UE4のスレッドの流れと Input Latency改善の仕組み

Epic Games Japan / Support Manager
Nori Shinoyama (篠山範明)

INPUT LATENCY

- UE4のDefault動作は、並列性を最大限活かす様につくられています
  しかしその副作用として、逆にLatencyが長くなる可能性があります

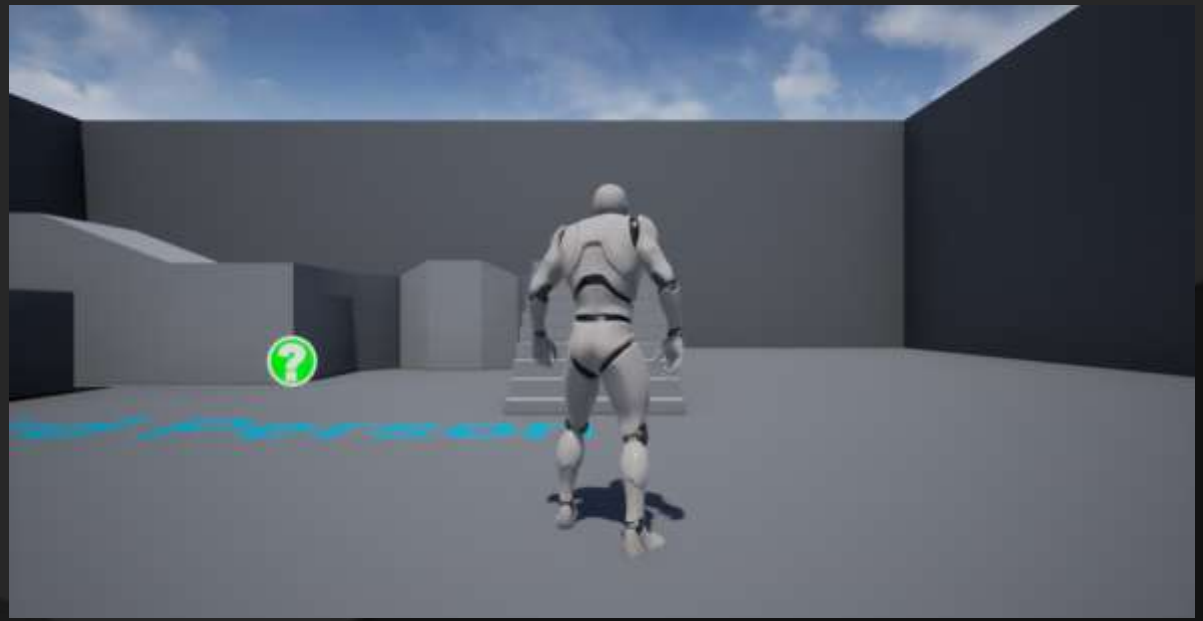
- 30fpsのゲームで特に顕著に
  - 30fpsのゲームでは130ms以上のLatency
  - 60fpsでは軽減されるも、格闘ゲームなど入力がシビアなゲームでは依然問題に

- 4.19からInput Latencyの改善が入りました。
  - DefaultではOffです。Onにすることで新たな制約が生まれます
  - 対応しているプラットフォームとしていないプラットフォームがあります、
    各プラットフォームのドキュメントを参考にしてください

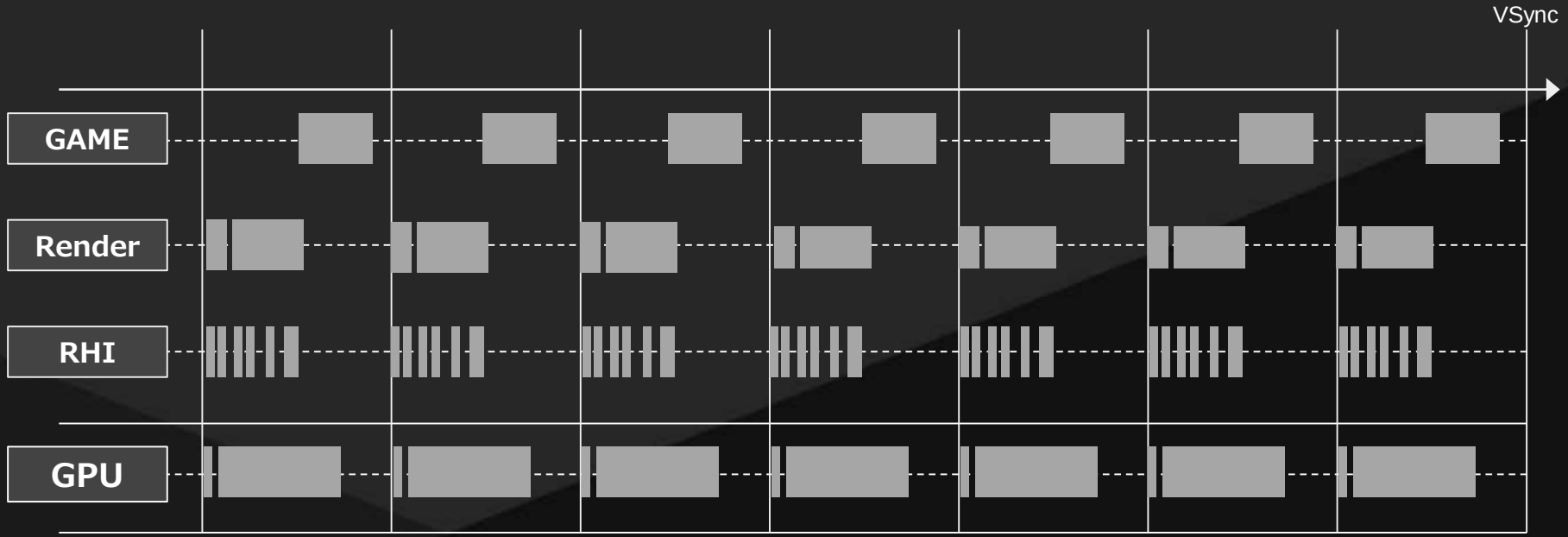
Input Latencyの話をする前に。。。
どのThreadがどの様に毎フレーム走っているのかをおさらいします

免責事項

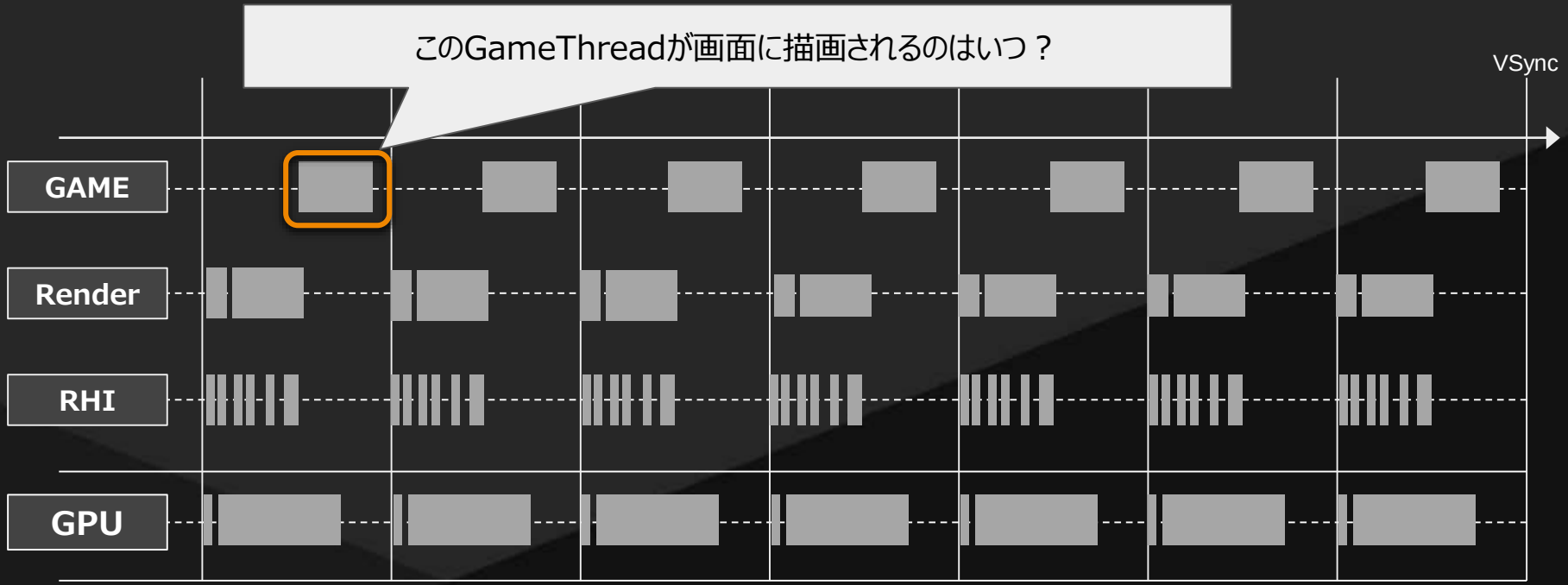
これから各スレッドの動きと同期について説明していきますが、
厳密には各プラットフォームによって動きが違う可能性があります
例えば。。。
　RHI Threadがあるかないか
　フリップ命令を発行するのがCPUかGPUか
などなど
各プラットフォームでの動作の詳細を確認したい場合、
必ずそのプラットフォームが出しているCPU Profilerを参照してください



