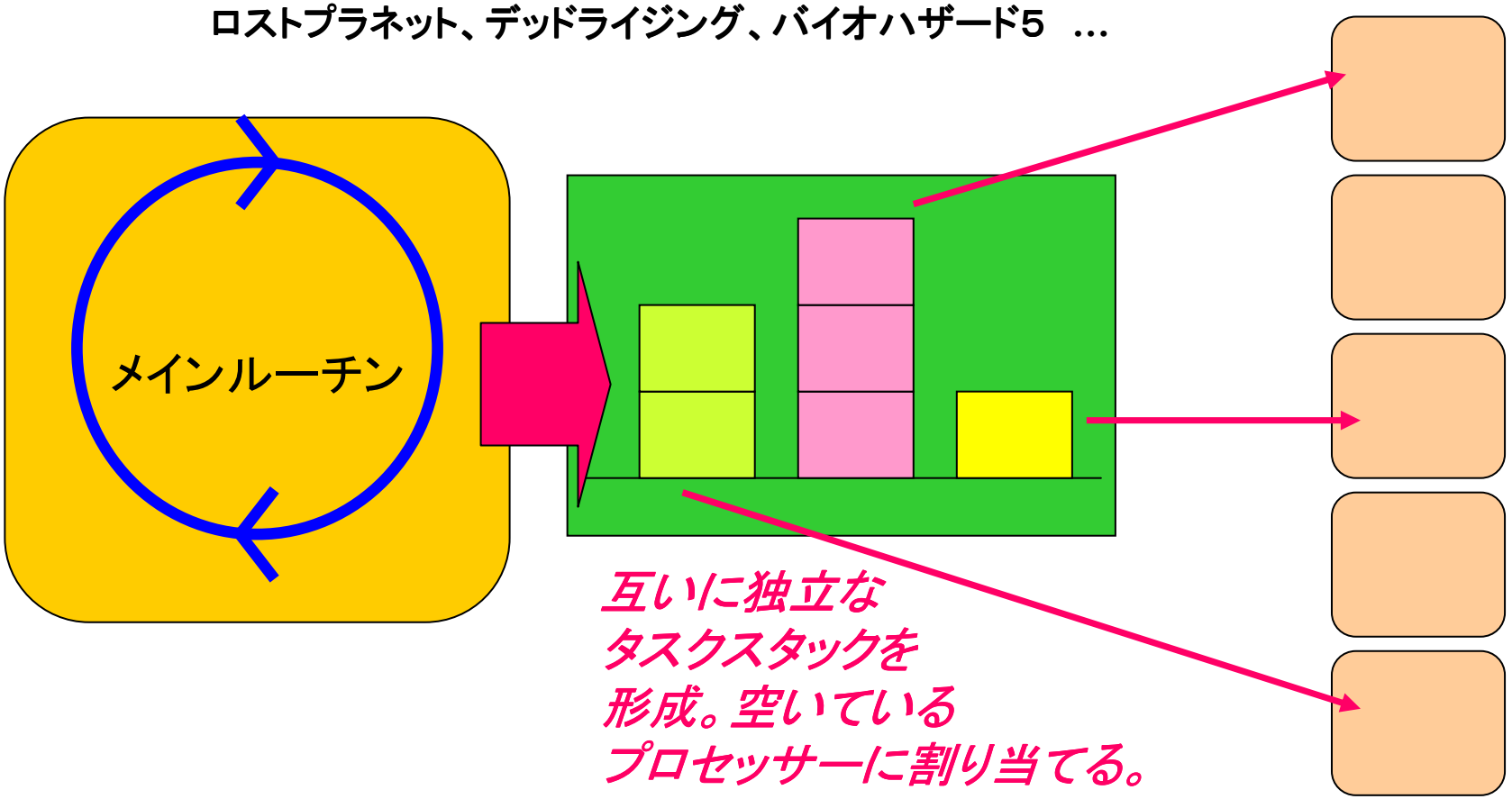
５つの独立した長い計算を５つのプロセッサーに実行させて結果を用いる。

並列化アーキテクチャー

(1) MTフレームワーク(カプコン)

http://game.watch.impress.co.jp/docs/20070131/3dlp.htm
http://www.4gamer.net/games/032/G003263/20080912001/

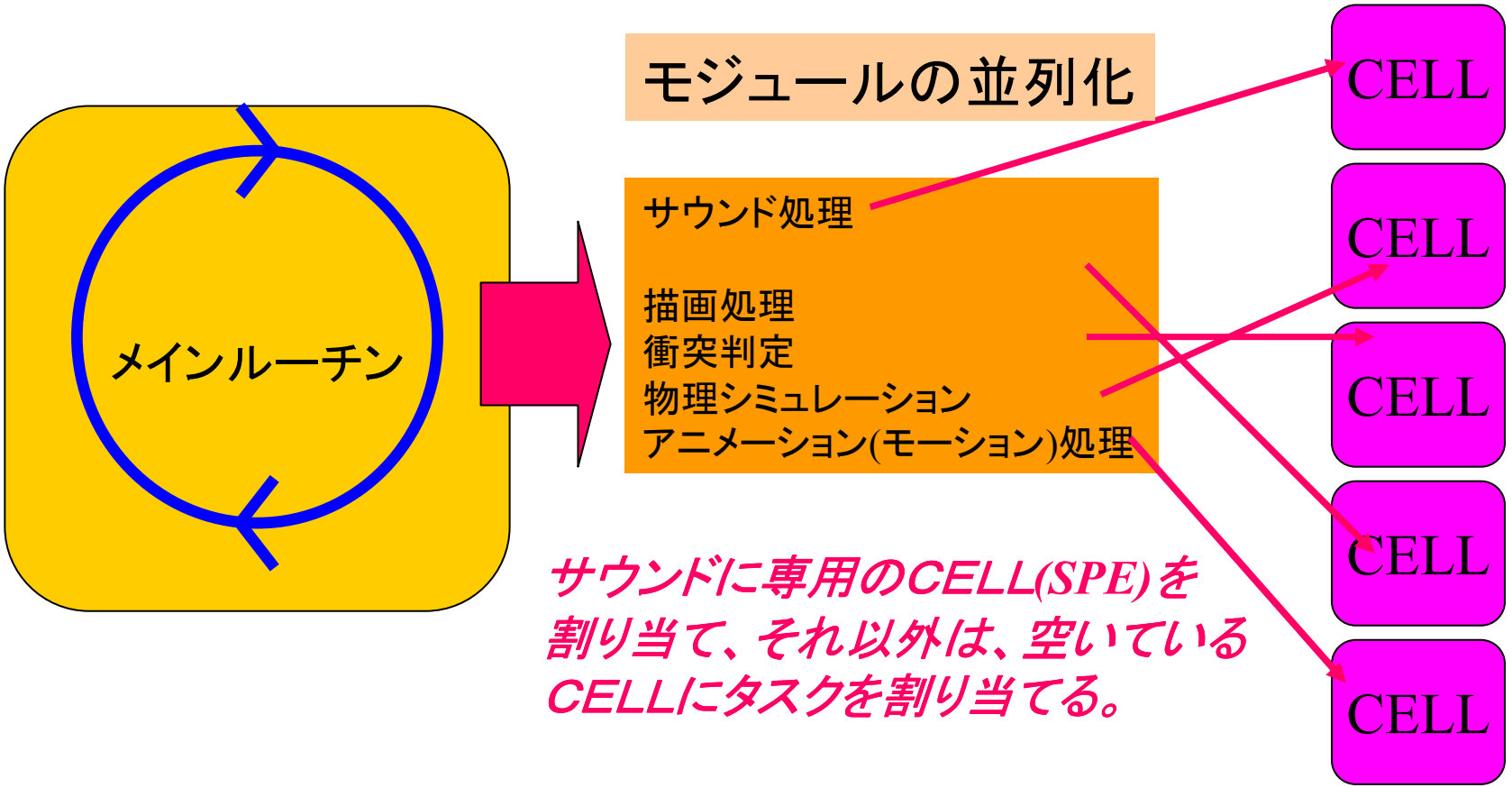
ロストプラネット、デッドライジング、バイオハザード5　…



並列化アーキテクチャー

(2) メタルギア・ソリッド4(コナミ)

