ANDROID

第４回Androidセミナーin大阪

OpenGLとスレッド処理とライフサイクル

エモーションプラス　原　秀樹

内容

- 自己紹介
- ケータイで3DCGを！
- OpenGLとは？
- 3DCG処理とOpenGL
- OpenGLとGPU
- マルチスレッド処理とは？
- ライフサイクルとは？
- マルチスレッドによる効率化
- ライフサイクルの恐怖！

ちょっと欲張っちゃいました！

プログラムのソースコードは
今回は登場しません。
プログラマじゃないという方
ご安心ください（？）

自己紹介

- 実は「現役の教師」、そして「フリープログラマ」
  - 専門学校HAL大阪で常任の教官としてゲーム、制御・ロボット系の分野を幅広く指導
  - 一方、個人事業（屋号：エモーションプラス）としてソフトウェア開発関連事業を手掛けている。

- 得意分野はゲームソフト開発、マイコン制御ソフト開発など
  - 教師になる前はゲームメーカー数社でハードウェア制御ソフト開発業務やゲームソフトウェア開発業務などを手掛ける。
  - 過去にPlayStation用ソフトやアーケードゲームを開発したこともある。

- すでにAndroid用アプリをAndroid Marketにて配信中！（１本だけですが）
  - お絵描きソフト「ポケットスケッチ　日本語版」

ケータイで3DCGを！

- 3DCGといえば…？
  - ゲーム！
  - テレビや映画でよく見る！
- しかし、3DCGは「とても重い処理」である！
  - なぜならゲームなどはリアルタイムに動かす必要があるので
    1/60秒単位で描画を完結させなければいけないから。
- じゃ、なぜケータイで3DCGがいるの？
  - やっぱりゲームのため…
  - 流行りのAR（拡張現実感）とか…

