

# セカイカメラ開発から見た Androidアプリケーション

## 開発の現状と Android内部構造

日本システム開発株式会社 第2事業部 石原正樹  
m.ishihara@nskint.co.jp (twitter: @ishihara\_twit)



# 会社紹介

## ■ 日本システム開発株式会社

### ▶ ソフトウェア開発事業

#### ◆組込み関連

- Android/組込みLinux/iTRONなど

#### ◆etc...

### ▶ 教育事業

#### ◆組込みLinux/単体テスト/etc...

#### ◆年内にAndroid教育サービスを開始予定!!

### ▶ 詳しくはこちら

◆ <http://www.nskint.co.jp/>

# 序文

- 現在「Android」は携帯電話だけではなく、組込みOSとしての注目を集めしており、組込みOSの潮流は確実に「Android」へ向きつつあります。
- この場には、これからAndroidのソフトウェア開発に取り組む、または開発受託を想定しているベンダー様も多いはず…。

# 序文

- 弊社では、頓智・様との協力体制のもと、Android版「セカイカメラ」の開発を行いました。
- その開発実績の中から、Androidアプリケーション開発の現状と、開発にあたり留意しなければならない点を、Androidの内部構造もふまえ紹介します。

# Agenda

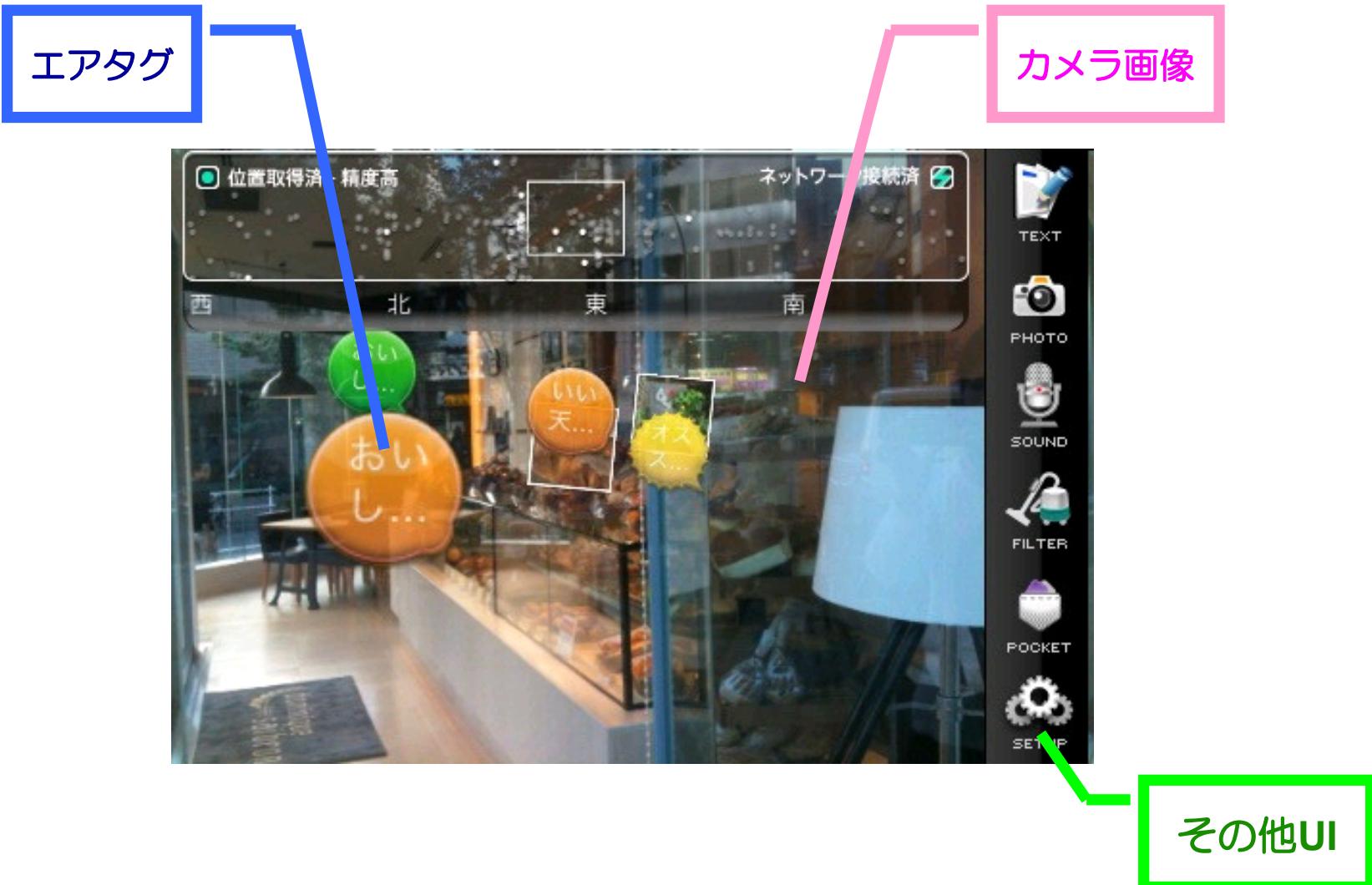
- Android版「セカイカメラ」の特徴
- Android版「セカイカメラ」開発における障害分析
- 障害例 1 : SurfaceViewとZ-Order
- 障害例 2 : SurfaceFlingerのメモリ不足
- まとめ

---

# Android版 「セカイカメラ」 の特徴

---

# Android版「セカイカメラ」の特徴



# Android版「セカイカメラ」の特徴



- どんな要素があるか?
  - ▶ 「エアタグ」のリアルタイム描画
  - ▶ カメラ画像のリアルタイム描画
  - ▶ その他UIの表現
- メインとなる要素はリアルタイム描画が必要
  - ▶ ゲームアプリケーションのような高速描画処理が要求される

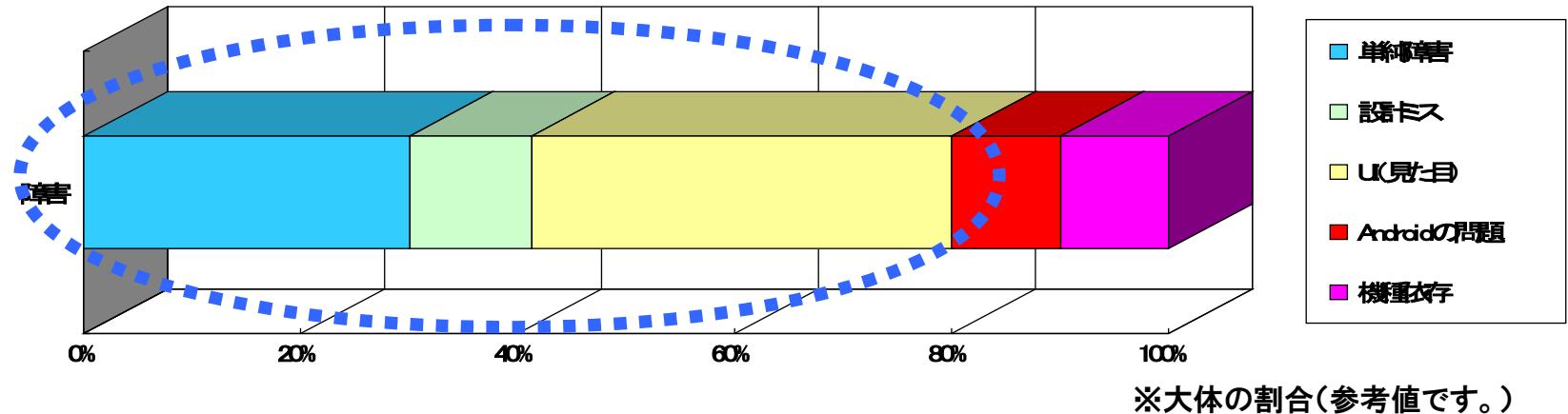
# Android版「セカイカメラ」の特徴

- Androidで高速な描画を表現するには…
  - ▶ 一般的に「**SurfaceView**」というコンポーネントが利用されます。
- もちろん、Android版「セカイカメラ」でも「**SurfaceView**」の機能を利用しています。

# Android版「セカイ カメラ」開発における障害分析

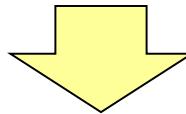
# Android版「セカイカメラ」開発における障害分析

- Android版「セカイカメラ」の開発では、実際にどのような問題（障害）が発生したのか？
- 以下、Android版「セカイカメラ」の初期開発における検出障害の割合