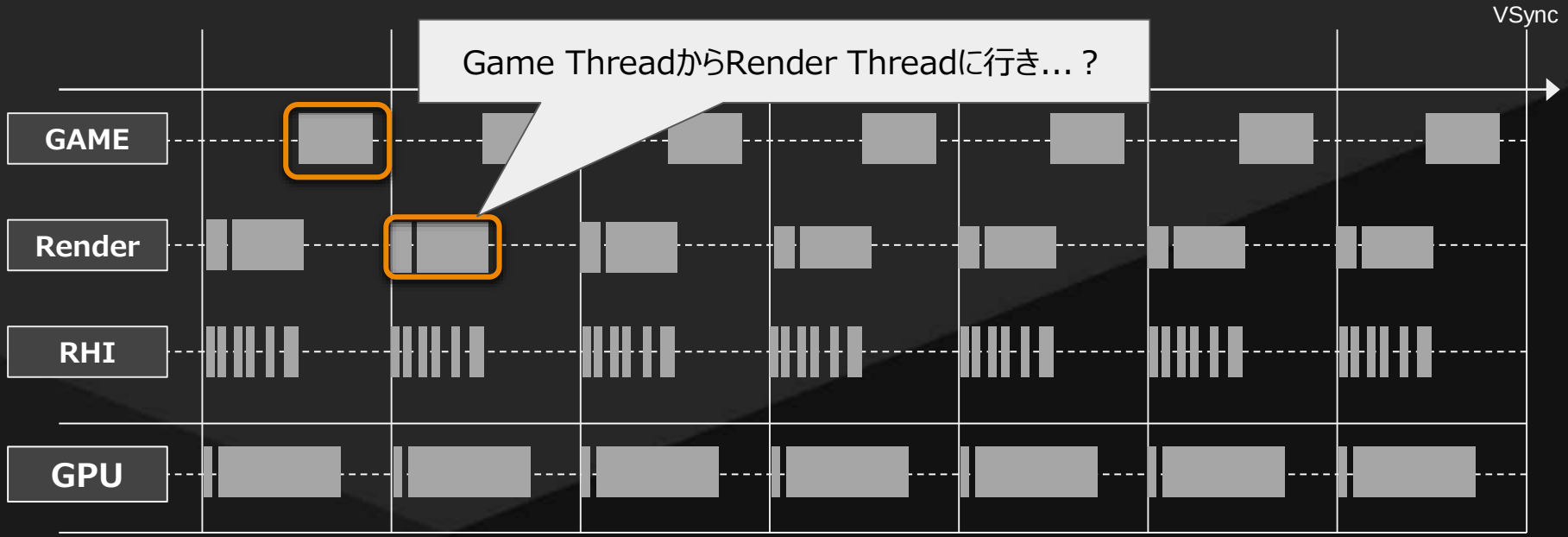
Third Person TemplateでInput LatencyのWorst Caseを見てみます



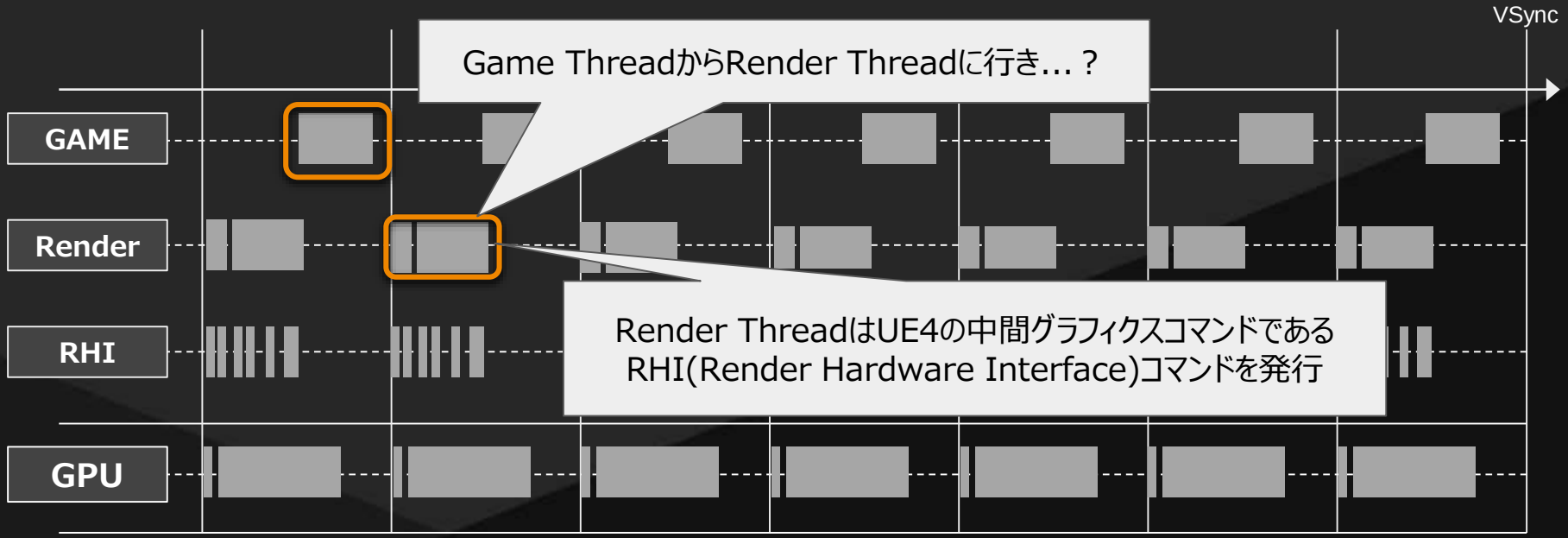
Profilerで見る各スレッドの処理タイムラインの例
超スーパーウルトラ簡略図