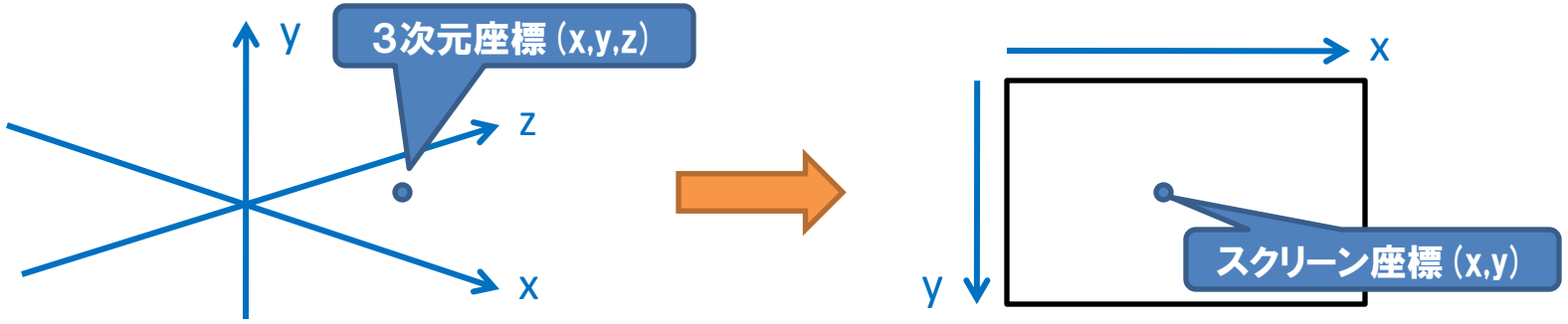
そう、情報表現手段の一つとして
3DCGを扱う必要性が増加してきている！

OpenGLとは？

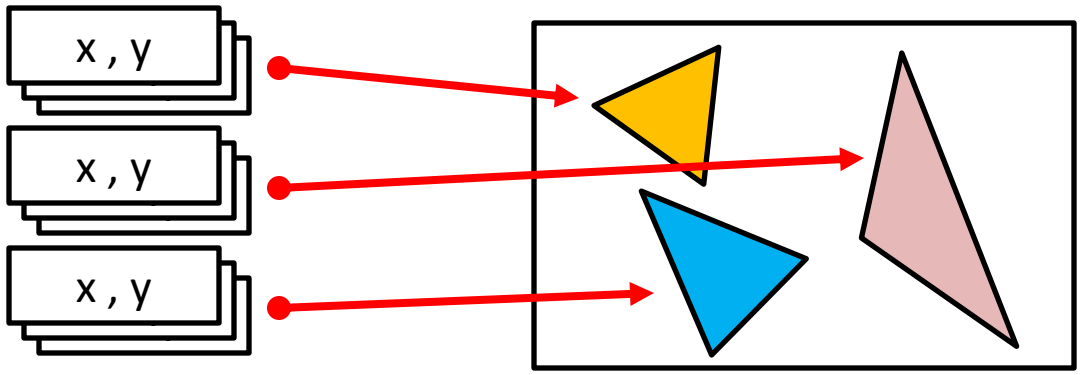
- 米Silicon Graphics(シリコングラフィックス)社が中心となって開発した、3次元グラフィックス処理のためのAPIのこと。
- 現在のAndroidには「OpenGL ES」という組込機器用のOpenGLが搭載されている。
- 携帯電話などの組込機器はパソコンなどに比べるとメモリの量やマシンパワーなどに制約が多く、通常のOpenGLでは荷が重過ぎるので、いくつか機能を削り取り重要なものだけを抜粋したものがOpenGL ESである。
- OpenGL ESにもバージョンがあり、Androidは公式サイトによると1.0をサポート。SDKには1.1の主要APIも含まれているが、「不完全なので使わないで」という事らしい…。

3DCG処理とOpenGL(１)

- 3DCG処理に必要不可欠な要素とは…？
  - 3次元座標 (x,y,z) をスクリーン座標 (x,y) に変換する**座標変換計算処理**

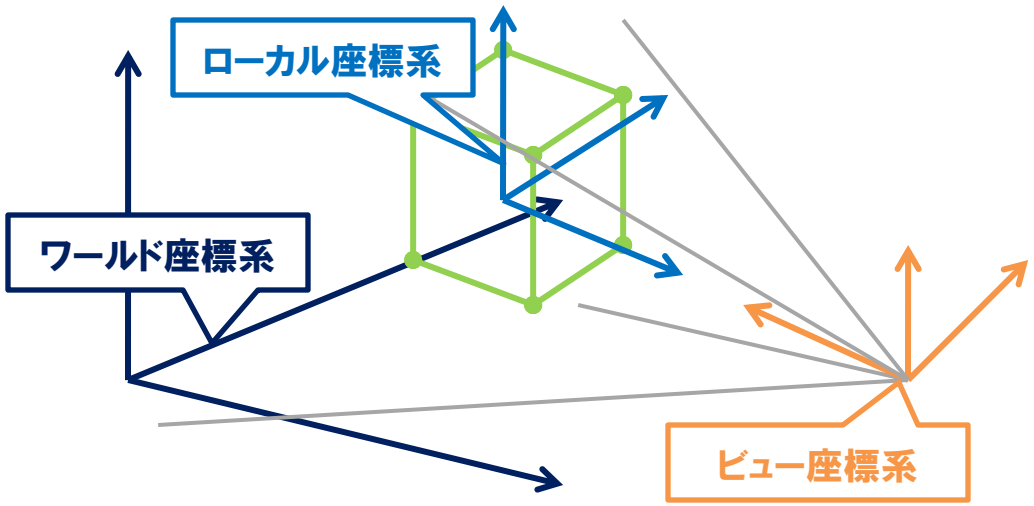


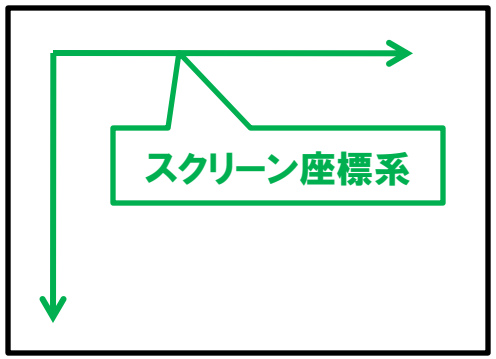
  - 複数の座標をまとめてポリゴン(三角形が多い)を描画する**描画処理**



3DCG処理とOpenGL(２)

- 実際には…
  - 複数ポリゴンをまとめてモデル(メッシュと呼ぶ)として扱う
  - メッシュ内の各頂点(ローカル座標)を「ワールド空間上の座標」に変換
  - カメラ(視点)からみたビュー座標系にさらに変換
  - 透視変換処理によって遠近感をつけて変換
  - スクリーン(表示デバイス)の座標系(ビューポート)に変換
  - 頂点データに基づいてメッシュを一気に描画