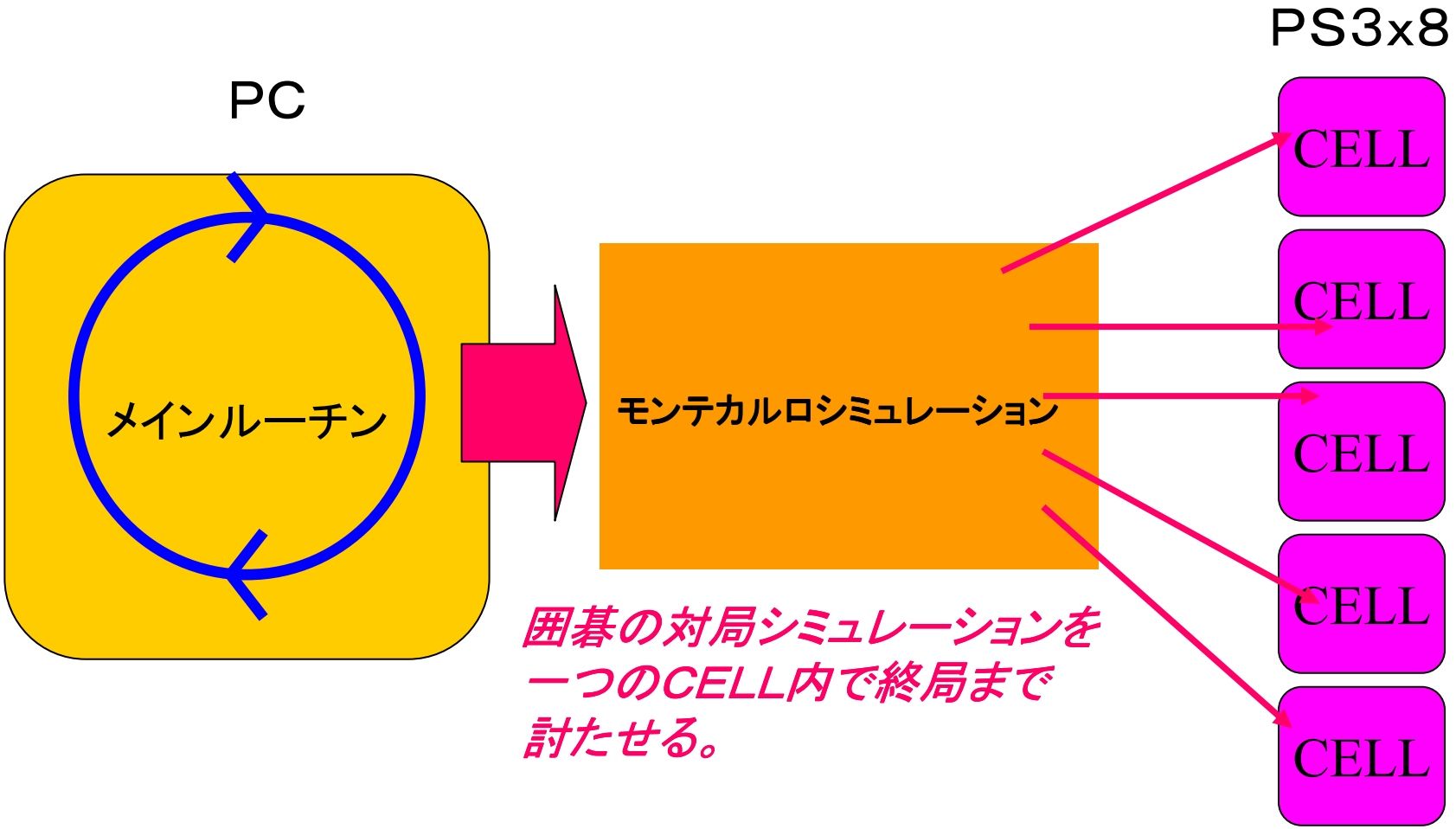
http://game.watch.impress.co.jp/docs/20081204/3dmg4.htm



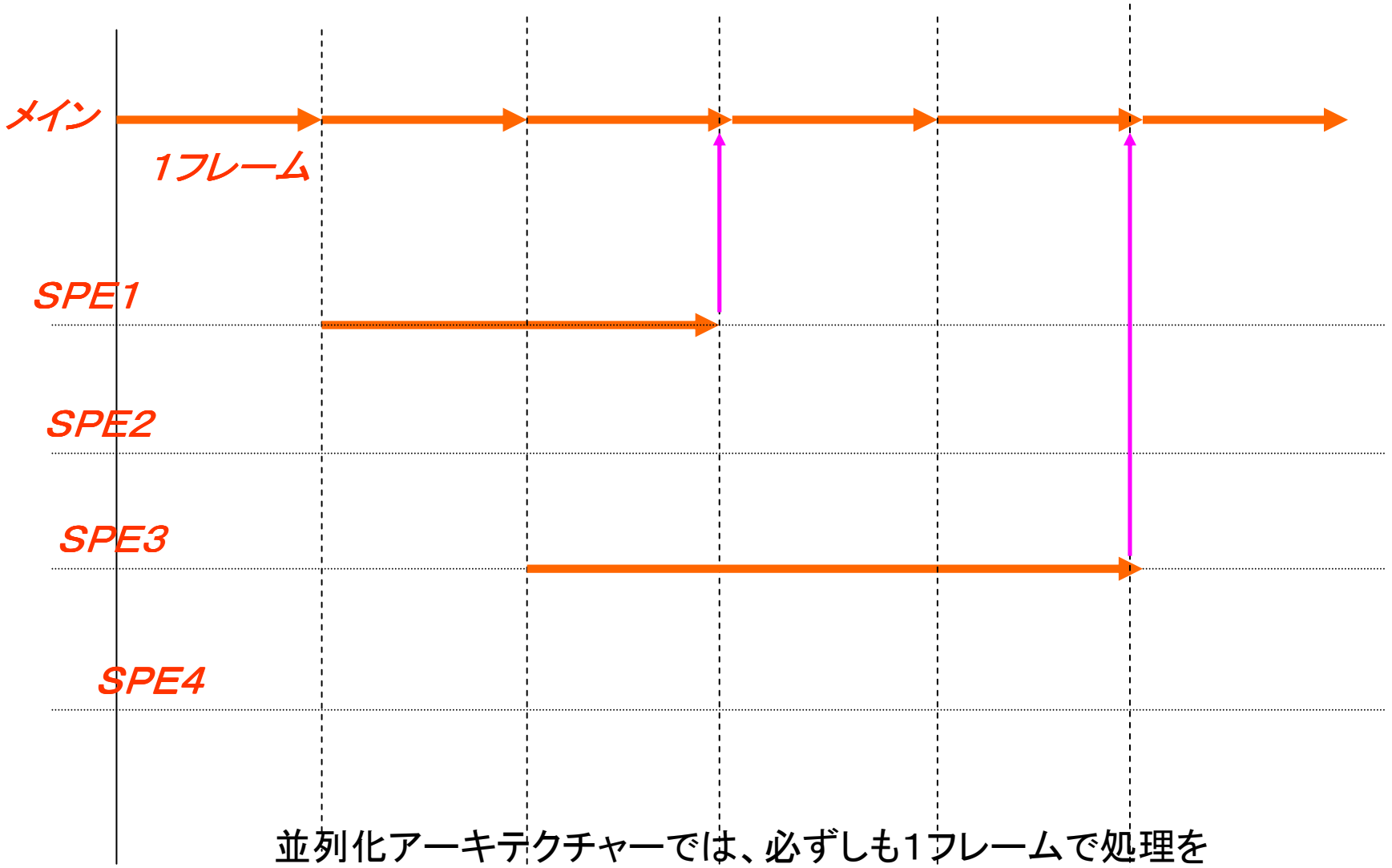
並列化アーキテクチャー

(3) 囲碁AI(不動碁、東京大学)