このGameThreadが画面に描画されるのはいつ？
VSync
GAME
Render
RHI
GPU

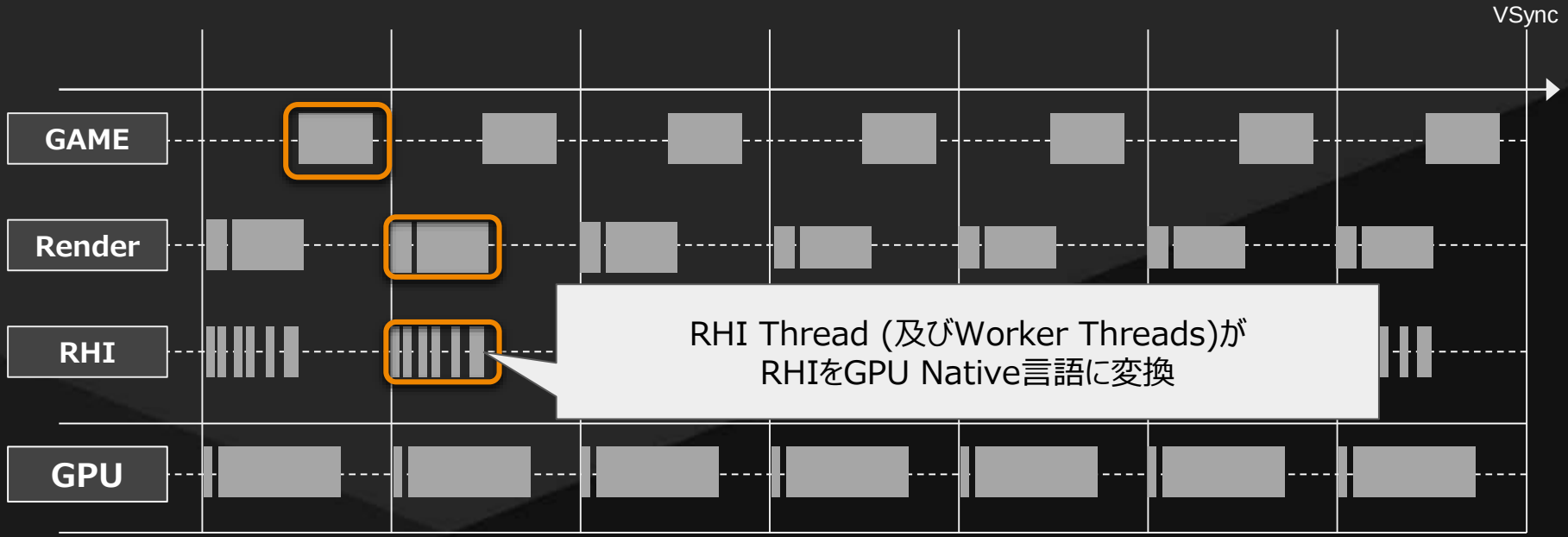
最初に誤った例を説明します



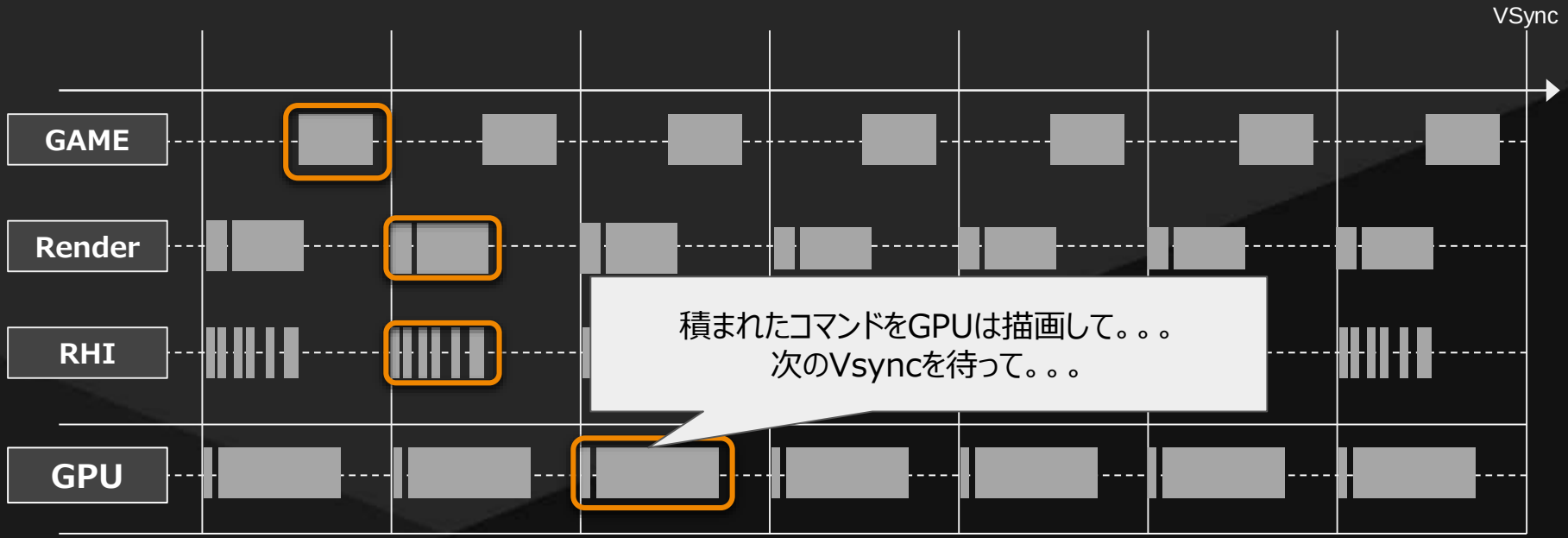
※誤った例です


VSync
Game ThreadからRender Threadに行き…？
GAME
Render
RHI
GPU
Render ThreadはUE4の中間グラフィクスコマンドである
RHI(Render Hardware Interface)コマンドを発行