3DCG処理とOpenGL（３）

- 多くの場合、これらと付き合うことになる！
  - **4次元ベクトル**（x,y,z,w）
    …回転と並行移動を表現するためには4次元の方がいい
  - **4x4の行列**（マトリクス）
    …座標の回転、平行移動、スケーリングなどに大活躍！

$$(x', y', z', w') = (x, y, z, w)\begin{bmatrix} m00 & m01 & m02 & m03 \\ m10 & m11 & m12 & m13 \\ m20 & m21 & m22 & m23 \\ m30 & m31 & m32 & m33 \end{bmatrix}$$

これは4次元ベクトルを
4x4マトリクスで変換する
基本的なパターン

- こんな複雑な処理をやらないと何も表示されない！

OpenGLはどこまでサポートしてくれるのか？

3DCG処理とOpenGL（４）

- OpenGLがサポートする部分は、主に
  - 各種マトリクスの生成
  - マトリクス、ベクトルを使った演算
  - 頂点データの座標変換処理
  - ポリゴンの描画処理

…つまり、「回転角度、位置、向きなどをどのようにコントロールするか」「モデルデータをどのようにして扱うのか」など**アプリケーションのアルゴリズムに深く関係する部分はサポートされていない**と考えてよい。

結論：

OpenGLを使うためには3DCGの基礎知識が必要！

OpenGLとGPU

- 3DCGに必要な「座標変換処理」「描画処理」は重たい処理なので、できるだけ高速化したい！

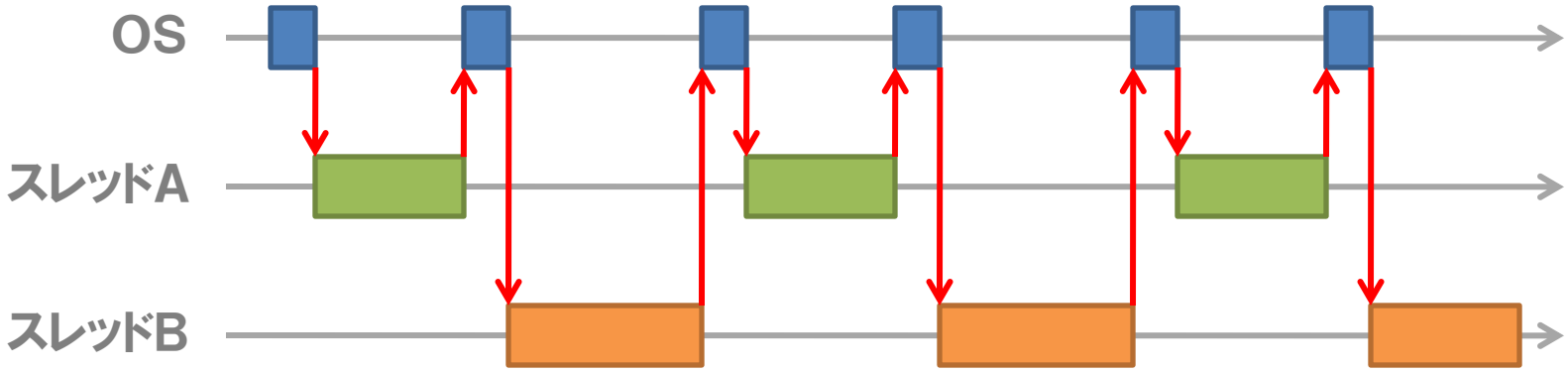
- CPUがこれらを処理するのではなく、**グラフィック処理専用のプロセッサに任せてしまえ！**という発想が生まれた。このプロセッサがGPUである。