http://www.igda.jp/modules/xeblog/?action_xeblog_details=1&blog_id=1038



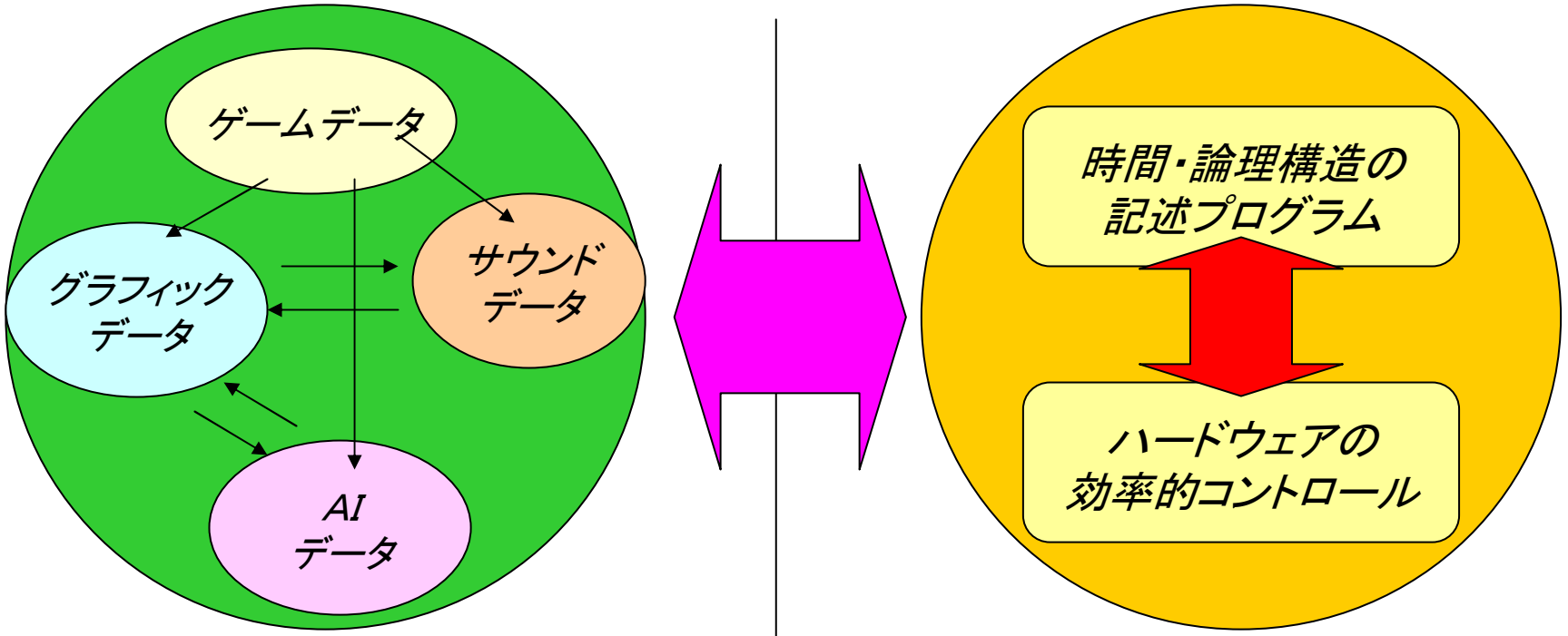
処理の並列化



並列化アーキテクチャーでは、必ずしも1フレームで処理を
終わらせる必要は無い。

第2章　ゲーム・プログラムの本質

データとプログラムの双対性



現代のゲームプログラムにおいて、
既にデータは一つの巨大な構造物

特徴 （1） ボリュームの増大
（2） データの相互関連性
（3） 高い専門性

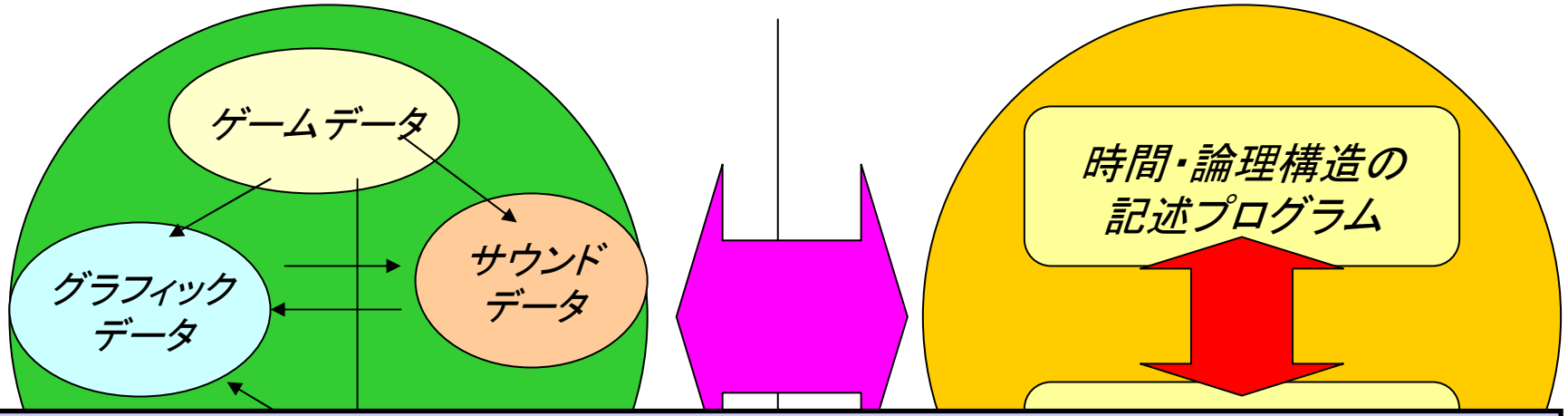
（例） 鎧 ＝ モデル＋テクスチャ＋アニメーション＋
当たった時の音＋スペック（防御力）

現代のゲームプログラムにおいて、
プログラムはハードウェアから抽象的なコンテクスト
までコントロールする巨大な車輪

特徴 （1） 大きなデータを軽く動かす馬力
（2） 仮想化、モジュール化、並列性
（3） 高いコーディング力と
高度なアーキテクチャー設計の重要性

（例） 並列化アーキテクチャー

データとプログラムの双対性



よく完備されたデータはアルゴリズムを簡略化し、
よいアルゴリズムは高い対称性を持ったデータ構造を求める。

逆に、データの不備は計算負荷を増大させ、
つまらないアルゴリズムは低次元のデータ構造しか産まない。

特徴 （1） ボリュームの増大
（2） データの相互関連性
（3） 高い専門性

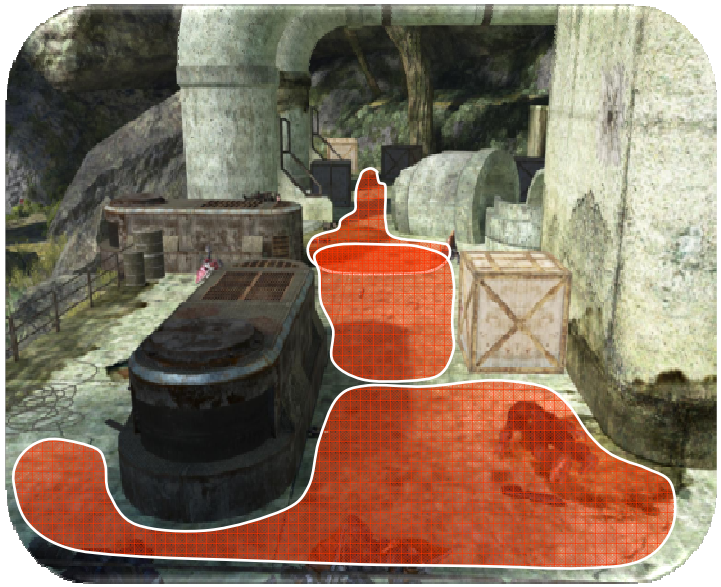
（例） 鎧 ＝ モデル＋テクスチャ＋アニメーション＋
当たった時の音＋スペック（防御力）

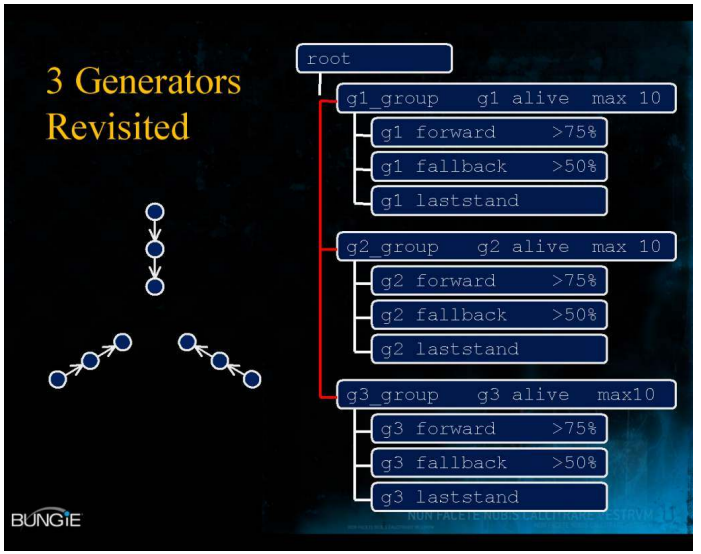
特徴 （1） 大きなデータを軽く動かす馬力
（2） 仮想化、モジュール化、並列性
（3） 高いコーディング力と
高度なアーキテクチャー設計の重要性

（例） 並列化アーキテクチャー

（例）キャラクターAI

Damian Isla, "Building a Better Battle: HALO 3 AI Objectives" (GDC2008)
http://www.bungie.net/inside/publications.aspx