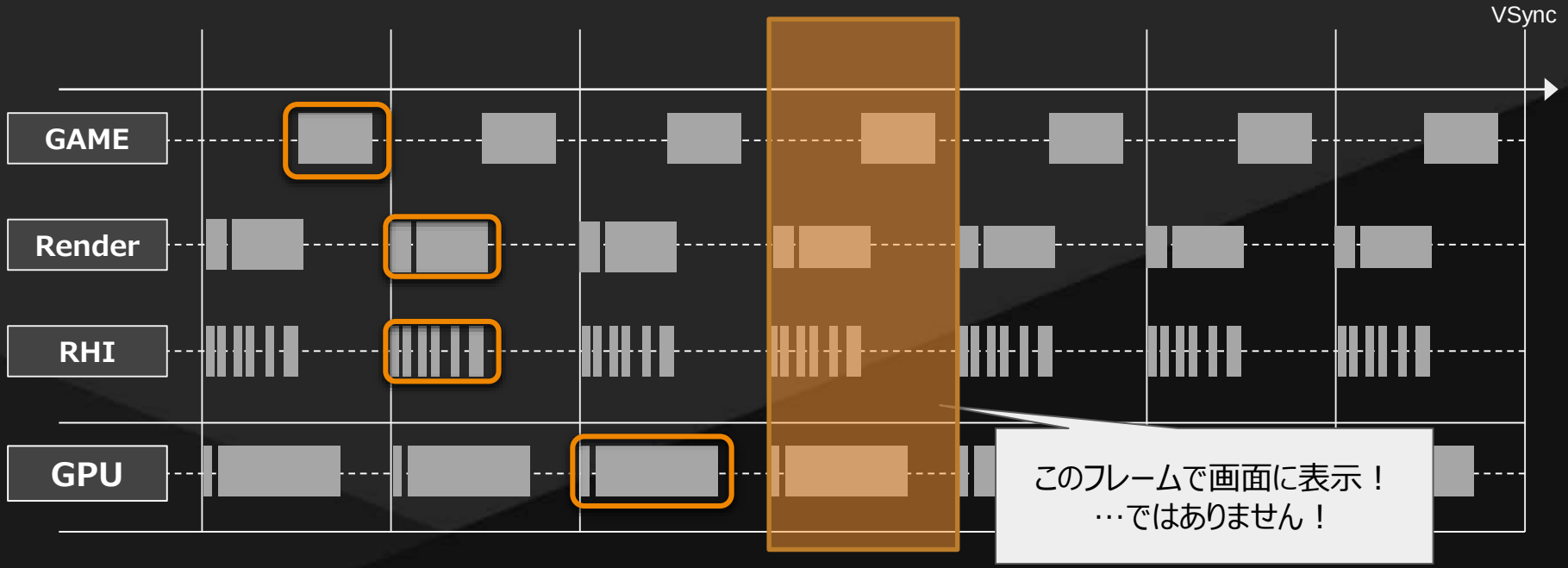
※誤った例です


VSync
GAME
Render
RHI
GPU
RHI Thread (及びWorker Threads)が
RHIをGPU Native言語に変換

※誤った例です

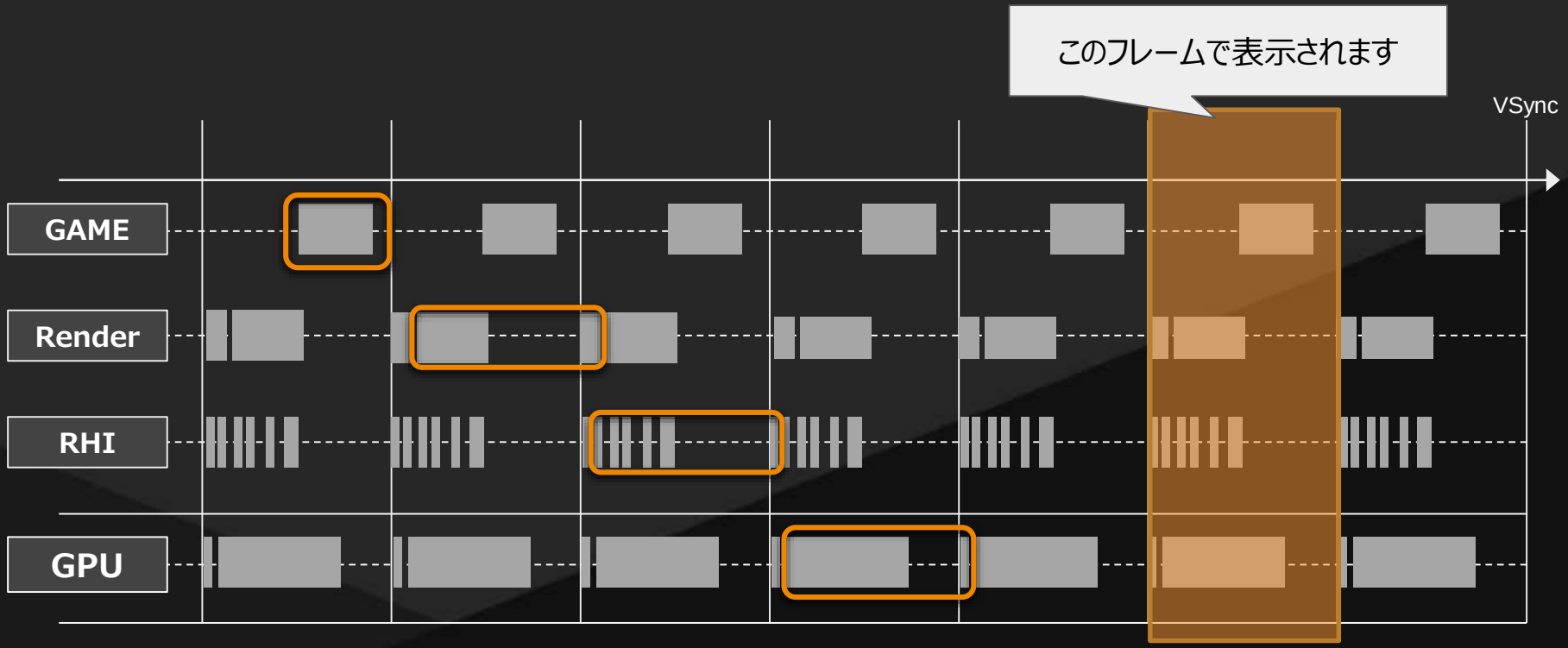


※誤った例です

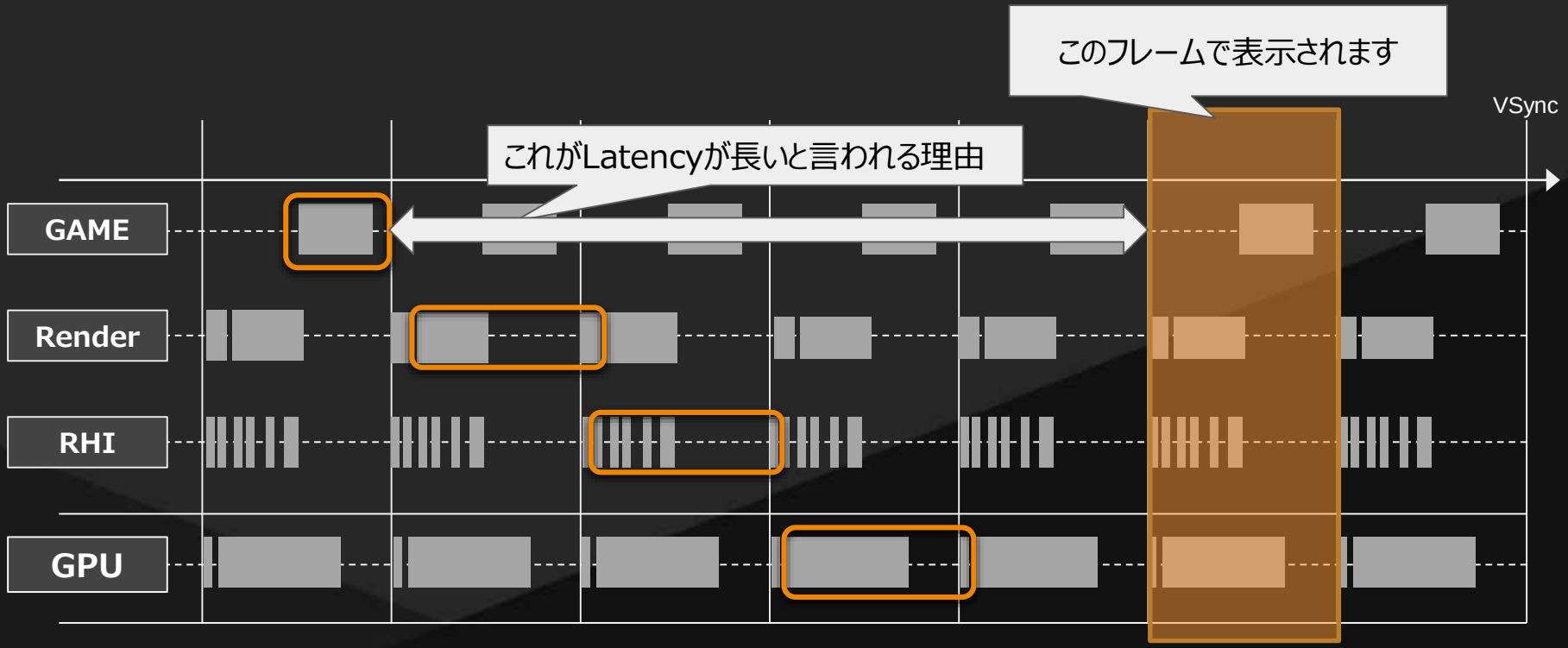


※誤った例です

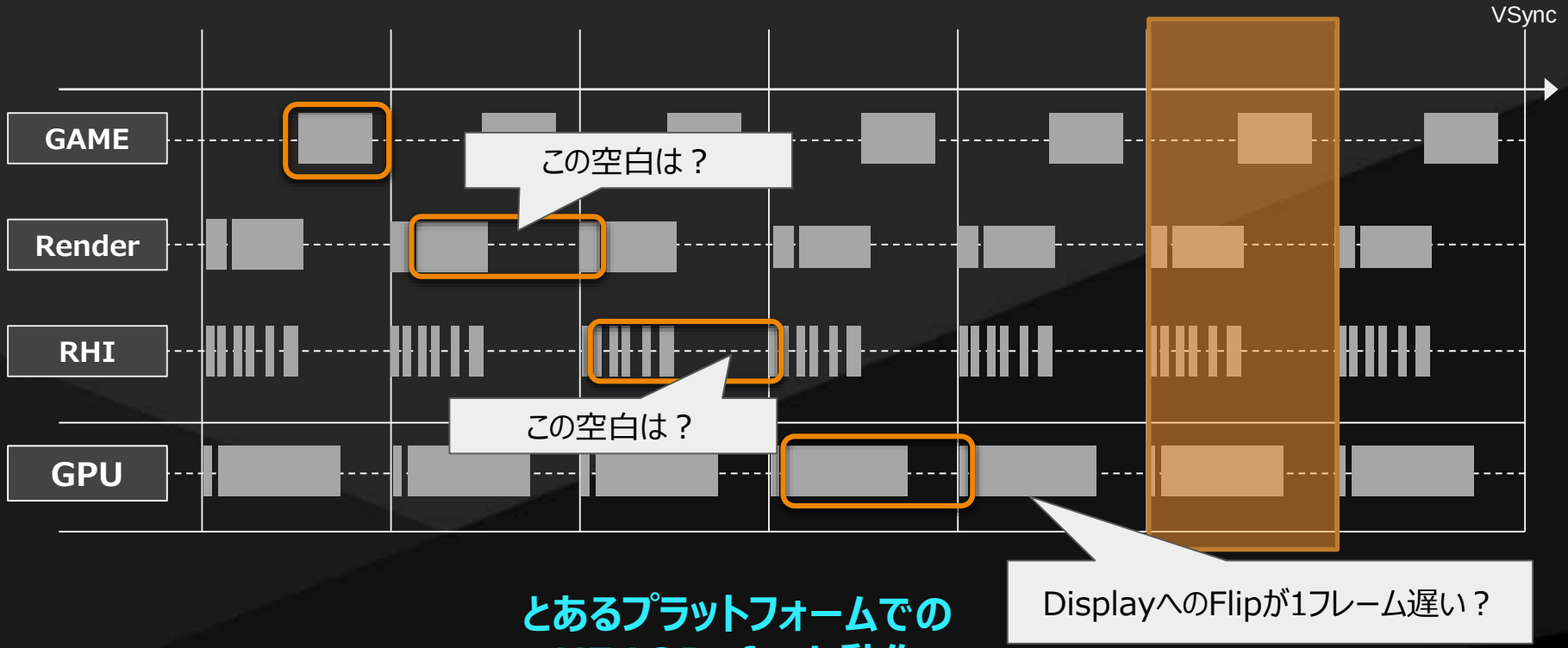
正しくはこんな感じです



とあるプラットフォームでの
UE4のDefault動作



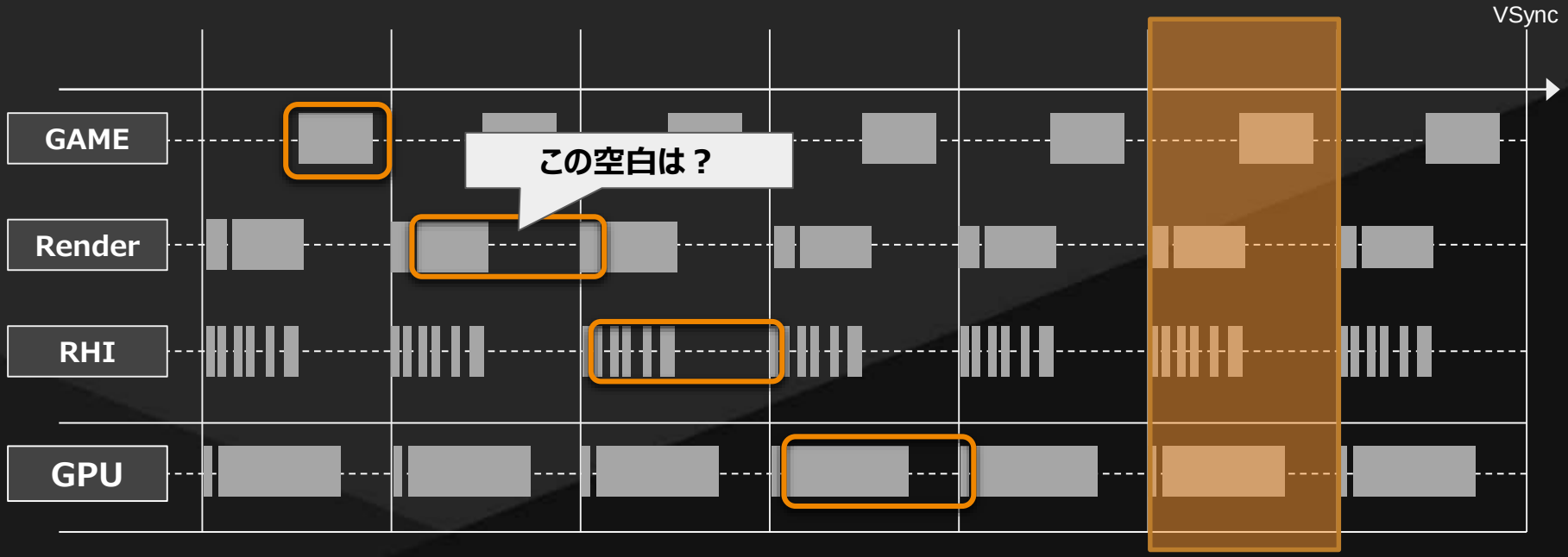