- よって
  - CPU…アプリケーション本体の処理
  - GPU…グラフィック用の演算・描画処理

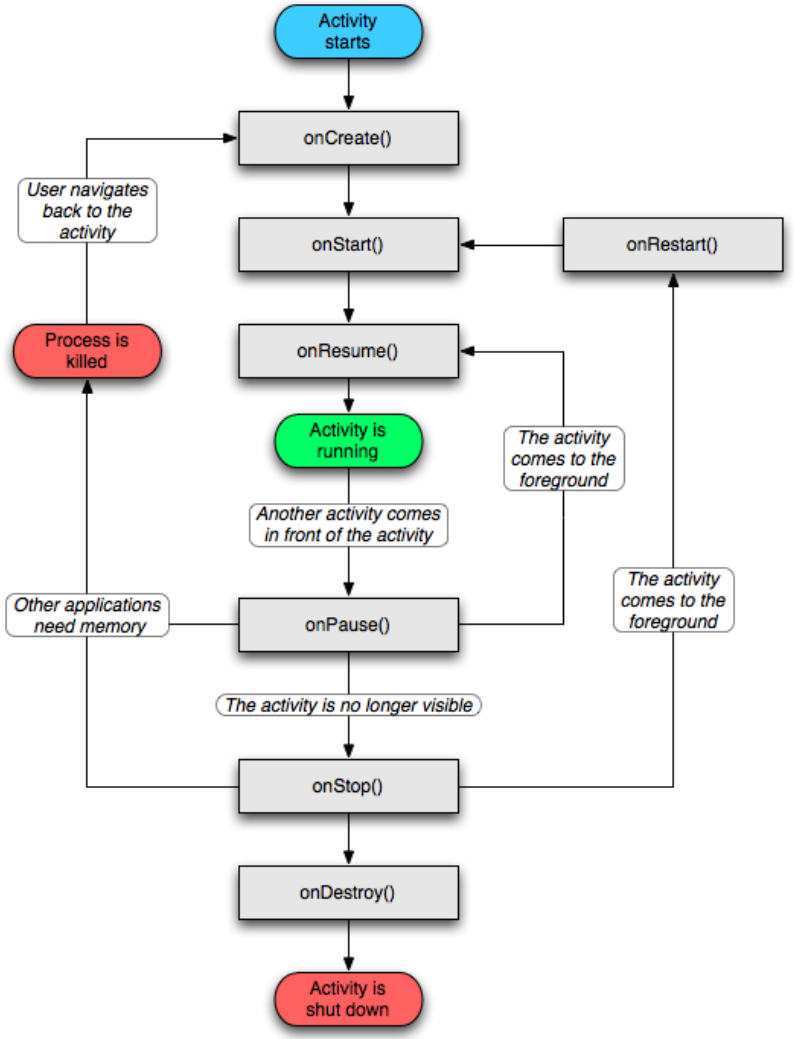
  という「**2つのプロセッサによる平行動作**」という形になる。

マルチスレッド処理とは？

- **スレッド…１つの処理単位**
- このスレッドをアプリケーション内で複数生成し、これらを見かけ上同時に動作させている処理をマルチスレッド処理と考えてください。
- CPUが１個しかない場合は**OSが非常に短い時間でタスクを次々に切り替える**ので、あたかも同時に動いているかのように見えます。



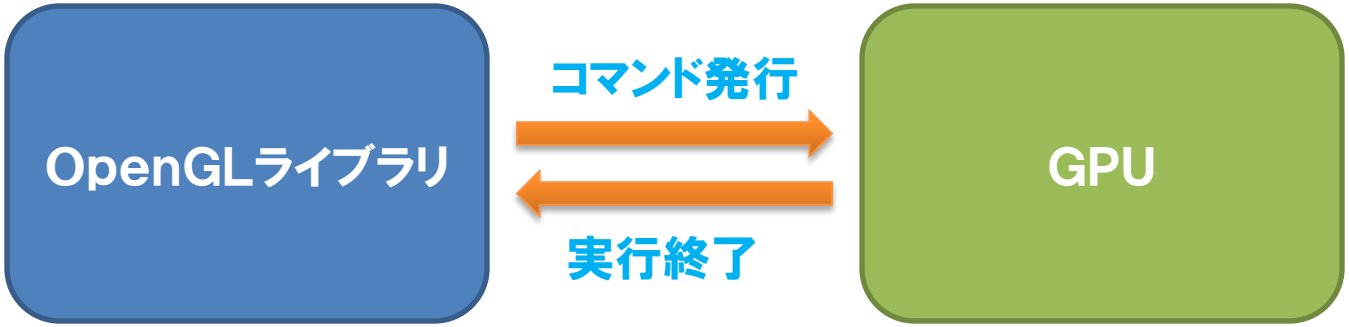
ライフサイクルとは？



- これを理解しないとまともなアプリは組めない！
- アクティビティは端末の状況によって強制的に状態が変化すると考えてよい。
  （例）アプリ実行中に突然電話がかかってきたら…
- 状態によってはOSから強制的に終了させられることもあり得る！
- つまり、アプリケーション実行時は常に「割りこまれて状態が変化しうるんだ」ということを理解する必要がある！