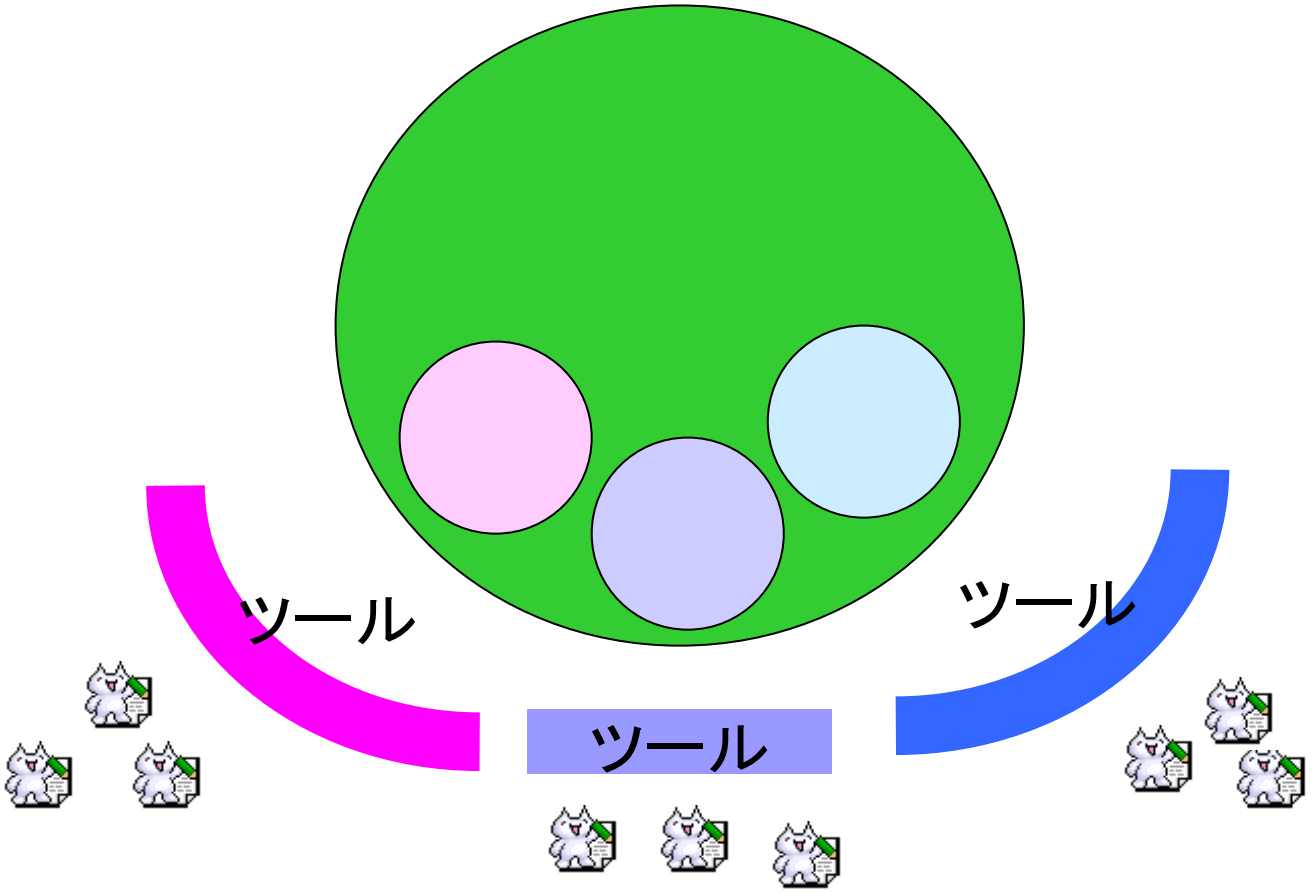


よく準備されたAIにための地形データ
（前衛、中間、後退位置のデータを用意、Halo3）

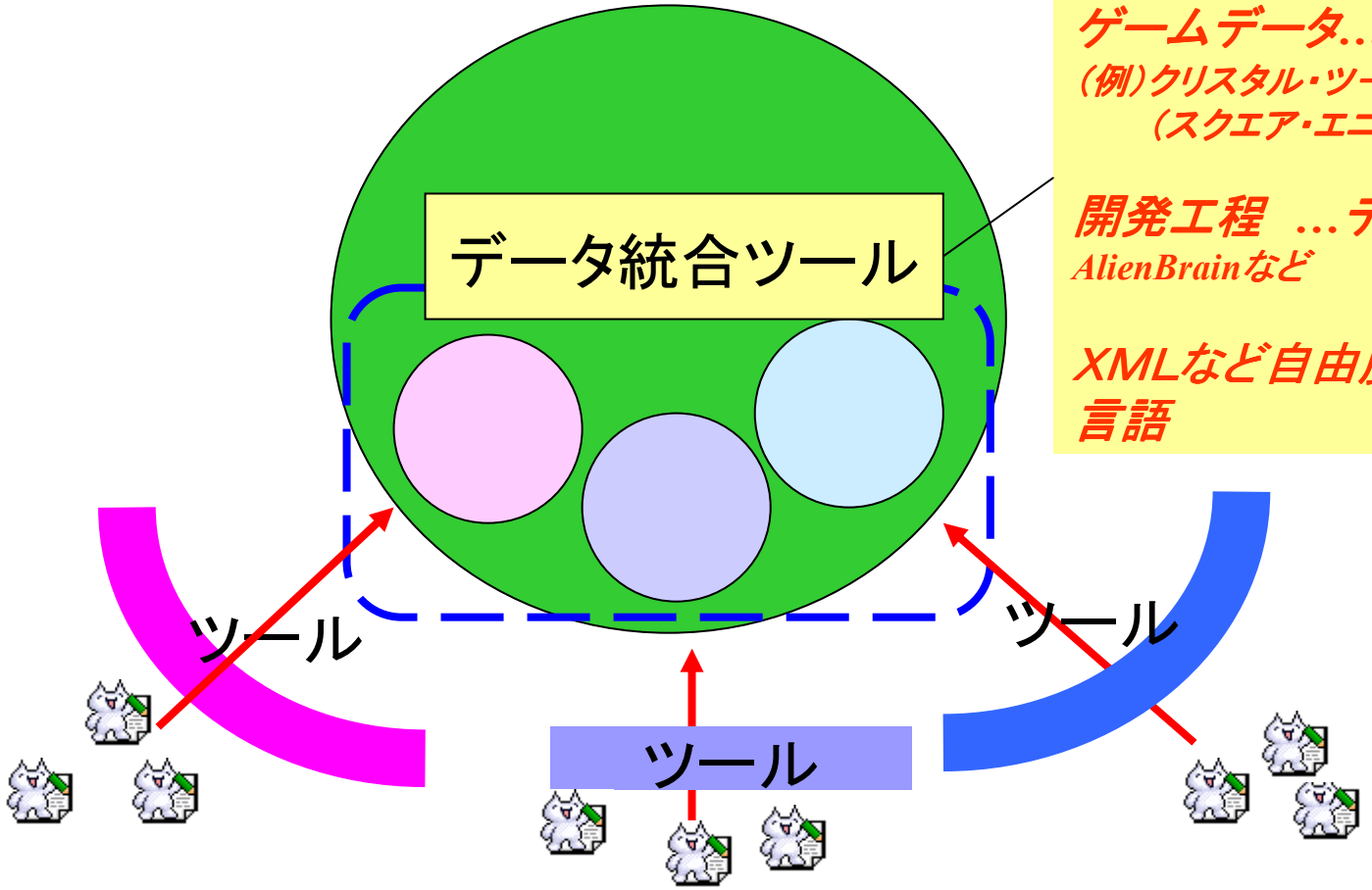
チーム線略のアルゴリズム。
プレイヤーの行動に応じて、戦略的な
位置取りをして翻弄する。(Halo3)