とあるプラットフォームでの
UE4のDefault動作



とあるプラットフォームでの
UE4のDefault動作

各空白の原因を知るために
各スレッドの役割と動きをおさらいしていきます

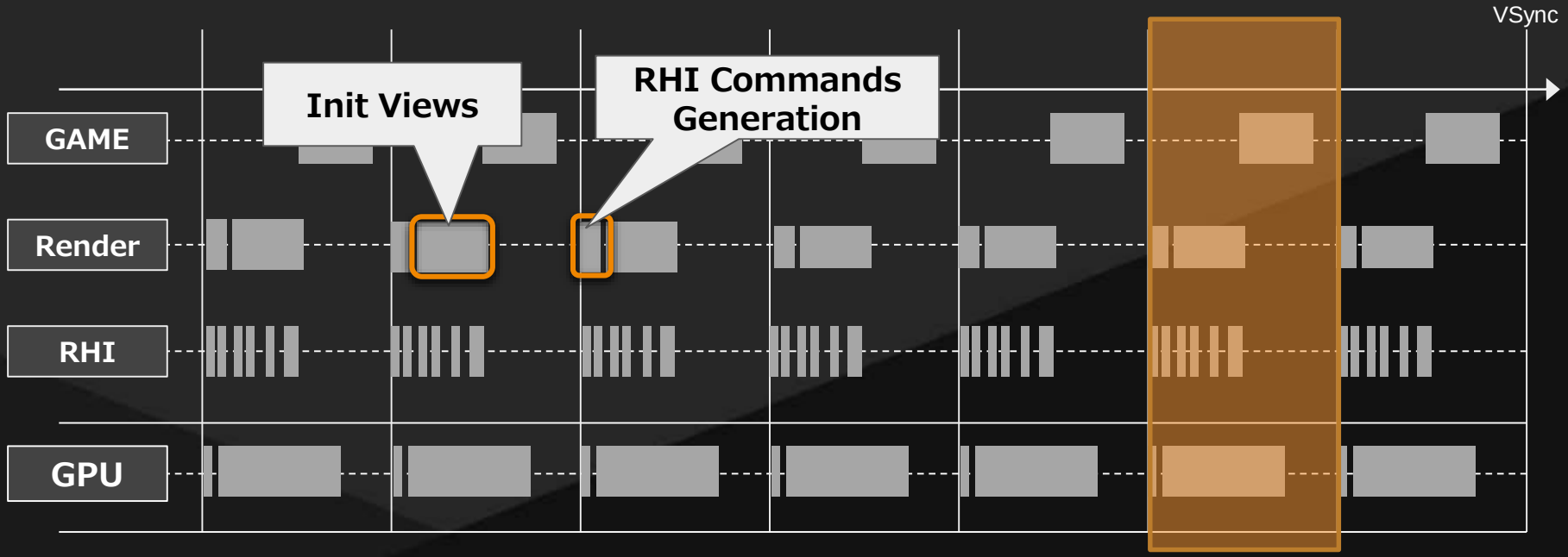
Render Thread

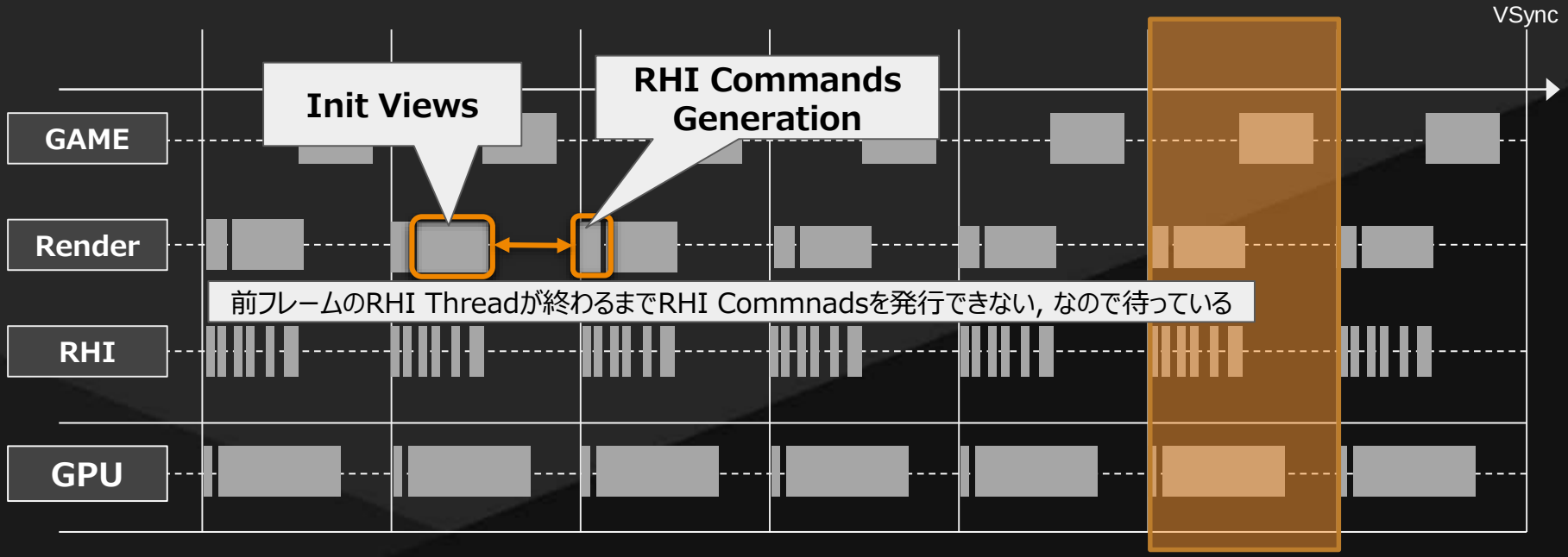

VSync
GAME
Render
RHI
GPU
この空白は？

Render Threadの動作を大きく２つに分けてみると。。。

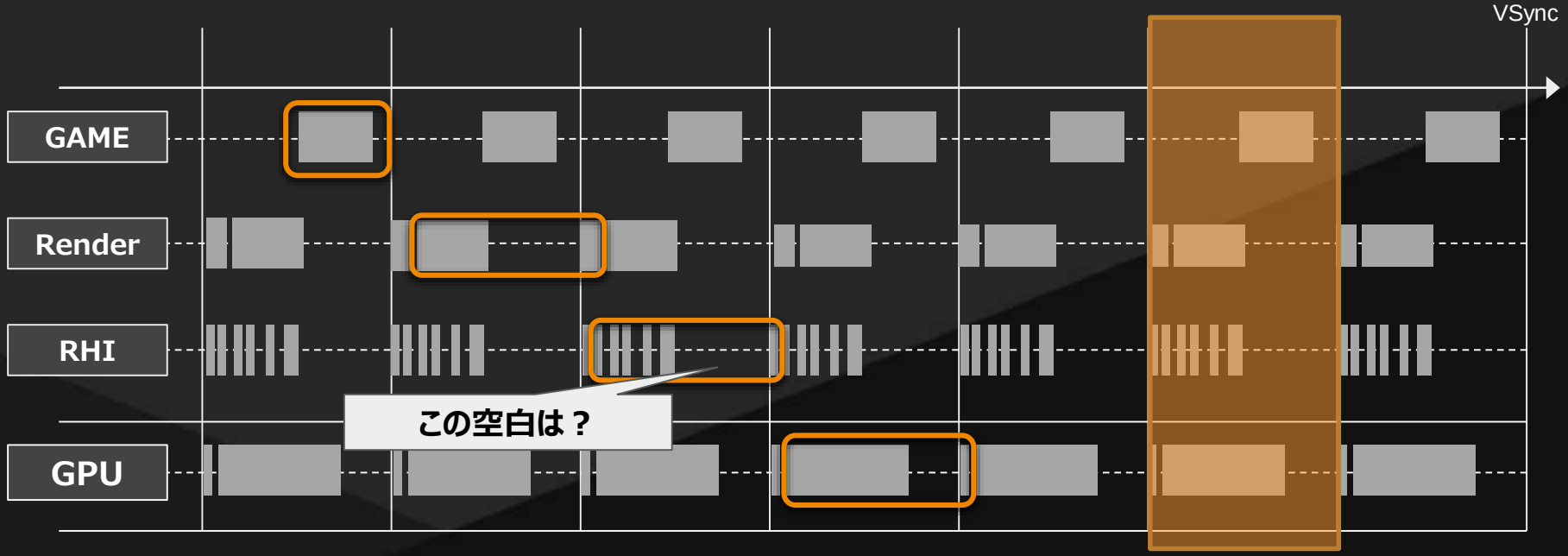
1. **Init Views**
   画面に映るオブジェクト、ShadowMapに描画されるオブジェクトの選別