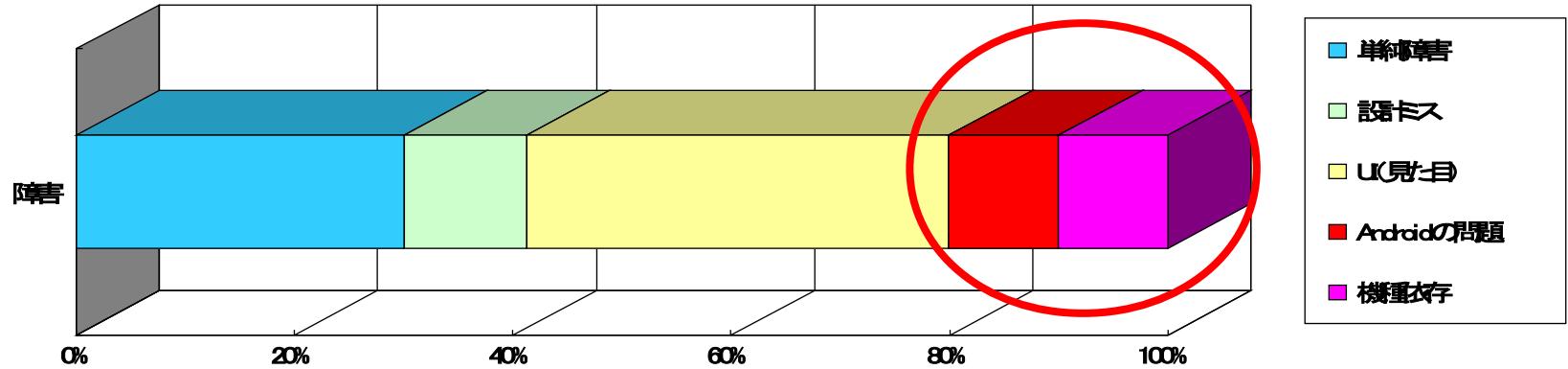
# Android版「セカイカメラ」開発における障害分析

- 各障害分類を分析すると…
  - ▶ 「単純ミス」は人的なもののなのでパス…。
  - ▶ 「設計ミス」「UI（見た目）」はAndroidフレームワークの理解不足によるものが多い。
- これらの問題の多くは比較的、開発中の早期に解決されました。
- また、要員のスキル的な問題であるため、他の開発ベンダ様にとっても、時間とともに解決されるものであると考えています。



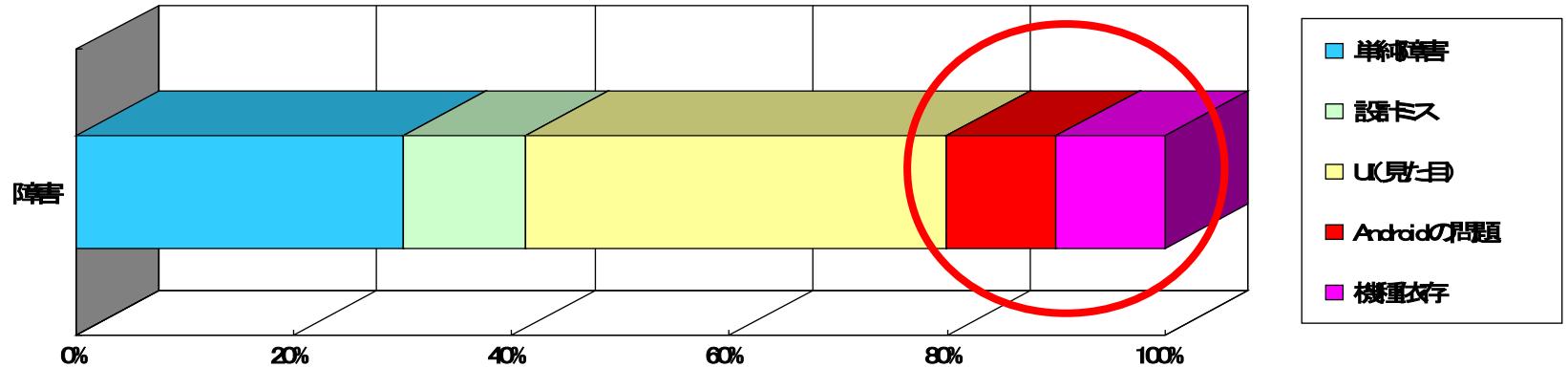
# Android版「セカイカメラ」開発における障害分析

- 開発の終盤で苦しめられた問題の多くは…



- Androidのプラットフォームバグと思われる現象、または、特定の機種（Android バージョン差分も含む）だけで発生する障害でした。

# Android版「セカイカメラ」開発における障害分析



## ■ これらの問題の多くは…

- ▶ 発生が予測しにくい（他の機種では動いていたし、アプリには問題はないと思うけど…）
- ▶ 原因が特定しにくい（ログも出ないし、システムがハングアップすることもあるし…）

# Android版「セカイカメラ」開発における障害分析

- 次のページからAndroid版「セカイカメラ」の開発で実際に検出し、開発終盤に苦しめられた障害例を紹介します。

---

# 障害例 1 :

# SurfaceView と

# Z-Order

# 障害例 1 : SurfaceViewとZ-Order

- 高速な描画レイヤを複数必要とするアプリケーションの場合、**SurfaceView(GLSurfaceView含む)**をレイヤとして複数枚扱いたいケースが存在します。



# 障害例 1 : SurfaceViewとZ-Order

## ■ まず「Z-Order」とは？

- ▶ レイヤの階層をあらわす番号

Z=11015

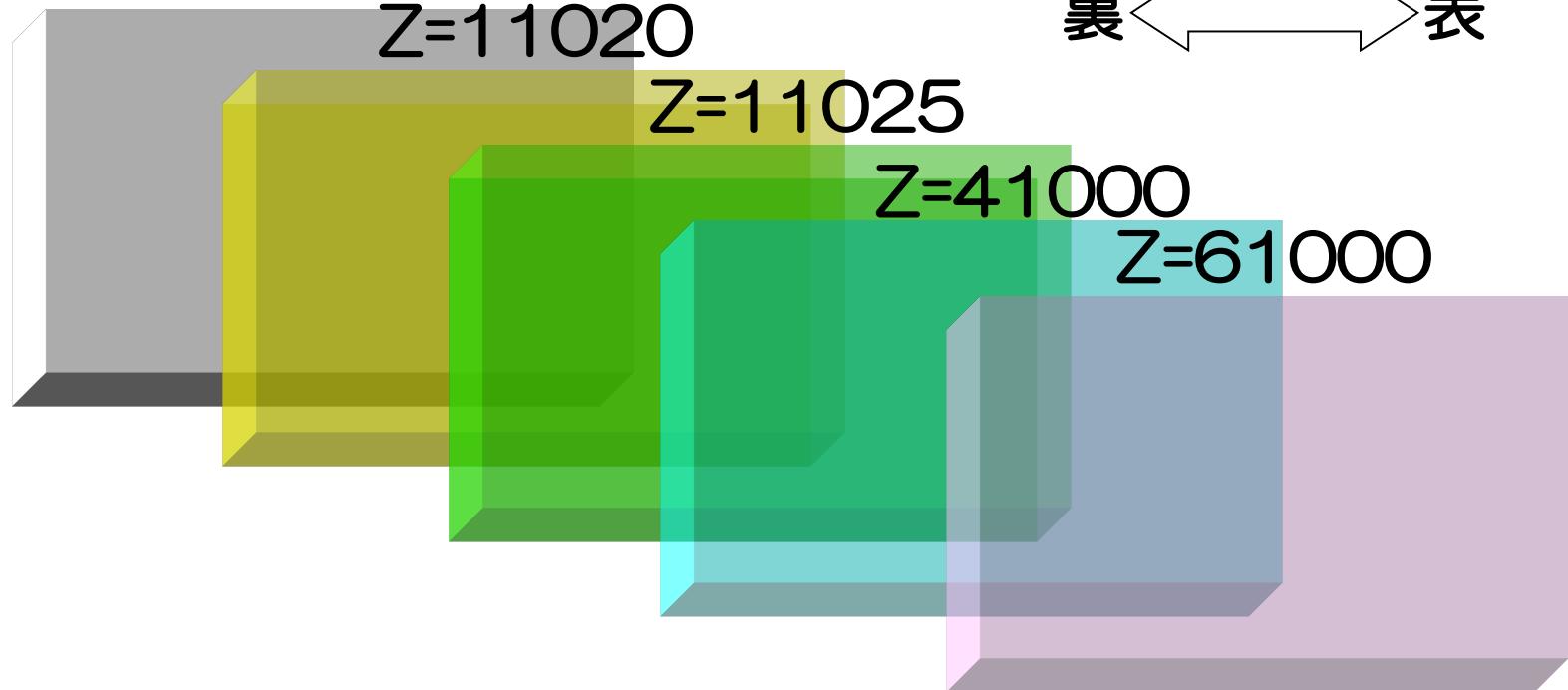
Z=11020

Z=11025

Z=41000

Z=61000

裏 ← → 表



# 障害例 1 : SurfaceViewとZ-Order

- **SurfaceView**をレイヤ（階層）として扱う際に発生する問題点



- この状態でアプリケーションを起動、または、停止状態からの復帰を行うと…

# 障害例 1 : SurfaceViewとZ-Order



- レイヤが期待する順番で見えない、または、今まで見えていたレイヤが見えなくなる。（Z-Orderがひっくり返る？）

# 障害例 1 : SurfaceViewとZ-Order

- それが発生するタイミングは?
  - ▶ アプリケーション起動時
  - ▶ アプリケーションが停止状態から復帰する時
    - ◆ 別のアプリ呼び出しからの戻り
    - ◆ HOMEキー押下後の再実行
- アプリケーションのライフサイクルと密接に関連…
- かつ、機種により発生するタイミング、レイヤの順番が異なる