描画時に起こっていること

- 実はOpenGLはGPUにコマンドを発行して実行させて結果を受け取るという動作をすることがある。



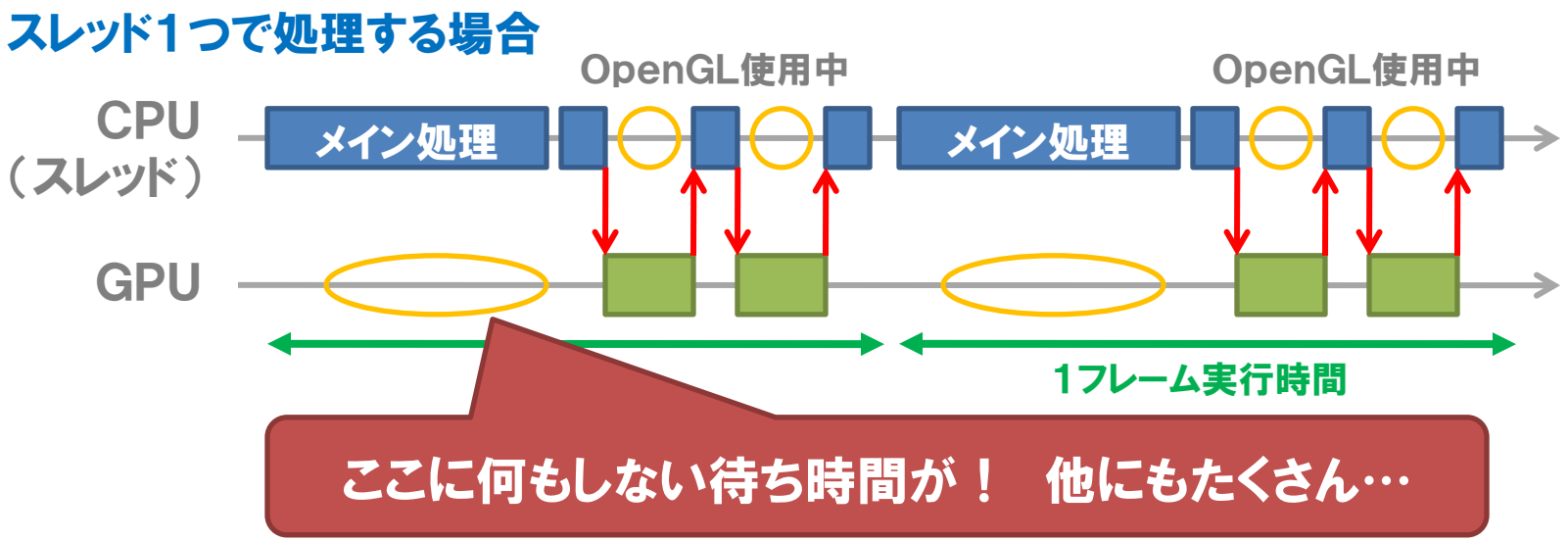
- …ということは、コマンド発行から結果受け取りまでの間、CPUは「待ち状態」になっています！

この待ち時間も有効に使いたい…

そう！**マルチスレッド化してしまう**のです！

マルチスレッドによる効率化(１)