2. **RHI Commandの発行**
   RHI Commandは、GPUのNativeCommandに変換するために
   RHI Thread(及びWorker Threads)らに送られる
   RHI Threadが前フレームの処理を終えるまで、RHI Commandを送れない


VSync
Init Views
RHI Commands Generation
GAME
Render
RHI
GPU


VSync
Init Views
RHI Commands Generation
GAME
Render
RHI
GPU
前フレームのRHI Threadが終わるまでRHI Commnadsを発行できない, なので待っている

RHI Thread



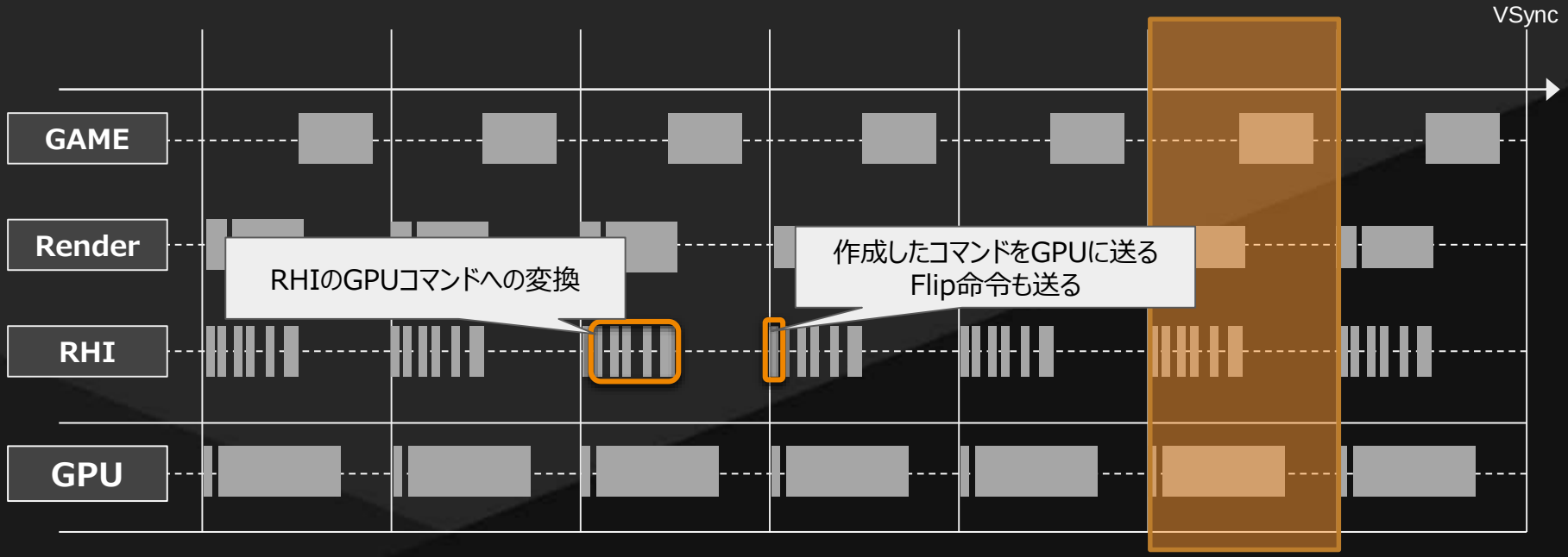
とあるプラットフォームでの
UE4のDefault動作

RHI Threadの役割

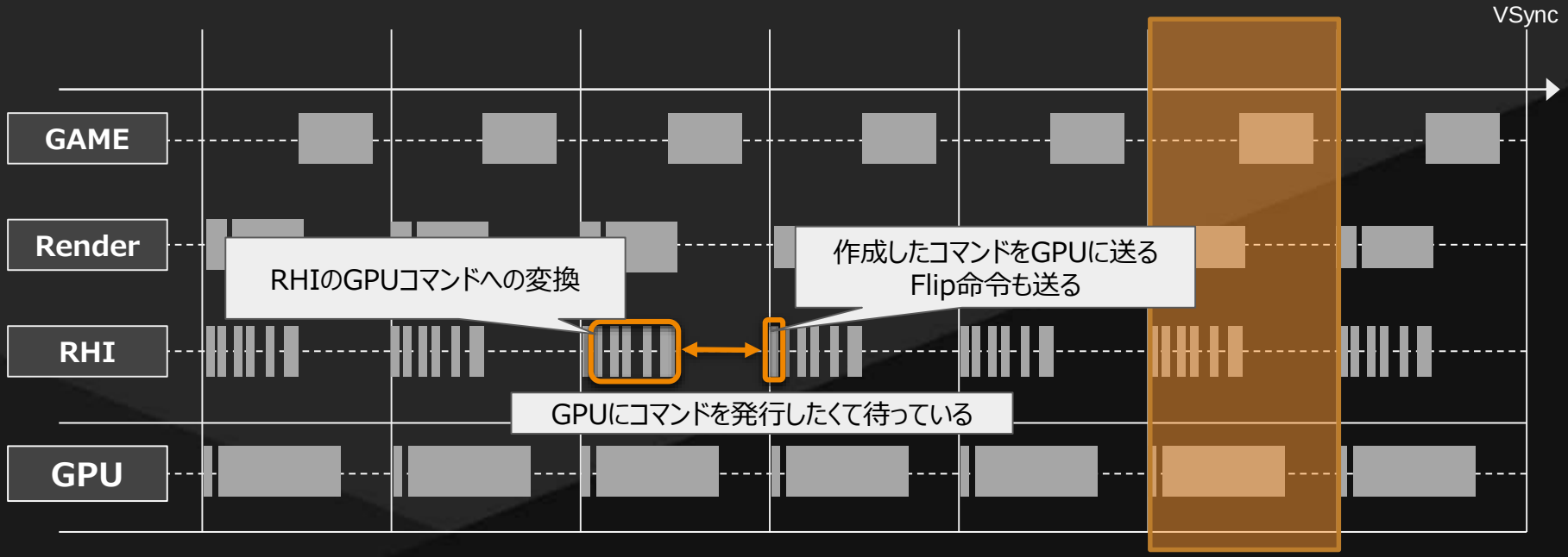
1. RenderThreadから送られてくるRHIをGPU Commandに変換する

2. GPU CommandをGPUに送る(Submitする)

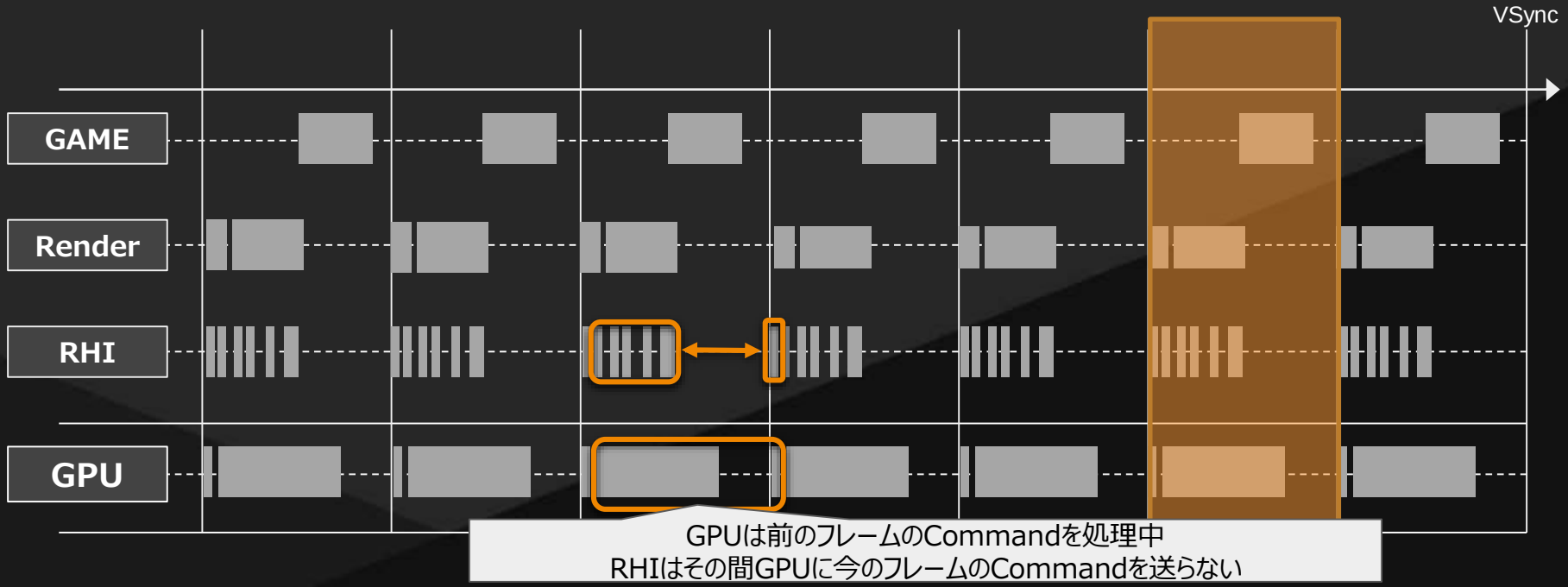
3. Flip命令をGPUに送る
   (Flip命令: ディスプレイに現在描画したフレームバッファを表示させる命令)