# 障害例 1 : SurfaceViewとZ-Order

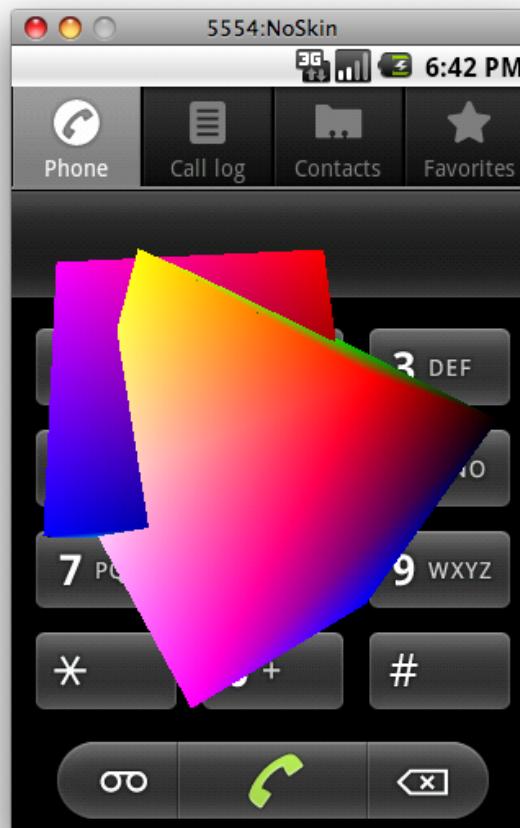
- なぜこの現象が発生するのか？
- まずは Z-Order の割り当てから説明
  - ...

# 障害例 1 : SurfaceViewとZ-Order

- Z-Order は基本「いま見えている範囲」のレイヤにのみ、割り当てが行われます。
- Z-Order は Android OS 内部の **SurfaceFlinger**（後述）により割り当て管理が行われています。