- CPUとGPUが同時に動作できることを知らずに
  シングルスレッドで処理フローを構築すると…



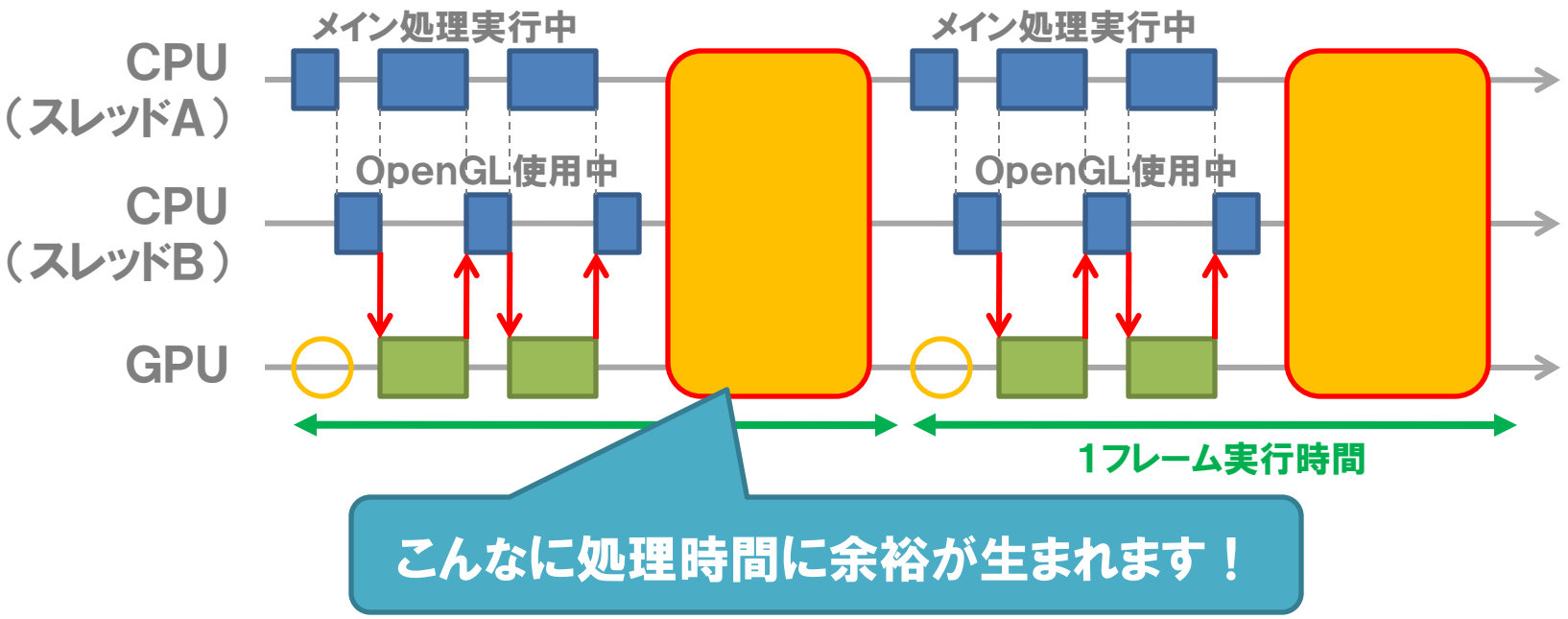
これではせっかくのマルチプロセッサ構成が生かされない
処理フローになってしまいます！

ああ、勿体無い、モッタイナイ、MOTTAINAI…

マルチスレッドによる効率化(２)