とあるプラットフォームでの
UE4のDefault動作

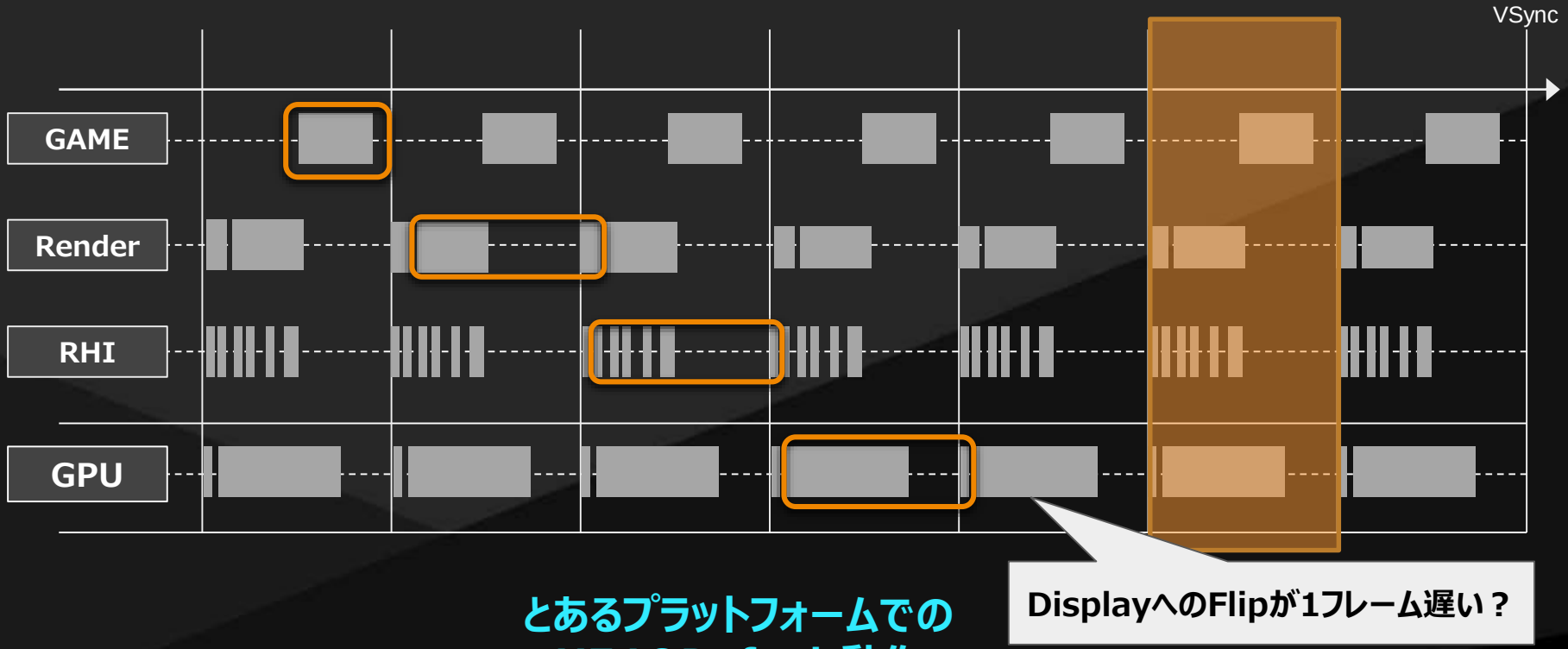


とあるプラットフォームでの
UE4のDefault動作



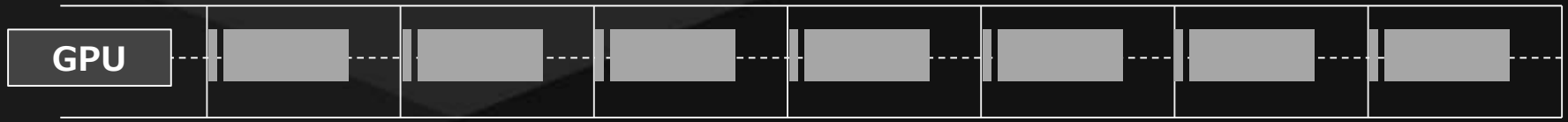
とあるプラットフォームでの UE4のDefault動作

GPU & Flip

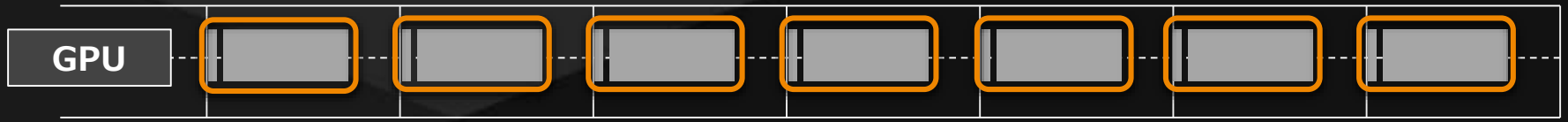


とあるプラットフォームでの
UE4のDefault動作

GPUの処理負荷だけ見てみると。。。

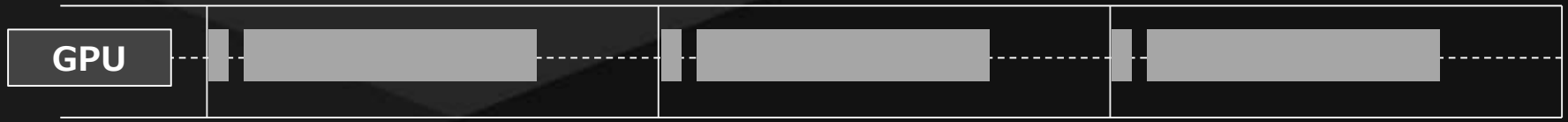


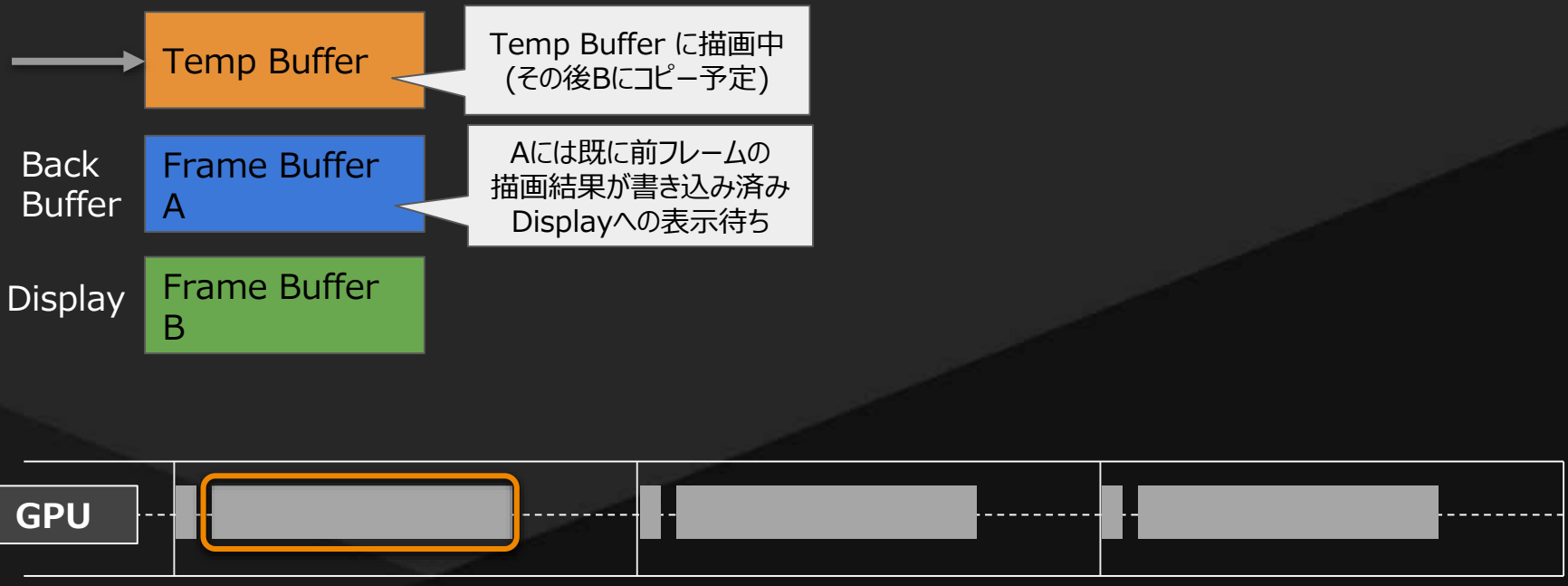