# 障害例 1 : SurfaceViewとZ-Order

■ 例えば、こんな画面なら



# 障害例 1 : SurfaceViewとZ-Order

表



Z=11015



裏

# 障害例 1 : SurfaceViewとZ-Order



Z=11020



Z=11015



表

裏

# 障害例 1 : SurfaceViewとZ-Order



Z=61000

ステータスバー

Z=11020

Activity  
3Dの四角形

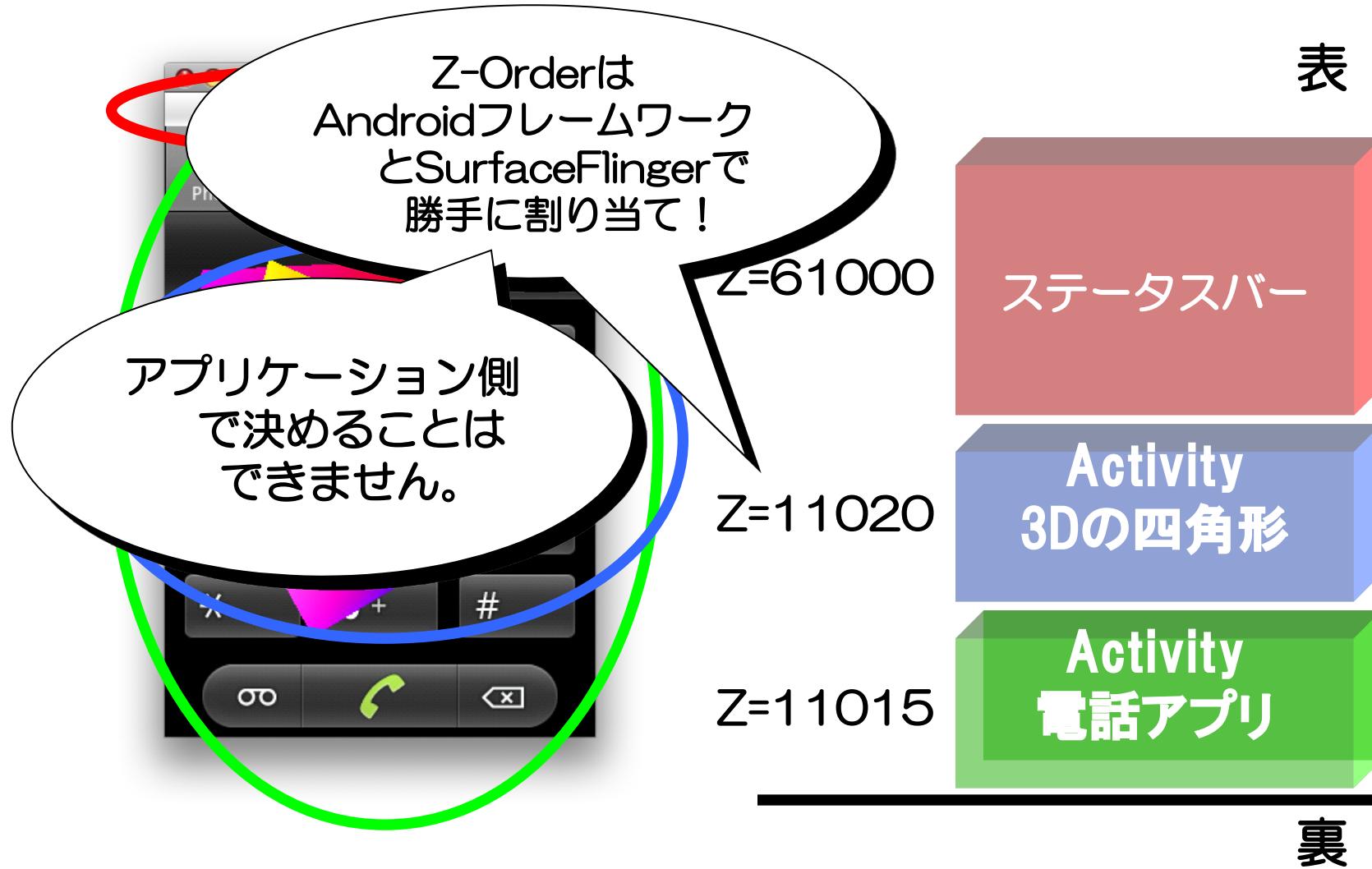
Z=11015

Activity  
電話アプリ

表

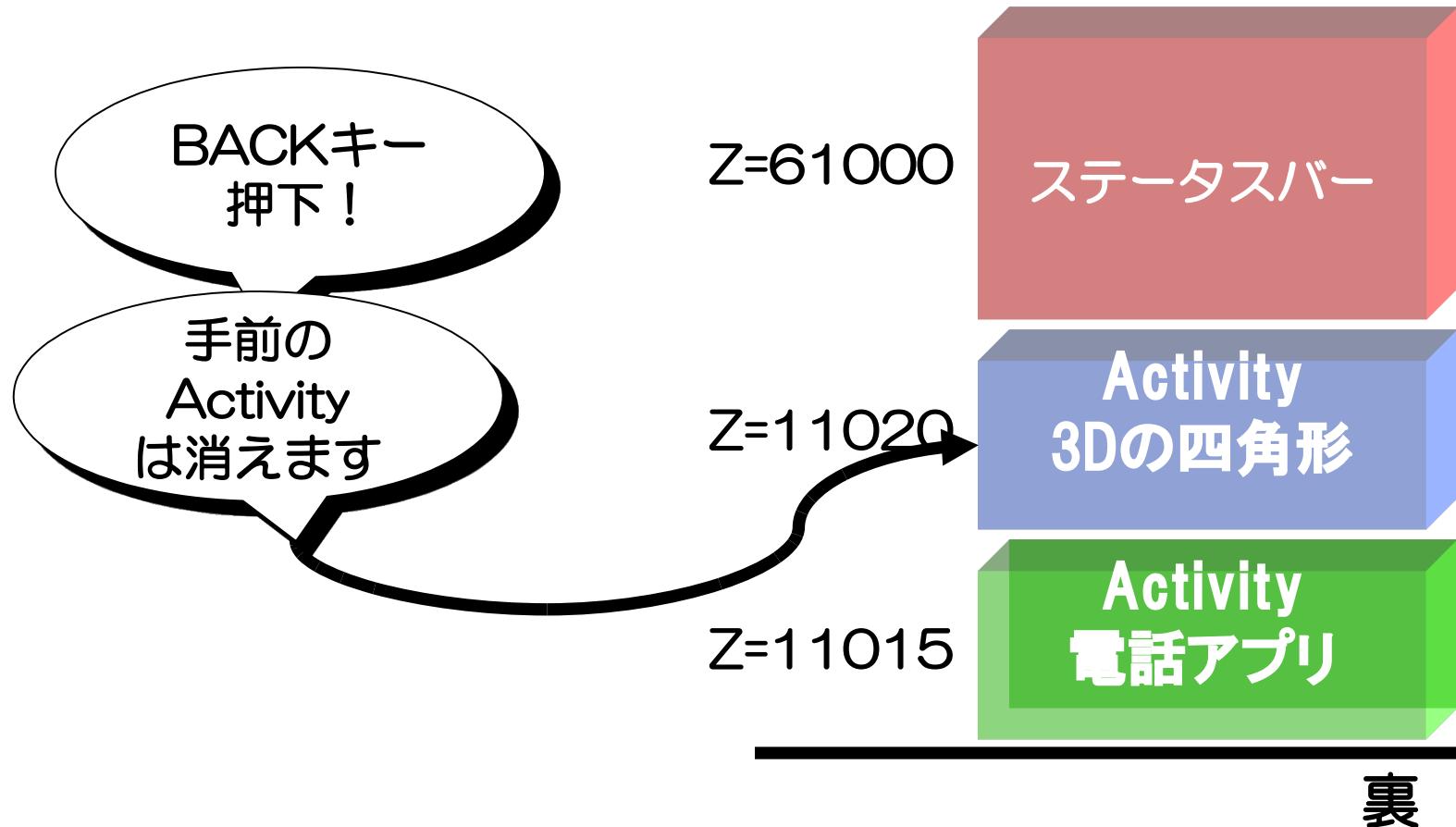
裏

# 障害例 1 : SurfaceViewとZ-Order



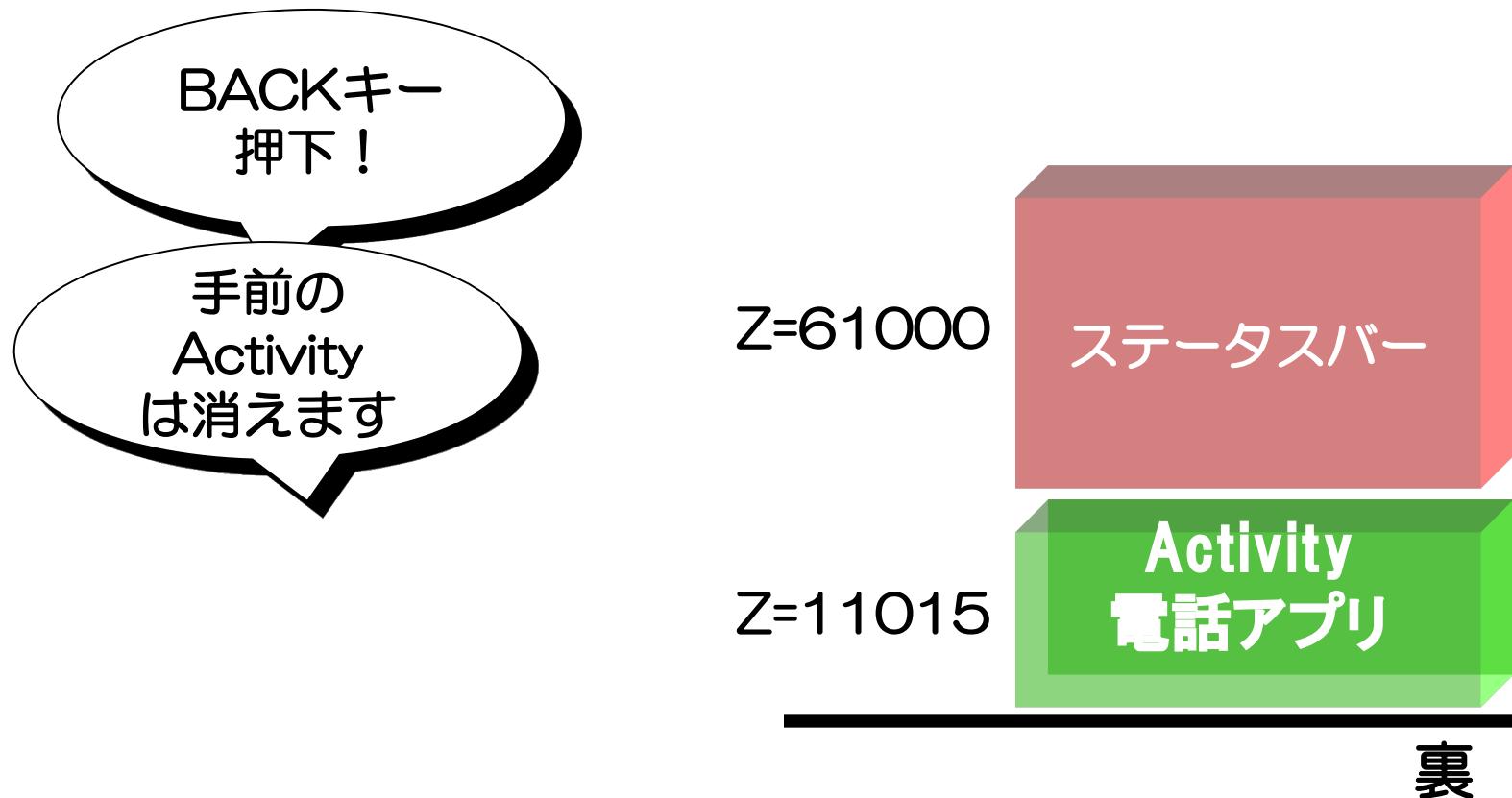
# 障害例 1 : SurfaceViewとZ-Order

- 基本はActivityの単位で管理される



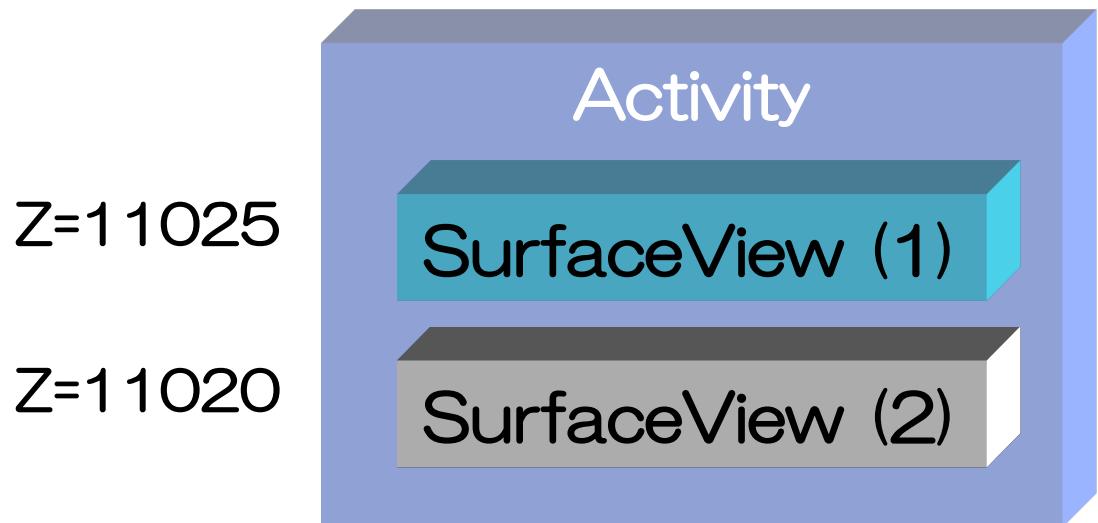
# 障害例 1 : SurfaceViewとZ-Order

- 基本はActivityの単位で管理される



# 障害例 1 : SurfaceViewとZ-Order

- しかし、**SurfaceView** には独立したレイヤ (Z-Order) が必要
- ひとつの Activity の中に複数の可視 SurfaceView が存在すると？

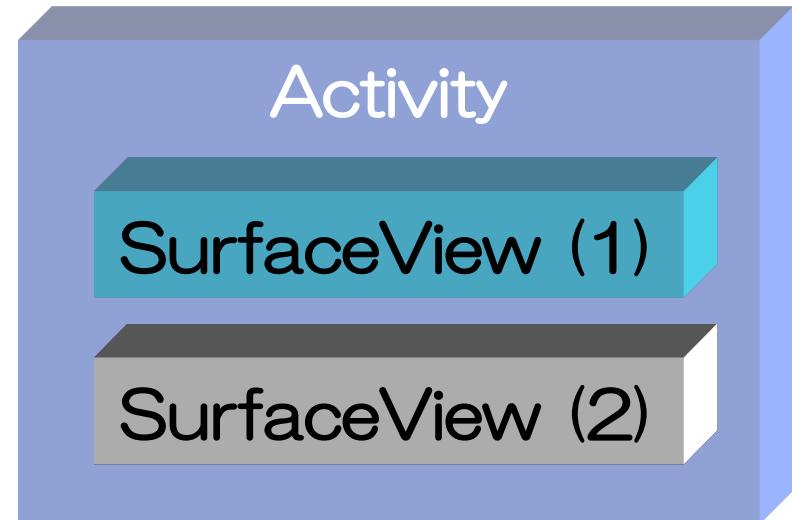


# 障害例 1 : SurfaceViewとZ-Order

- しかし、Activity内にある複数のSurfaceViewは、現状のAndroidでは管理されていないこと…
- Androidフレームワークとして、そういう使いつ方は想定していない！？  
(※詳細は未調査)