AIのために事前に地形を解析したデータを作っておくと、
シンプルなアルゴリズムで十分に高度な知能を実現できる。
（世界表現の考え方）

高度なデータの生成には？



高度なデータの生成には？



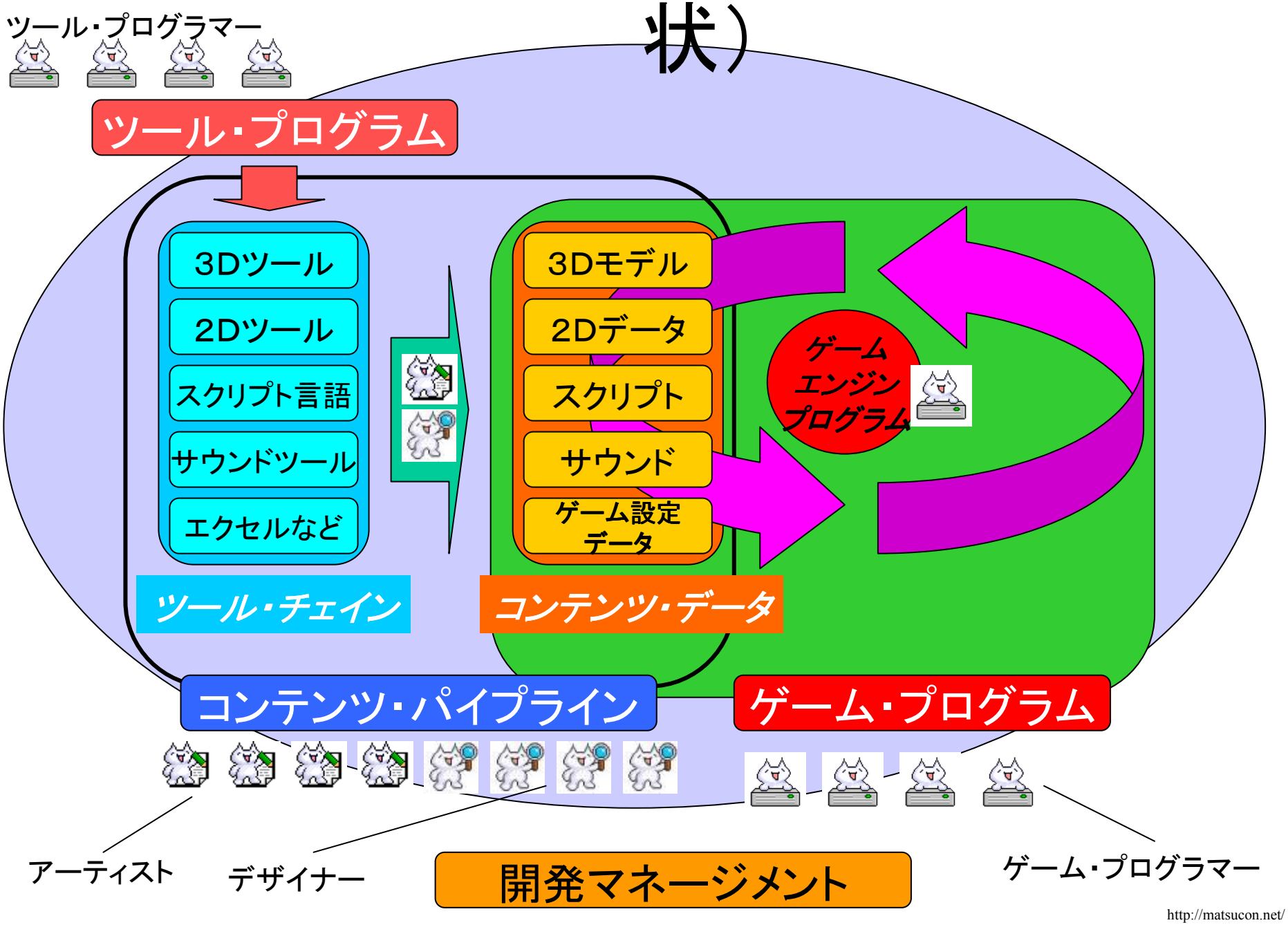
ゲームデータ…自社ツール
（例）クリスタル・ツールズ
（スクエア・エニックス）

開発工程　…データ管理ツール
AlienBrainなど

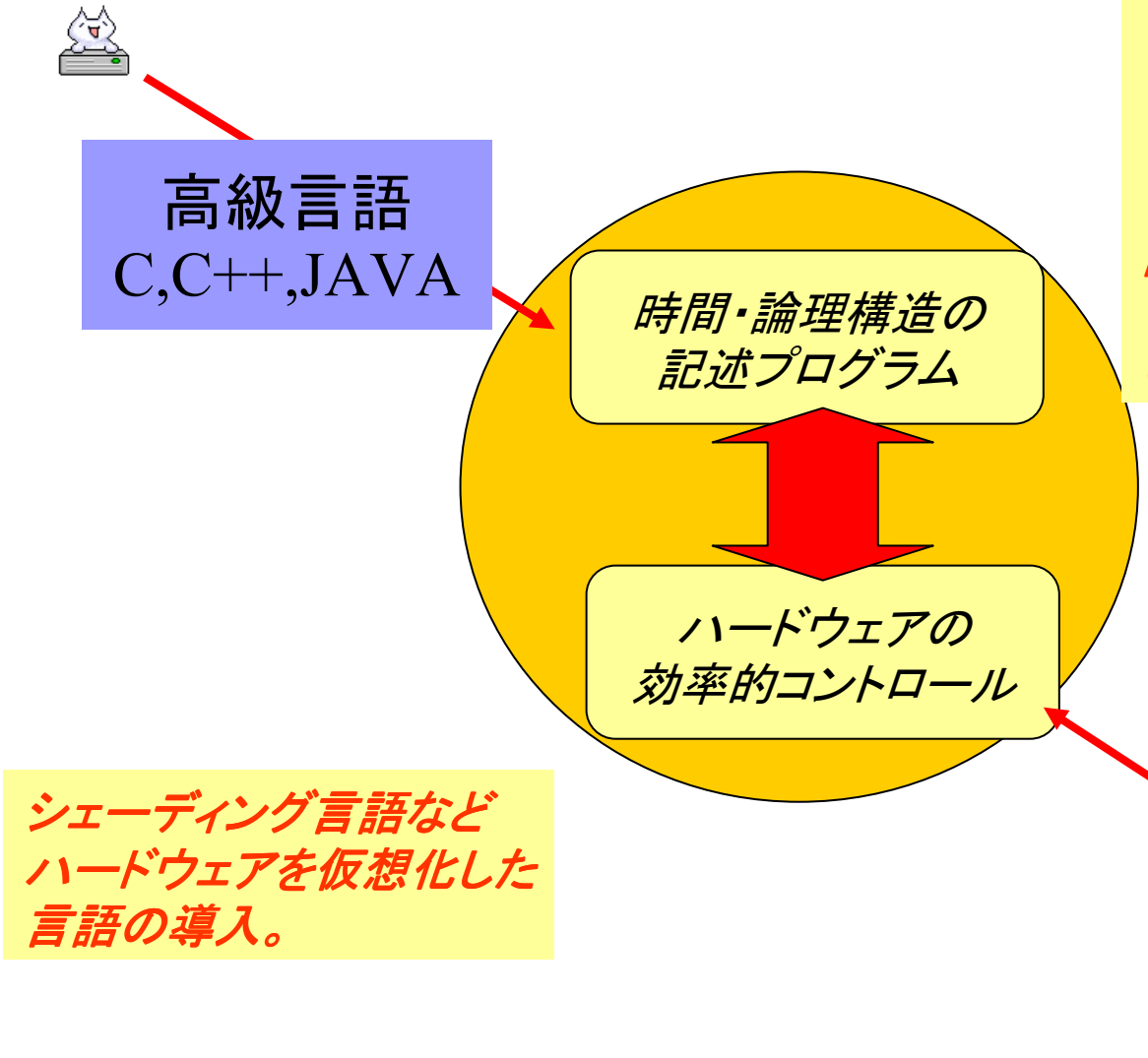
XMLなど自由度の高いデータ記述言語

人材・工程管理・ツールチェイン

プログラマから見える風景（現状）



高度なプログラムの生成には？



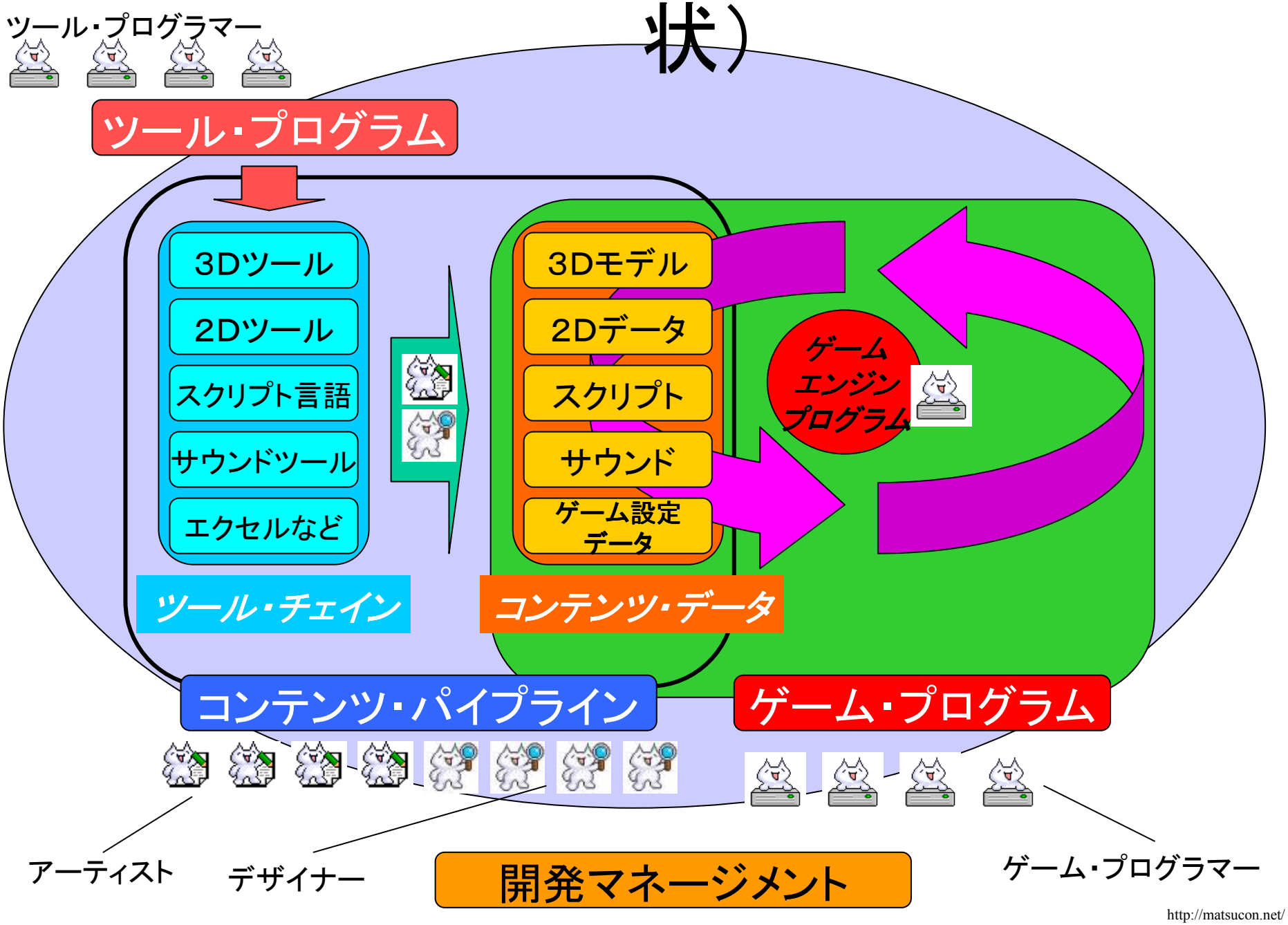
特にオブジェクト指向は高度に抽象的な概念のプログラミングを可能にした。（クラスの導入）