- CPUがメイン処理を実行する部分とOpenGL命令を主に処理する部分を別スレッドにすると効率化が図れる！

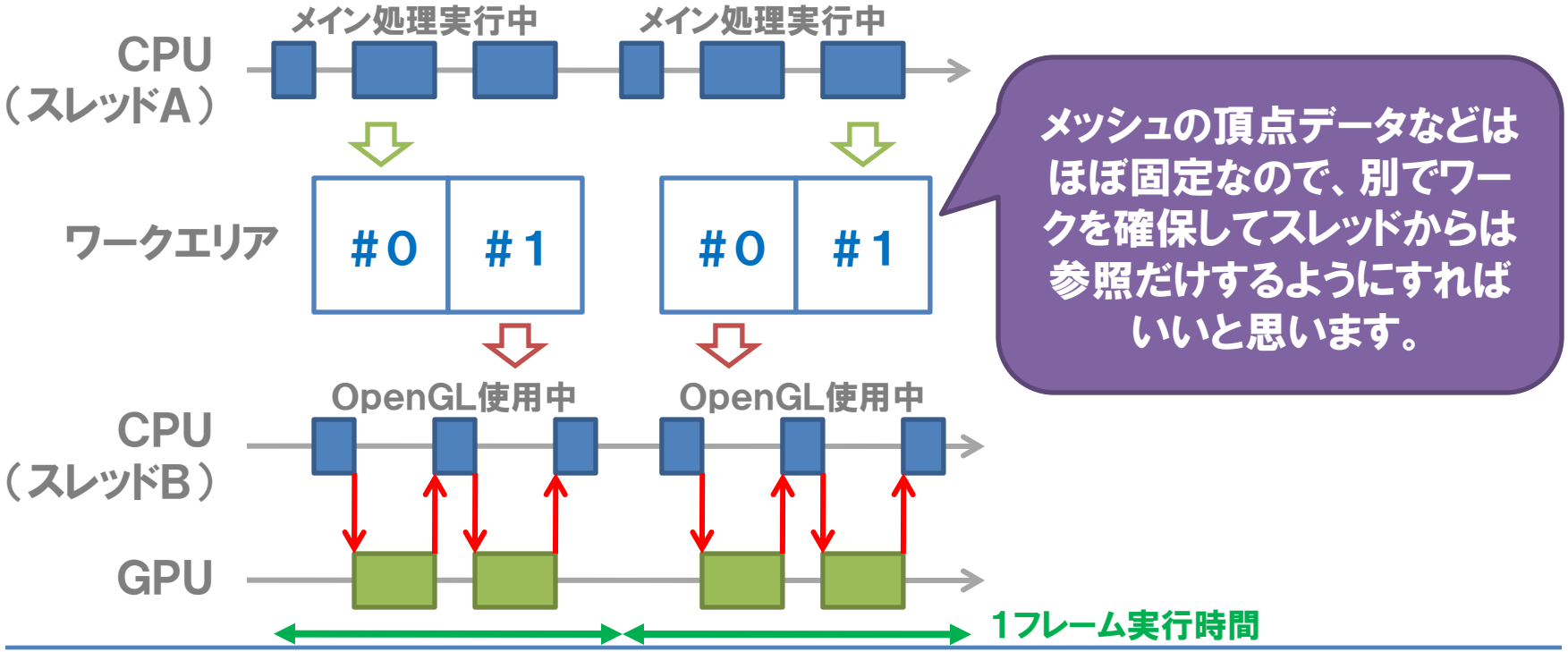
スレッド2つで処理する場合



マルチスレッドによる効率化（３）

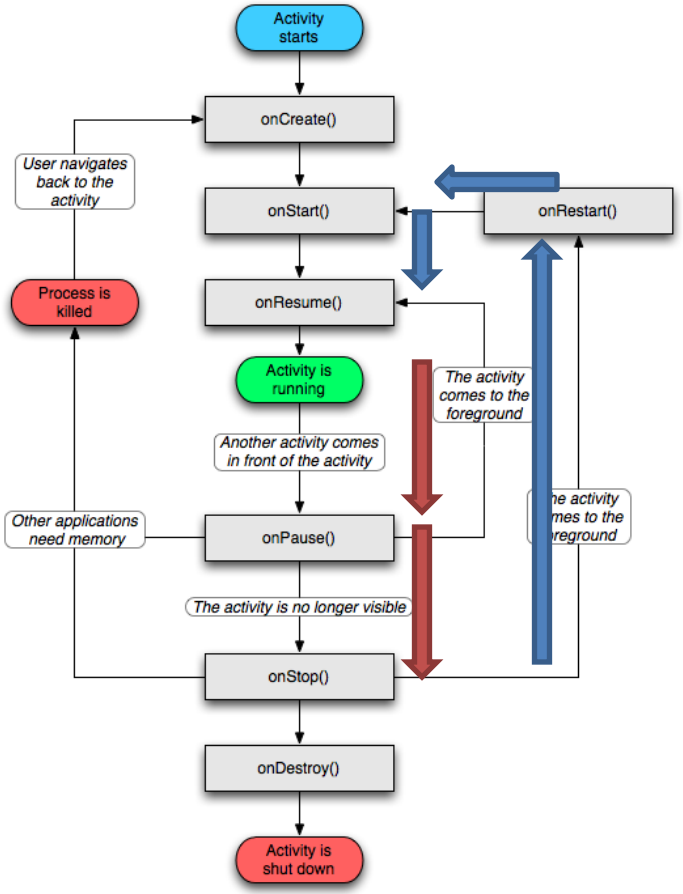
- 実装時は2つのスレッドから共通で扱えるワークエリアを用意しておき、それぞれのスレッド間でデータの受け渡しができるようにする。
- 同じワークに同時アクセスすることを防ぐため**ワークをダブルにしてフレームごとに交互に切り替える方法が有効**と思われる。

※いわゆるダブルバッファリング



ライフサイクルの恐怖！（１）

- もしゲーム実行中に電話がかかってきたら…？



- 画面が突然切り替わって…実はアクティビティの状態も突然切り替わる！
- 電話がかかってくると
  onPause、onStopと続けてコールされる
- 電話が切れると
  onRestart、onStart、onResumeと続けてコールされる
- onPauseを通過すると、常に「強制終了」の恐怖が付きまとう…