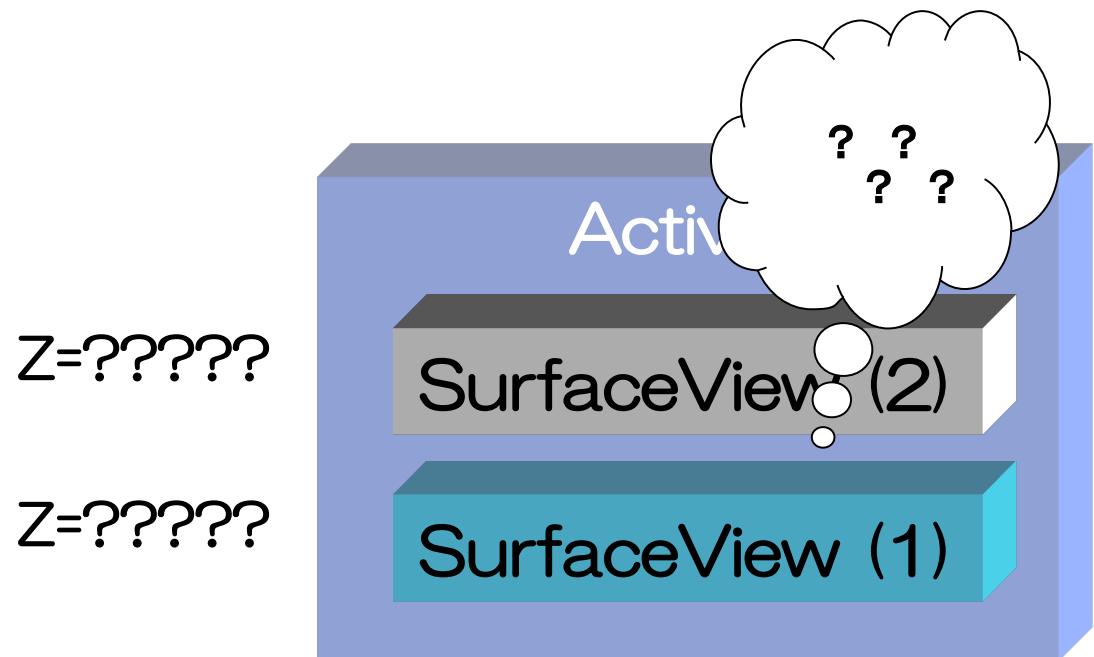
Z=?????

Z=?????



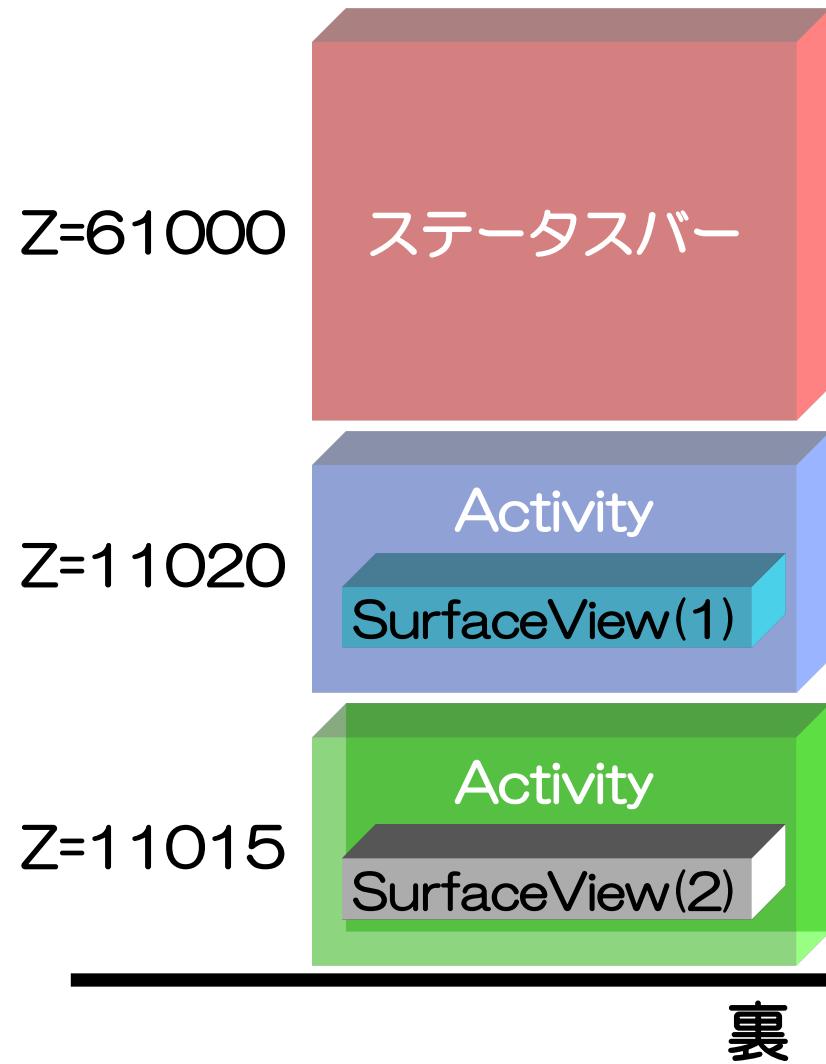
# 障害例 1 : SurfaceViewとZ-Order

- そのため、画面に表示される際の Z-Order 割り当て順序は、**毎回同じ**になる保障がない…



# 障害例 1 : SurfaceViewとZ-Order

- 解決策は？
- 現状の構成を見るにActivityをわける方法が確実



# 障害例 1 : SurfaceViewとZ-Order

- または…
- API level 5 (Android2.0) から、SurfaceView に Z-Order を調整する下記のメソッドが追加されました
  - ▶ setZOrderMediaOverlay()
  - ▶ setZOrderOnTop()
- 完全に制御ができるわけではないですが、調整は可能に

# 障害例 1 : SurfaceViewとZ-Order

## ■ この問題について…

- ▶ Androidフレームワークの想定しない（または間違った）利用方法は、思わぬところで障害を招く
- ▶ しかし、想定しない使い方をしても動作してしまうことが多々あるため、検出が難しい…
- ▶ また、こういったBad KnowHow的な情報はまだまだ少ない…

# 障害例 1 : SurfaceViewとZ-Order

## ■ この問題について…

- ▶ 現状では、Androidフレームワークの利用方法に間違いがないかは、小さなプロトタイプで検証を行い、不安点を取り除くのが一番確実な方法ではないかと考えています…