UMLなどの
ソフトウェア・アーキテクチャの
設計言語。仕様記述言語。

CMMIなど工程管理技術。

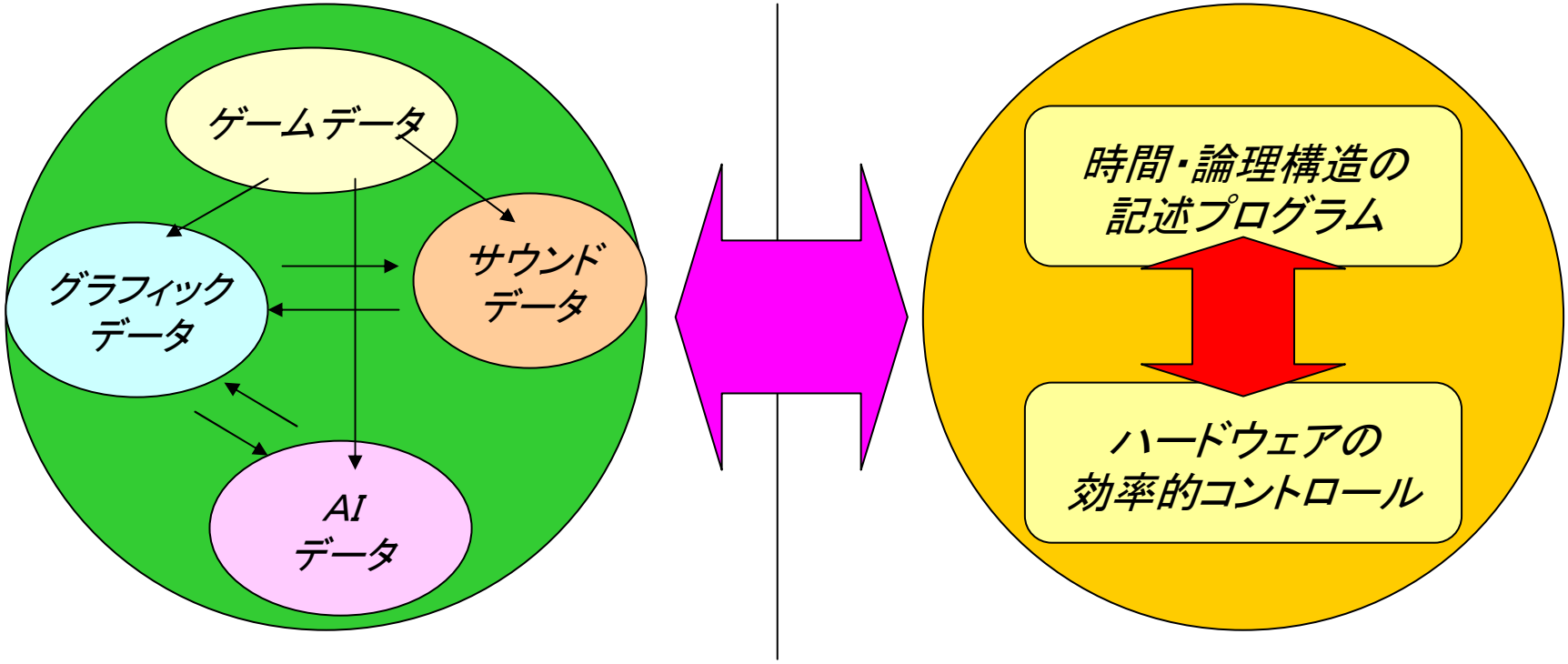
シェーディング言語など
ハードウェアを仮想化した
言語の導入。

プログラマから見える風景（現状）



データとプログラムの双対性

高度に抽象的なコンテンツを回しながら



効率的にハードウェアを活用する。

第3章　ゲーム開発技術の現象学

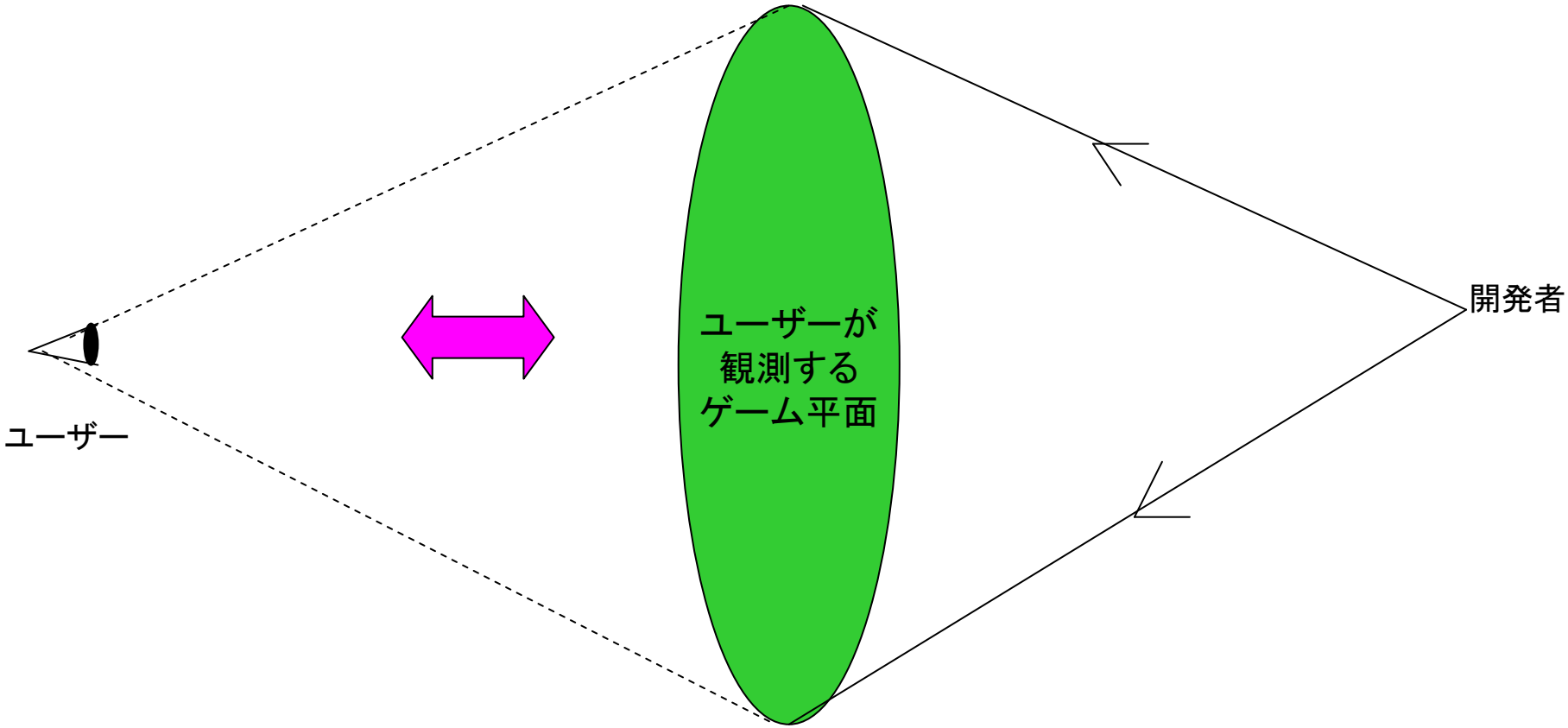
ゲーム開発技術の現象学

(1) ゲーム開発技術は、ユーザー・エクスペリンスにどのように貢献するか？ (効果)