ライフサイクルの恐怖！（２）

- つまり…

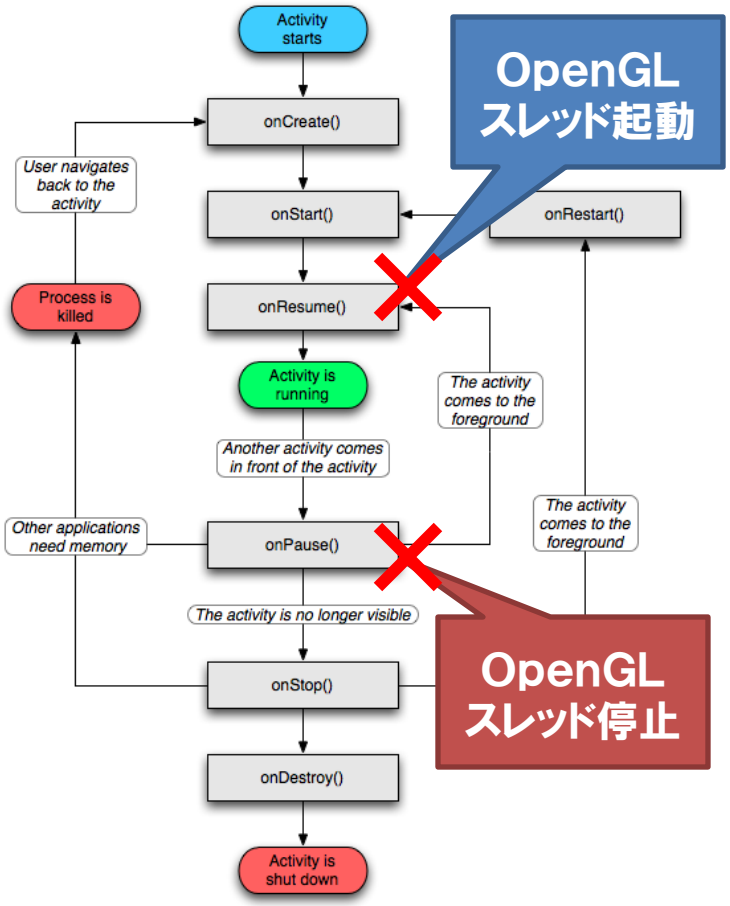
  こちらが意図しないタイミングでアプリが
  勝手に終了してしまう可能性がある！

- ということで動いている**スレッドをどうすればいいの？**
  という疑問＆不安が…。
  - onPauseが走った段階でスレッドを停止？
  - どのタイミングでスレッドを再起動すればいいのか？
  - スレッドを動かしっぱなし…じゃアカンのん？

結構面倒くさそうな予感がしませんか？

ライフサイクルの恐怖！（３）

- onResumeでOpenGLスレッド起動、onPauseでOpenGLスレッド停止



- もしOpenGLスレッドのrunメソッド内部でEGLを使った初期化処理が入っていると…このパターンではNG！
- なぜならEGLでの初期化はすでにサーフェスが生成されていないとうまくいかないから。
- …ということは、onResumeではなくて、SurfaceHolder.Callbackで呼び出されるsurfaceCreatedメソッド内でスレッド起動をかけるとうまくいきそう。
- さらに、サーフェスに依存するということであればスレッドの停止はonPause内ではなくSurfaceHolder.Callbackで呼び出されるsurfaceDestroyedメソッド内でスレッド停止をかける必要がある。

ライフサイクルの恐怖！（４）

- ゲームなどを作っているとこんなコトしたくなります。

  他のアプリが割りこんできてもゲームスレッドやOpen GLスレッドを終了させずにそのままバックグラウンドで生かしておきたい！（ポーズ状態にするとしても）

- 理由：割り込んだアプリが終了したらすぐに復帰させやすい

  ・・・しかし、CPUパワー、メモリ、バッテリーのことを考えるとオススメできません！

アプリがバックグラウンドになったらスレッドを停止させて不要なメモリを解放してあげましょう！

ライフサイクルの恐怖！（５）

- でもでも、どーしてもスレッドをそのまま生かし続けたいんや･･･（気持ちはわかる）
- **では、実際にやってみるとどーなるか？？？**

- 当然ライフサイクルのどこかに「スレッド起動」処理があるので、**そこを通過するたびにスレッドがボコボコと増殖する**ことになります…**嗚呼恐ろしや！**
- **対策：スレッド起動時に起動済みかチェックする手も**

※しかし、ちょっと面倒くさいですよ。

結論

- スレッドがいつ停止しても大丈夫なよう、アプリの処理フローをしっかり考えること！

- 場合によってはスレッド停止・起動時に前回実行時の情報を保存・復元できるような処理を入れること！

- PCと異なりバッテリーで動く機械であるという認識を強く持って、使用するメモリ量の削減、効率のよい処理フローの構築などAndroidにとって優しいプログラムを開発しましょう！

最後に

- 今回はソースコードがありませんでしたが、現在**実験中**だからです。
- もし今後機会があれば続きを発表するかもしれません。

- 実は**SDK1．5から「GLSurfaceView」なるものが追加**されています。（なぬ？）
- これを使うと**「今までの苦労は一体・・・（以下略）」**。

以上です。どうも有難うございました！