# 障害例2：

## SurfaceFlingerの

## メモリ不足

## 障害例2：SurfaceFlingerのメモリ不足

- アプリケーションが停止状態から復帰、または、端末の向き（オリエンテーション）が変わったタイミングで…

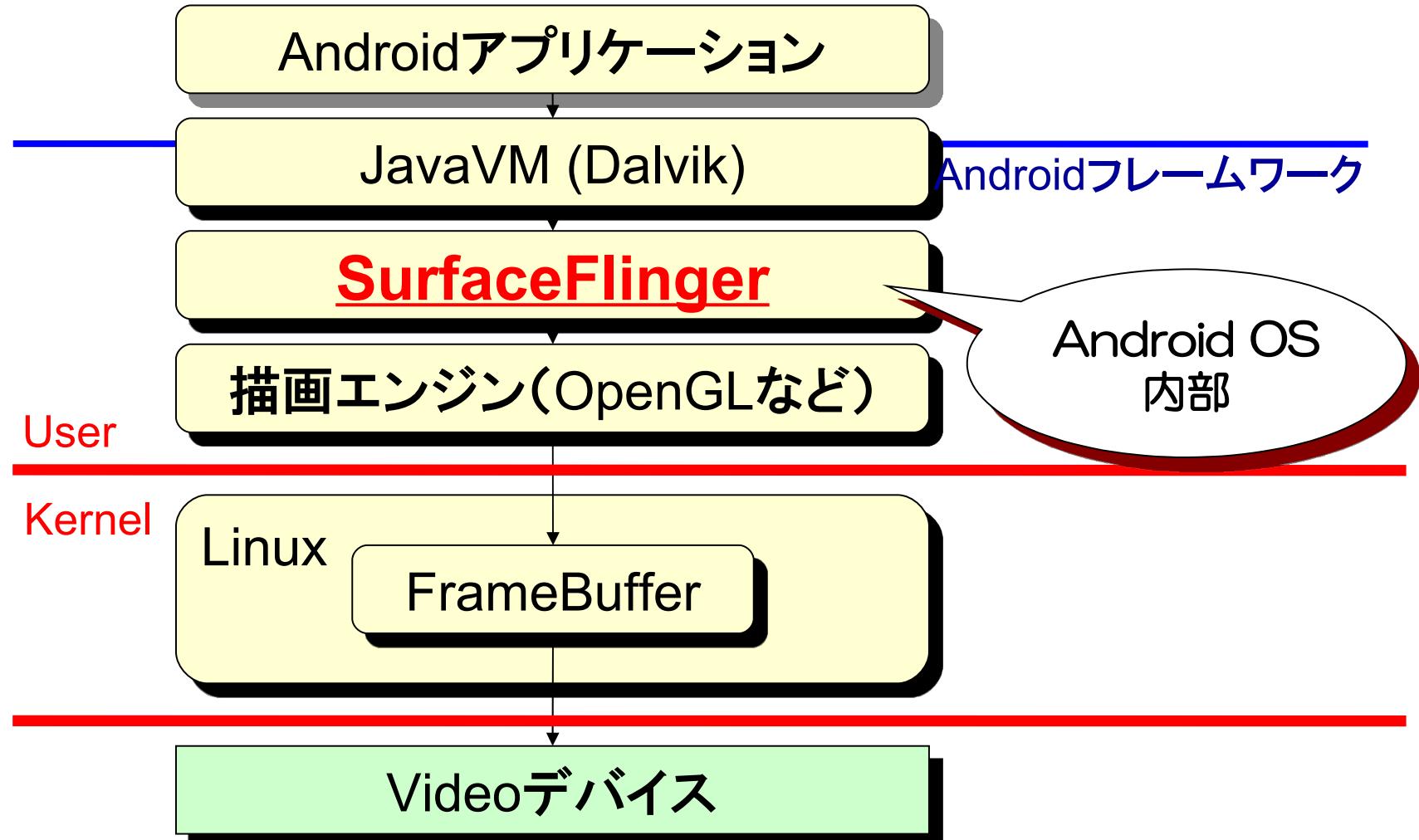
logcat

```
05-31 16:55:09.535: ERROR/SurfaceFlinger(955): not  
enough memory for layer bitmap size=1658880  
(w=854, h=480, stride=864, format=1)
```

- **SurfaceFlinger**のメモリ不足で、  
アプリケーションが**強制終了**させられるという現象

# 障害例2：SurfaceFlingerのメモリ不足

## ■ SurfaceFlingerとは…



# 障害例2：SurfaceFlingerのメモリ不足

## ■問題点

- ▶ SurfaceFlingerがメモリ不足エラーを返却
- ▶ **Androidフレームワーク内でabort()**  
(C/C++のNativeコードの処理)
- ▶ アプリケーションのプロセスが終了…