(2) ゲーム開発技術は、どのような経路をたどってゲーム開発へ持ち込まれるか？（経緯）

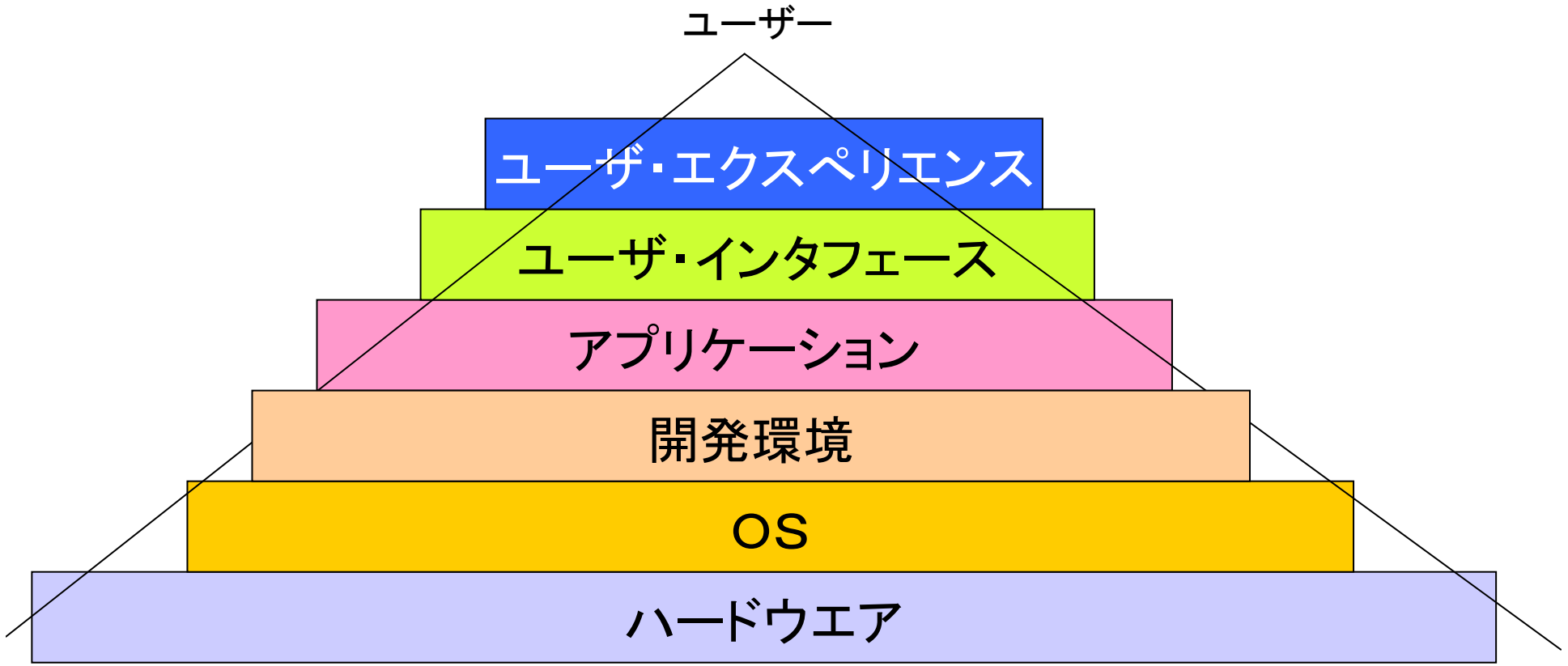
ユーザー・エクスペリエンスと技術

ゲーム＝エンターテインメントであるとすれば、ゲーム内の技術は、
ユーザー・エクスペリエンスに貢献しなければならない。



ユーザー・エクスペリエンスの形成

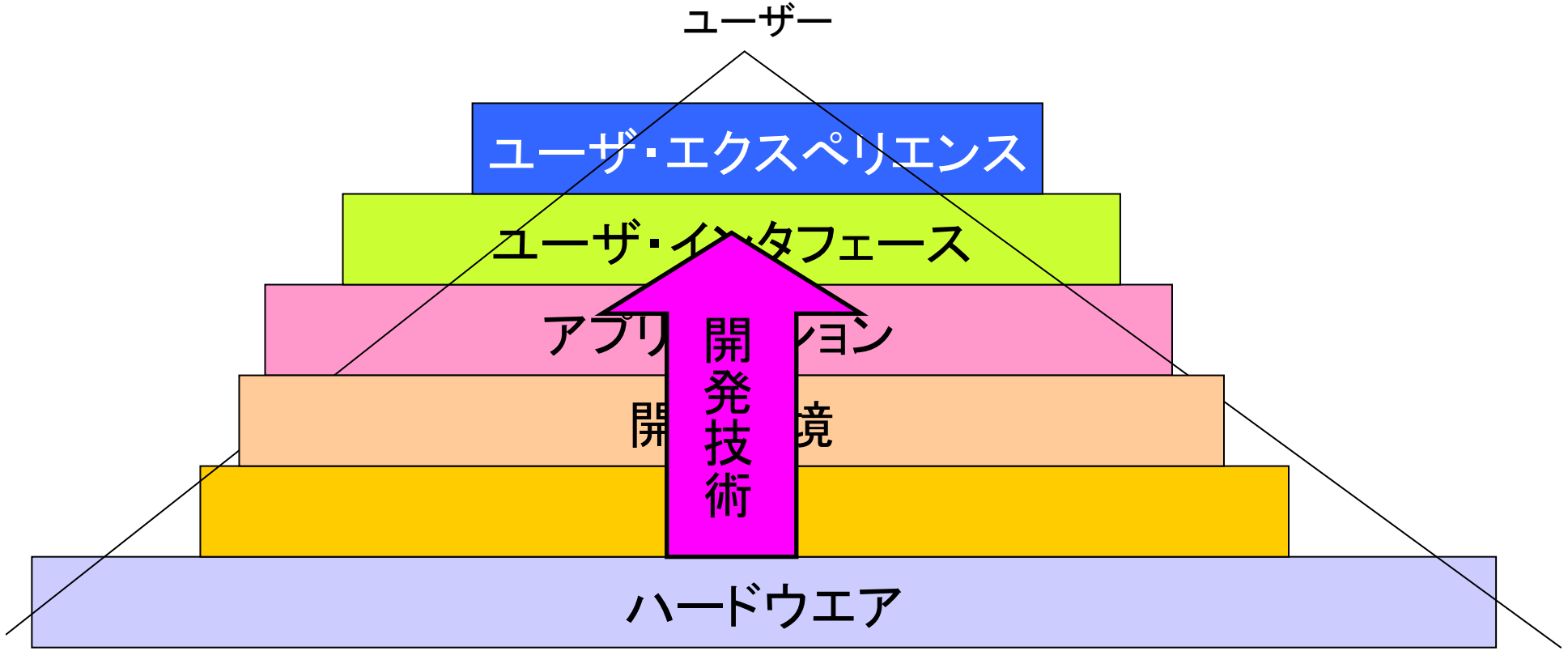
基礎技術から始まってユーザー・エクスペリエンスを形成するまで



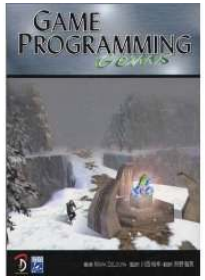
ゲーム開発技術は、
（ゲーム的）ユーザー・エクスペリエンス

ユーザー・エクスペリエンスの形成

基礎技術から始まってユーザー・エクスペリエンスを形成するまで

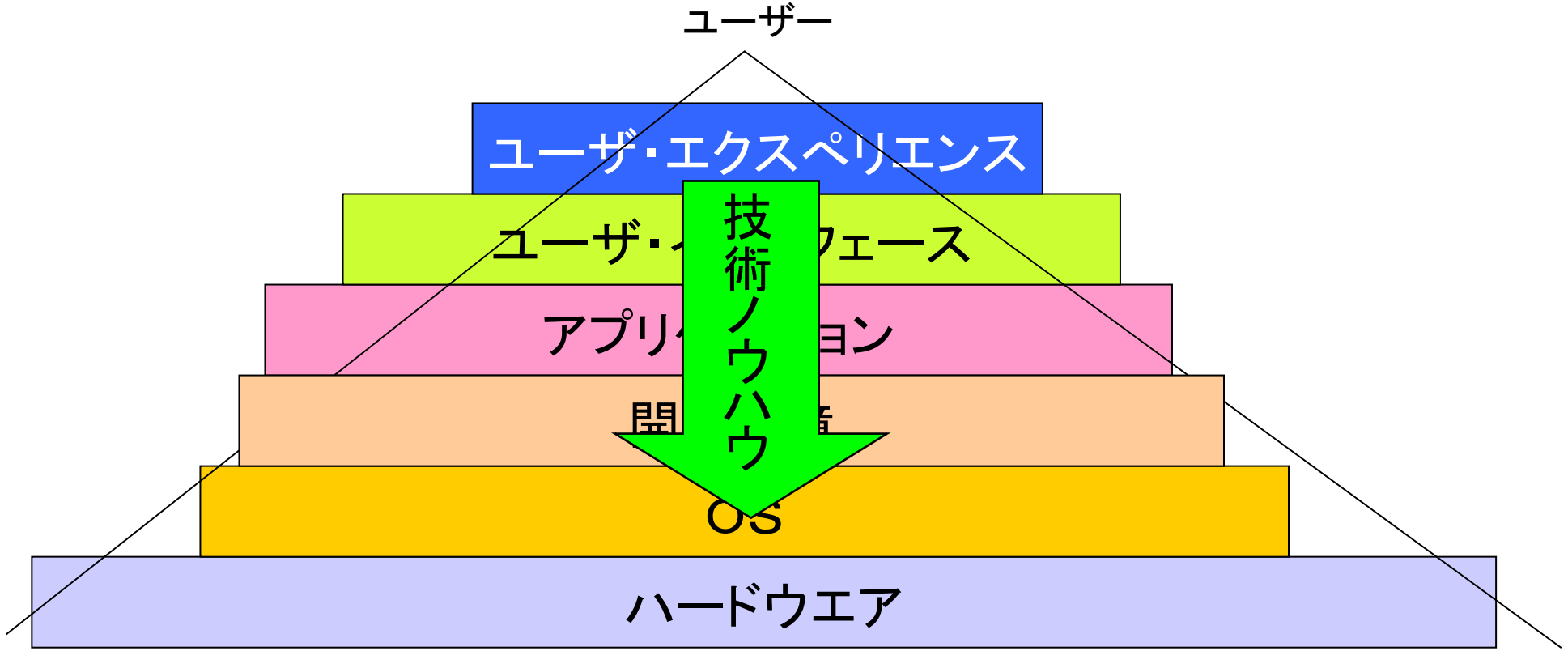


ゲーム開発技術は、(ゲーム的)ユーザー・エクスペリエンスを産み出す源泉・過程として記述されなければならない。



ユーザー・エクスペリエンスの形成

基礎技術から始まってユーザー・エクスペリエンスを形成するまで

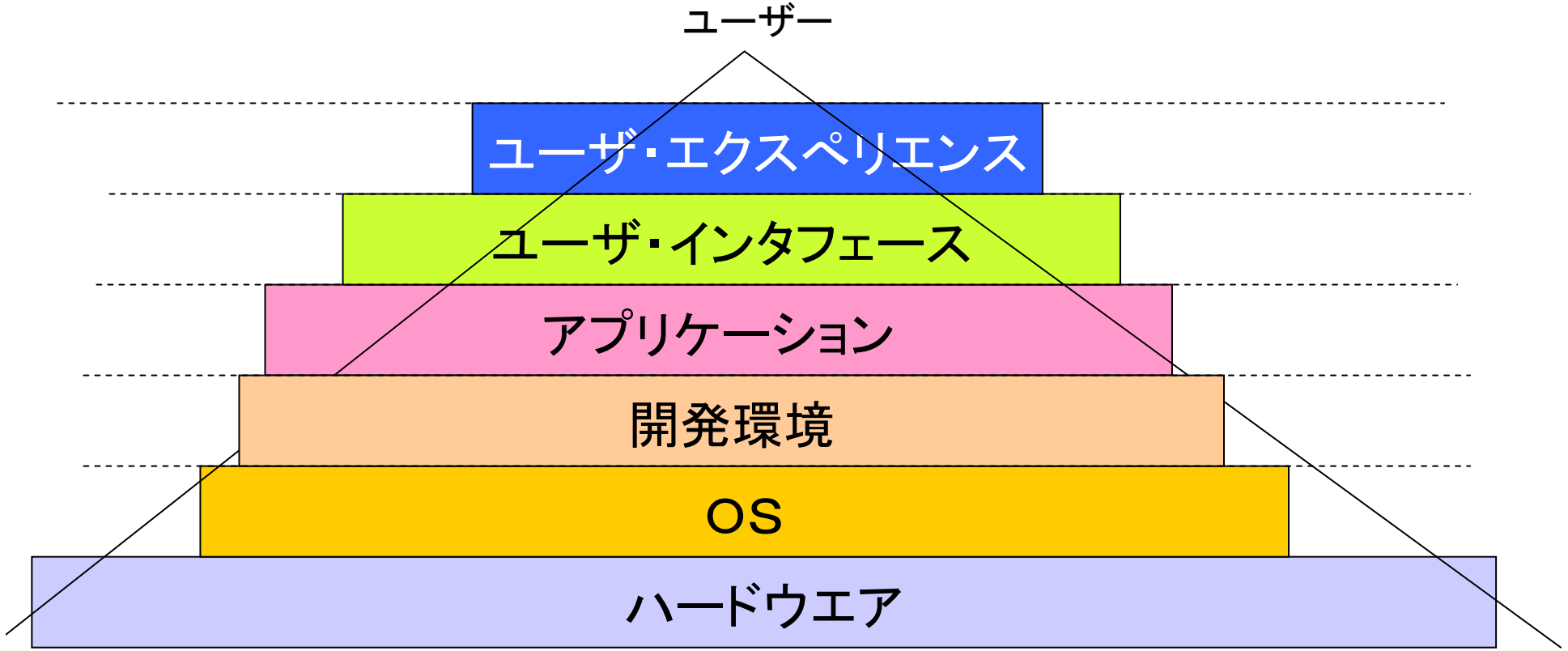