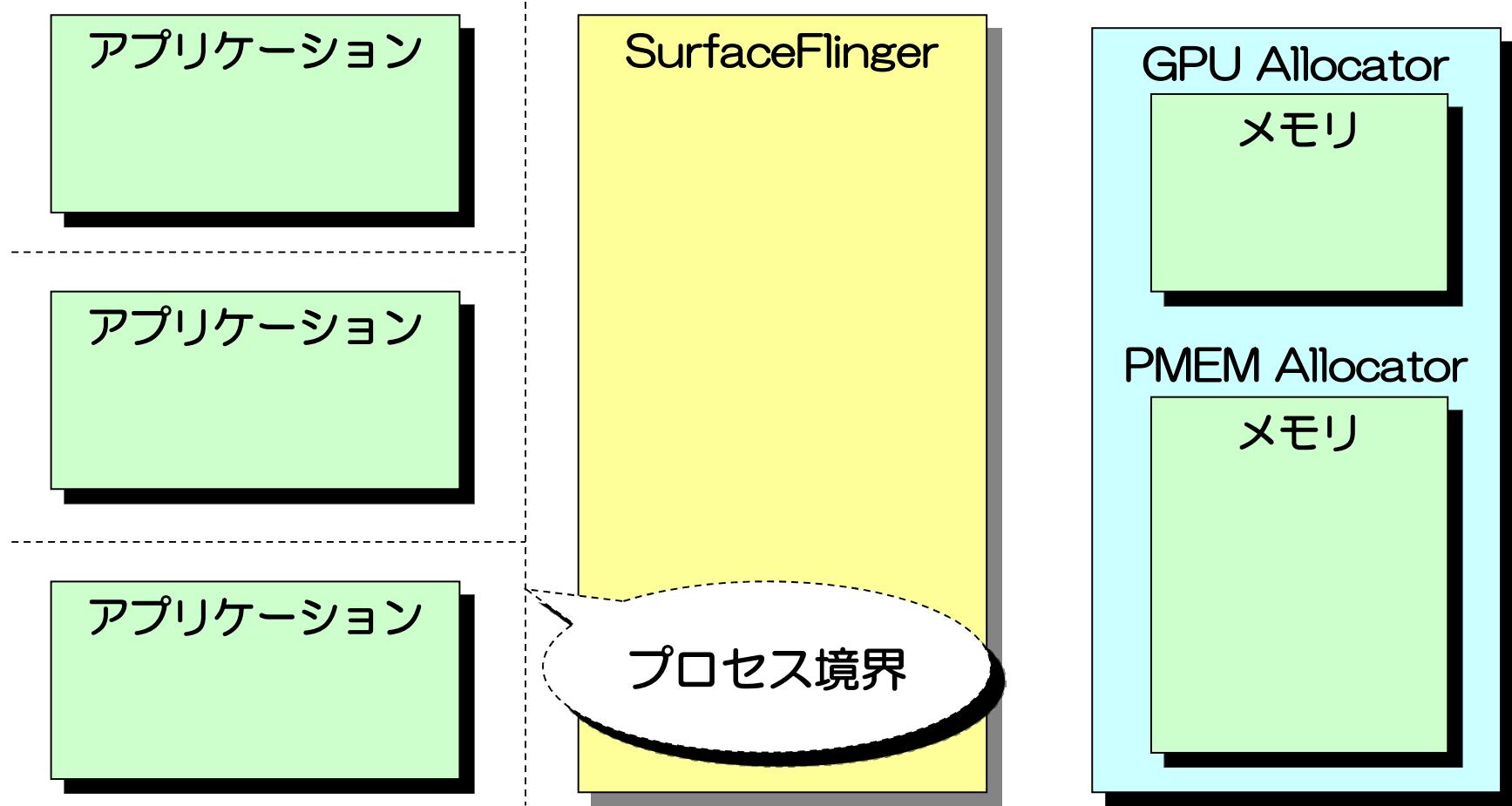
## ■アプリケーションでエラーの発生をハンドリングできません！！！

## 障害例2：SurfaceFlingerのメモリ不足

- なぜこの現象が発生するのか？
- まずは、SurfaceFlingerの構成説明から…

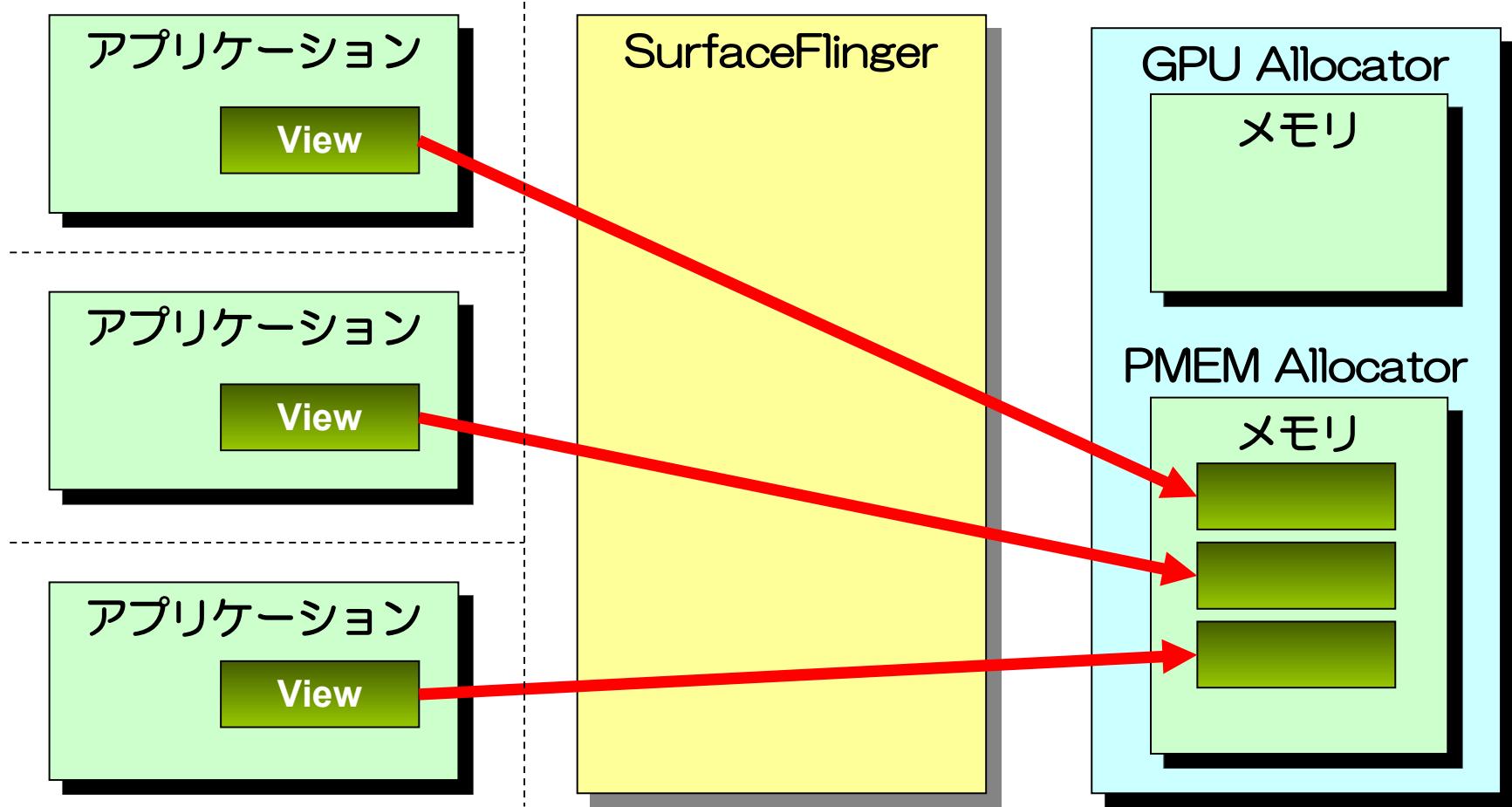
# 障害例2：SurfaceFlingerのメモリ不足

## ■ SurfaceFlingerの構成



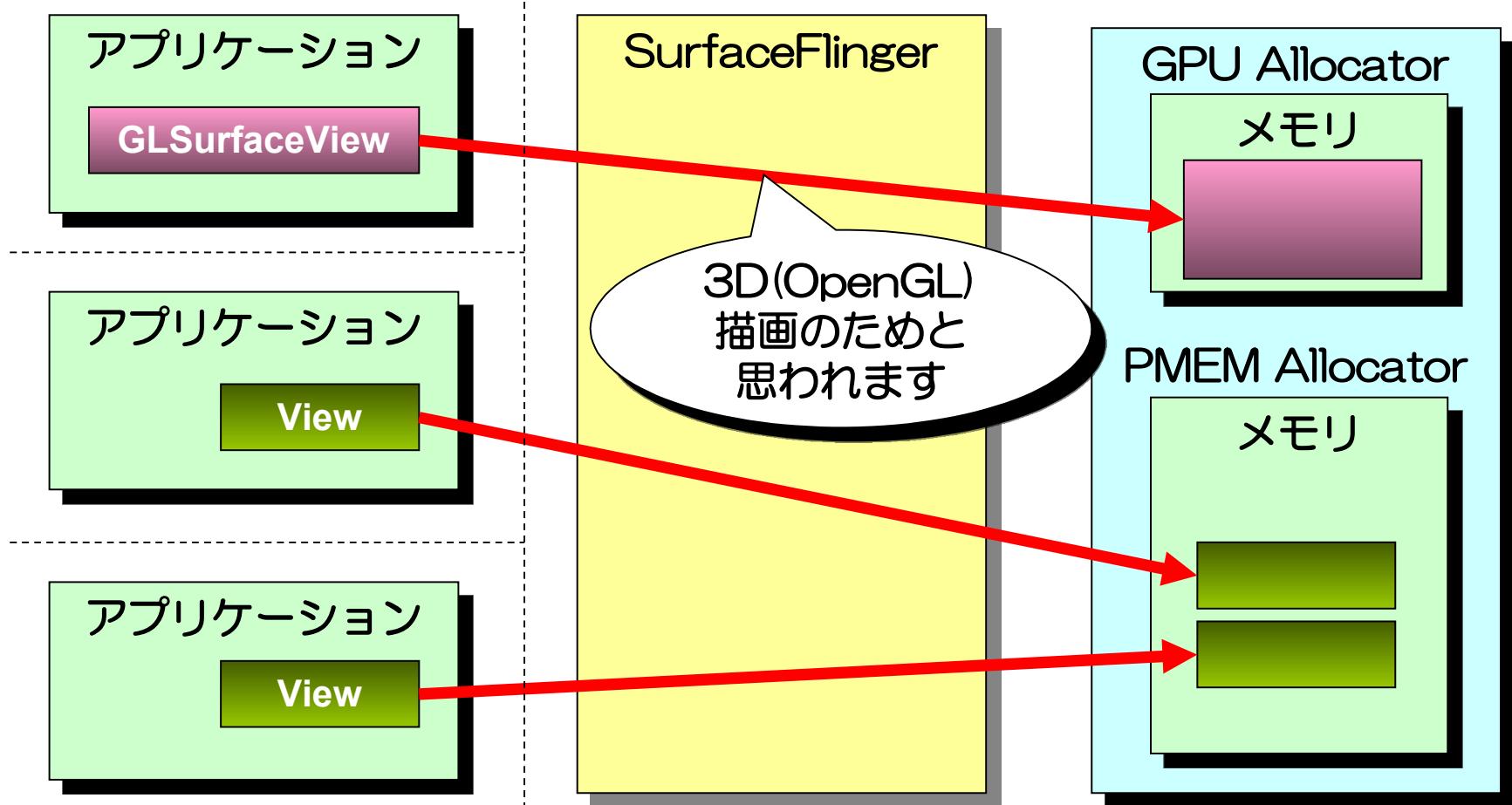
# 障害例2：SurfaceFlingerのメモリ不足

## ■描画メモリの確保（一般的なView）



# 障害例2：SurfaceFlingerのメモリ不足

- GLSurfaceViewの場合は特殊です



## 障害例2：SurfaceFlingerのメモリ不足

- 通常は描画メモリに PMEM を使用します。
  - ▶ PMEM は **8MB～24MB**  
(機種により異なります。)
- GLSurfaceViewは GPU メモリを用します。
  - ▶ GPU メモリは **1MB～4MB**  
(これも機種により異なります。)

## 障害例2：SurfaceFlingerのメモリ不足

- 「SurfaceFlingerのメモリ不足」は、これらのメモリ容量を超えたメモリ確保を要求した際に発生します。
  -
- 特に GPU メモリは容量が少ないので  
**枯渇する可能性が高い！**

# 障害例2：SurfaceFlingerのメモリ不足

## ■ 例) Sony Ericsson Xperia™ (SO-01B)

- ▶ ハードウェア
  - ◆ Qualcomm SnapDragon
    - (GPU内蔵です。)
- ▶ 画面サイズ
  - ◆ 854 × 480
- ▶ PMEM
  - ◆ 16MB
- ▶ GPU メモリ
  - ◆ 約3.2MB

「Xperia」は Sony Ericsson Mobile Communications AB の商標または登録商標です。

### ※ 注意

アプリケーション層から読み取れる情報を元に記載しました。  
内部解析を行ったわけではないため間違いが含まれている可能性があります。  
また、アップデートによる変更もあると思われます。

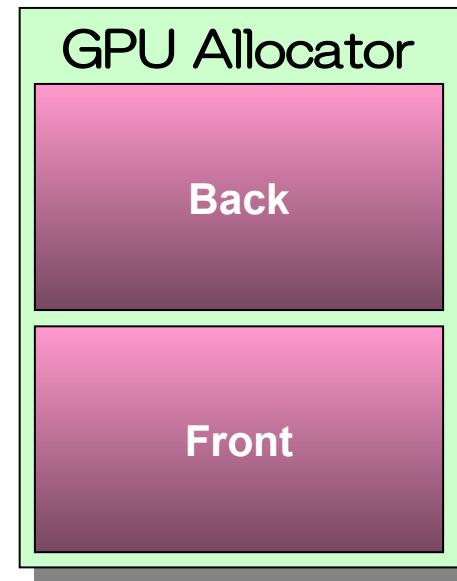
## 障害例2：SurfaceFlingerのメモリ不足

- 例をもとに **GLSurfaceView** の使用メモリを計算
  - ▶ 全画面表示とします。
    - ◆  $854 \times 480 = \mathbf{409920 \text{ dot}}$
  - ▶ 透過機能を使用すると32bitカラー
    - ◆  $409920 \times 4\text{Byte} = \mathbf{1639680 \text{ Byte}}$
  - ▶ Androidはダブルバッファ機構
    - ◆  $1639680 \times 2\text{面} = \mathbf{3279360 \text{ Byte}}$

## 障害例2：SurfaceFlingerのメモリ不足

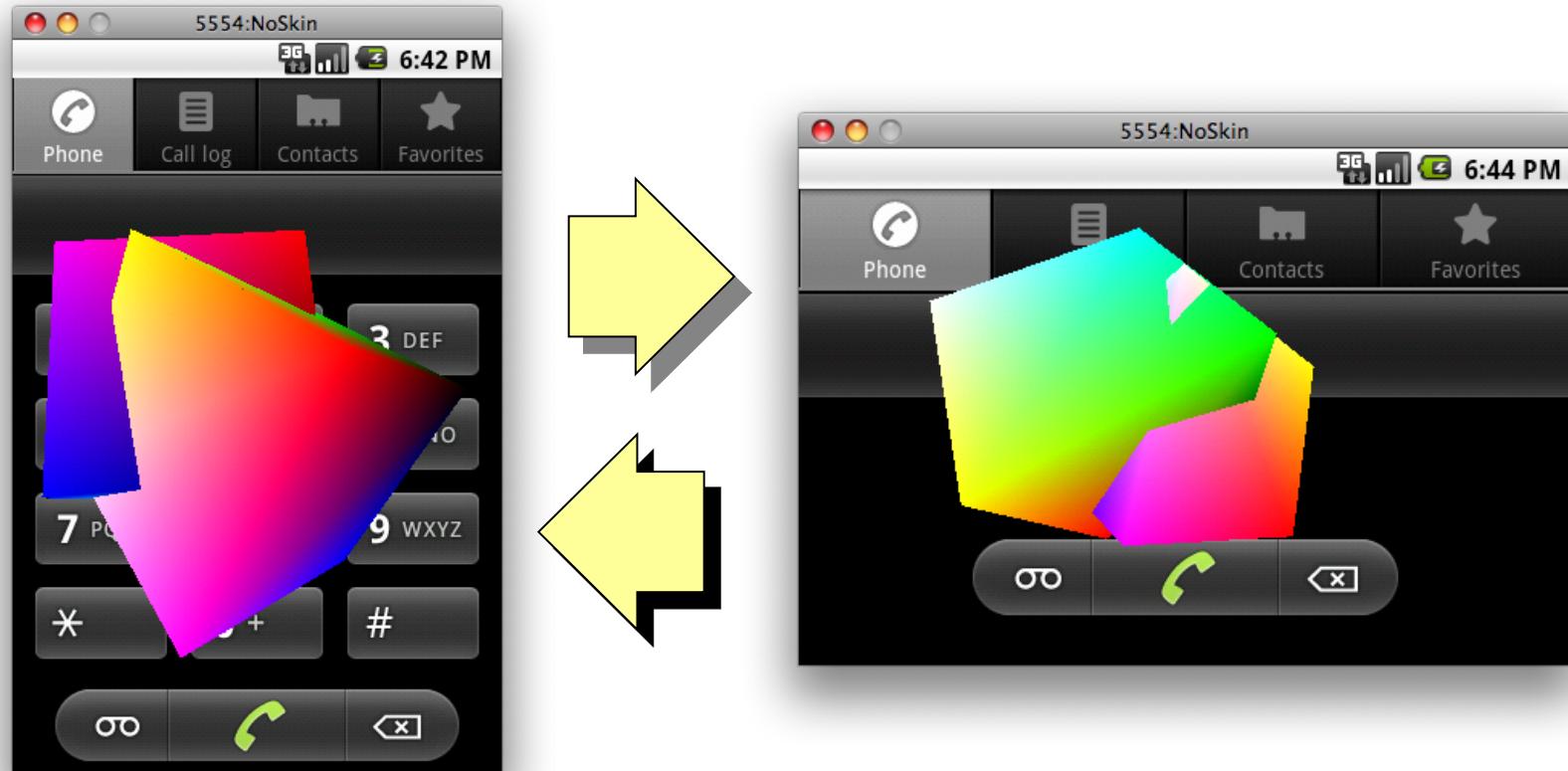
- 3279360 Byte → 約3.1MB
- GPU メモリが 約3.2MB のため、すでに占有状態

※しかし、通常の運用を考えればこれでも充分（メモリ領域はZ-Order割り当てと同じく「いま見えている範囲」のレイヤにのみ割り当てられため）



## 障害例2：SurfaceFlingerのメモリ不足

- さらに、Androidには端末の向き（Orientation）という要素もあります。

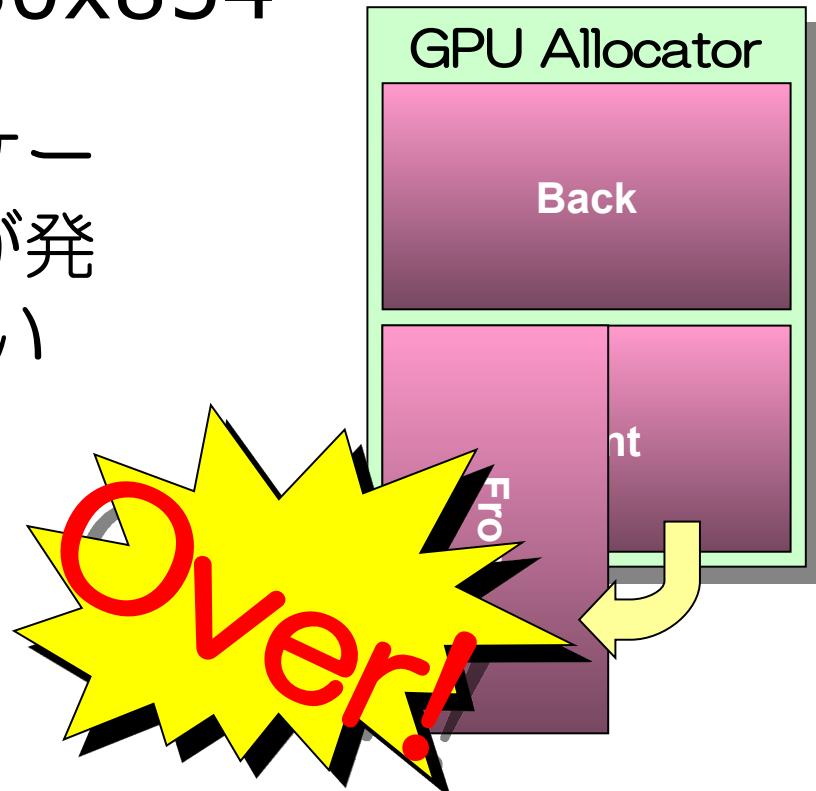


## 障害例2：SurfaceFlingerのメモリ不足

- Androidは、端末の向きにあわせ描画領域（描画メモリ）を伸縮する機能も持ち合わせています。
- しかし、それが特定のタイミングで実行されると…

## 障害例2：SurfaceFlingerのメモリ不足

- SurfaceFlingerは描画メモリをリサイズしようとしますが…
  - ▶ 例) 854x480 → 480x854
- しかし、リサイズのアロケーションは失敗し、エラーが発生します。（※ 発生しないこともあります。）
- 最悪の場合はシステムがハングアップ！！



## 障害例2：SurfaceFlingerのメモリ不足

- どんなタイミングでリサイズが発生するか…
  - ▶ 端末の向きを変えたとき
  - ▶ アプリが停止状態から復帰したとき
  - ▶ etc...
- 機種によりリサイズによるエラーが発生するタイミングや状況は異なる

## 障害例2：SurfaceFlingerのメモリ不足

- 現象が発生しない機種も存在します。  
(Androidバージョンの違いによる可能性もあり)
- 現象を発生させない確実な方法は不明ですが、傾向的に描画メモリに余裕があると発生頻度が低くなるように思われます。

## 障害例2：SurfaceFlingerのメモリ不足

- 例に挙げた Xperia<sup>TM</sup>と同じ SnapDragon 搭載の類似端末で GPU メモリのサイズが異なっているケースもありました。

- ▶ Xperia 約3.2MB
- ▶ 類似端末 約4MB

- ハードウェアの違い？
- でも、GPUはSnapDragonに内蔵…
- OSポーティング実装の違い？

# 障害例2：SurfaceFlingerのメモリ不足

## ■ 本現象の回避策は？

- ▶ OS実装まで手を入れることができるなら、本現象は回避可能かもしれません。
- ▶ アプリケーションだけで回避するには、現象の発生タイミングでレイヤを開放するなどの対処が考えられます。（機種毎に対応が必要）

## ■ 結局のところ…

- ▶ アプリケーション開発であっても、ハードウェア、プラットフォーム特性を知っていないと回避・解決できない問題点が存在します

# まとめ

# まとめ (1)

- **SurfaceView・  
GLSurfaceView** は特殊
- 要注意！！

# まとめ (2)

- Androidフレームワークは想定しない使い方をしても動いてしまう
- 後で障害原因となるかも…
- プロトタイプで要確認！！

# まとめ (3)

- ハードウェア、プラットフォーム特性もしらないと解決できない問題があります
- 製品として完成させるためには、アプリだけではなく、OS のことも知る必要あり



以上、  
ご清聴ありがとうございました。