しかし一　、(ゲーム的)ユーザー・エクスペリエンスが発見されると、
それが実現できれば、どのような技術であっても構わない。

(例)3Dが大　なのではない。3D的ゲーム体　が大　。

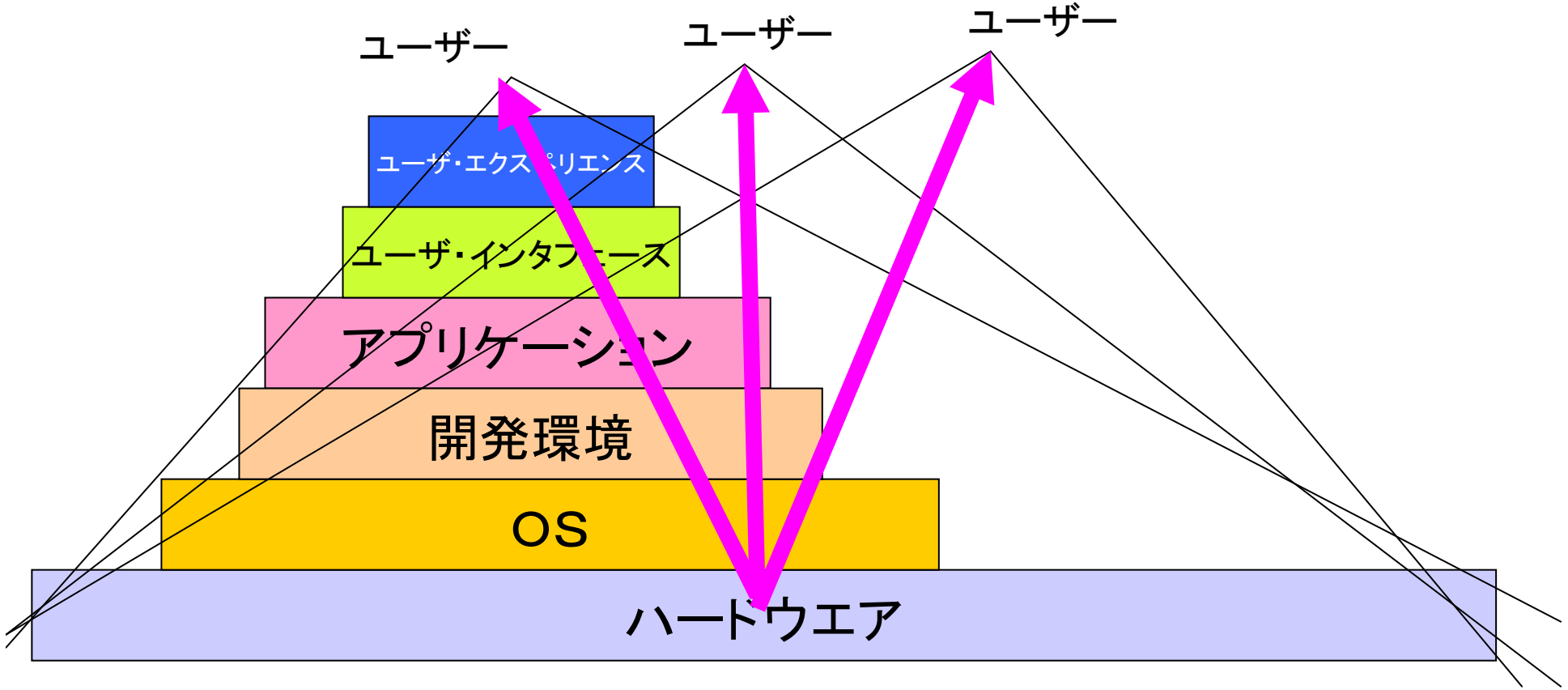
ユーザー・エクスペリエンスの形成

1つ1つの　を　　の　念に　　することで新しいゲームが産まれる



ユーザー・エクスペリエンスの形成

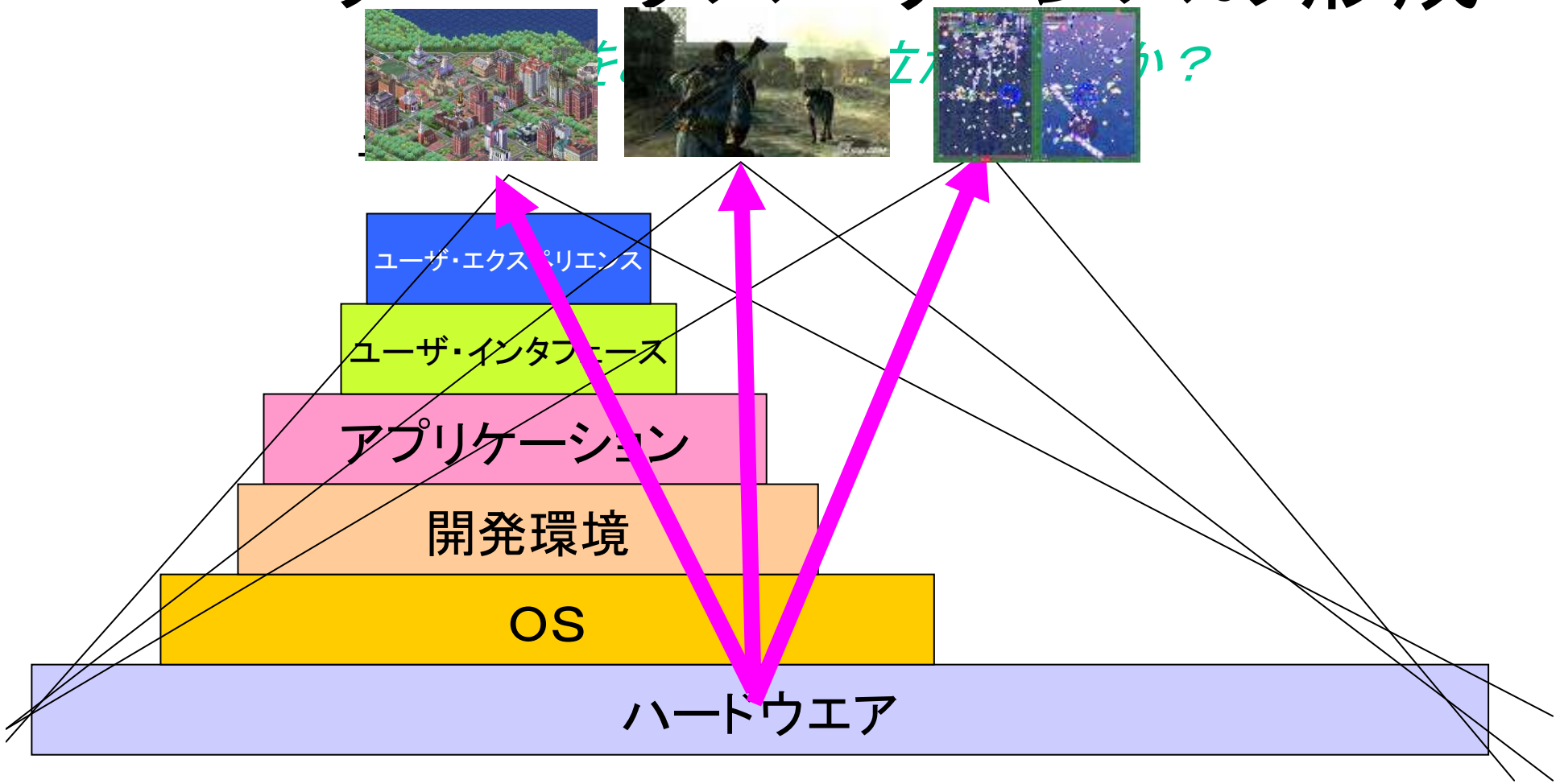
ユーザーをどの　　に立たせたいか？



ユーザー・エクスペリエンスを形成する方　　＝　ゲーム技術

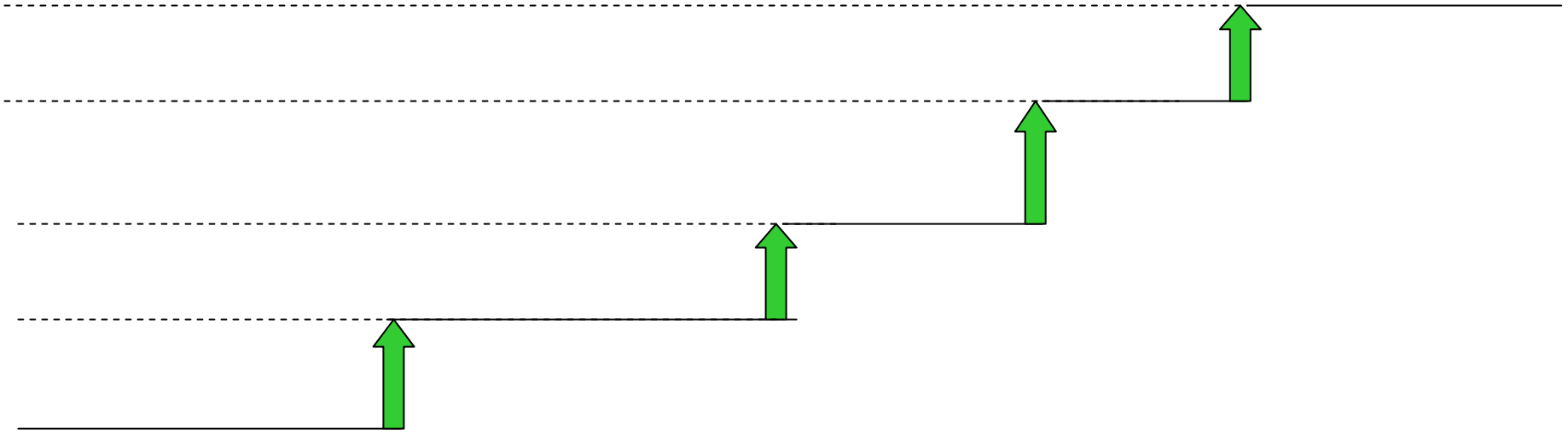
ゲームが　　とき＝ユーザー・エクスペリエンスが　定化されるとき

ユーザー・エクスペリエンスの形成



ユーザー・エクスペリエンスを形成する方　　＝　ゲーム技術

ゲーム開発技術の社会学



ゲーム会社は、長い目で見れば、技術力が上がって行く？
どはどのようにして、新しい技術が導入されて行くのか？

ゲーム開発技術の社会学

技術導入の経緯

(1) ５年に一度の新ハードウェア
(2) ミドルウェアの導入
(3) 自社開発技術
(4) 新しい人材
(5)

各開発会社の技術の方向とばらつきは、
どのような経緯で発生しているのか？