この様な単位で1フレーム毎に描画しているように見えますが。。。
実際は違います



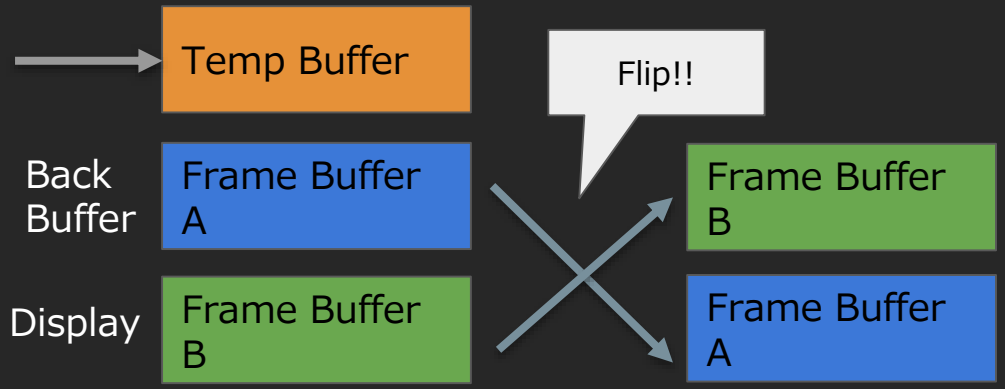
本当はこのように描画されています

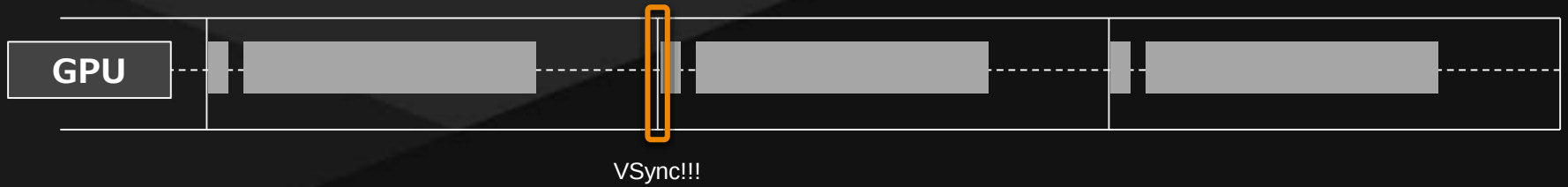
拡大して見ていきます

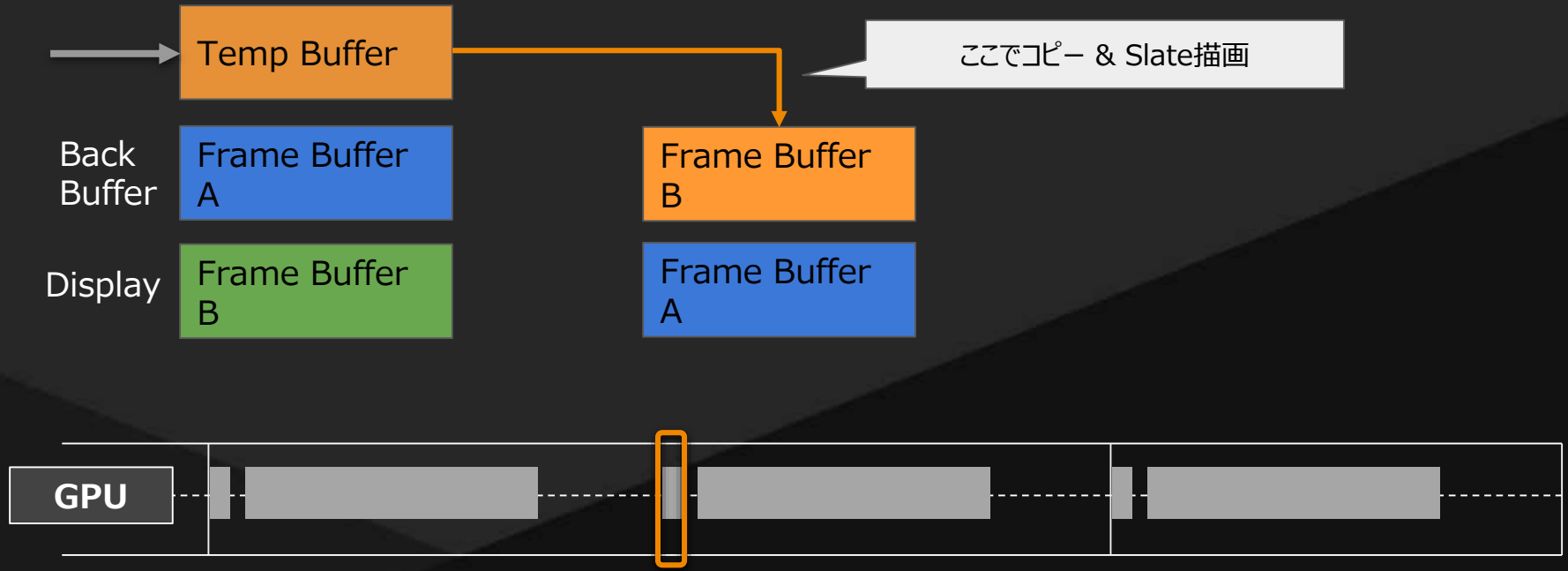


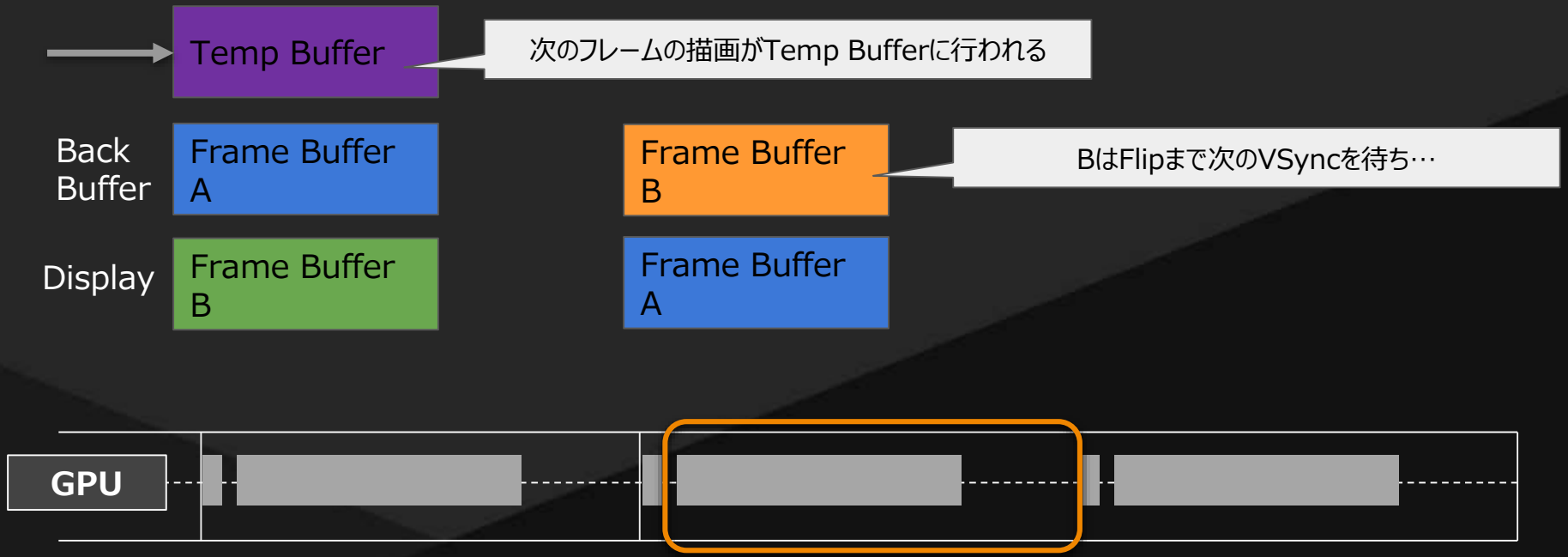

Temp Buffer
Temp Buffer に描画中
(その後Bにコピー予定)
Back
Buffer
Frame Buffer
A
Aには既に前フレームの
描画結果が書き込み済み
Displayへの表示待ち
Display
Frame Buffer
B
GPU

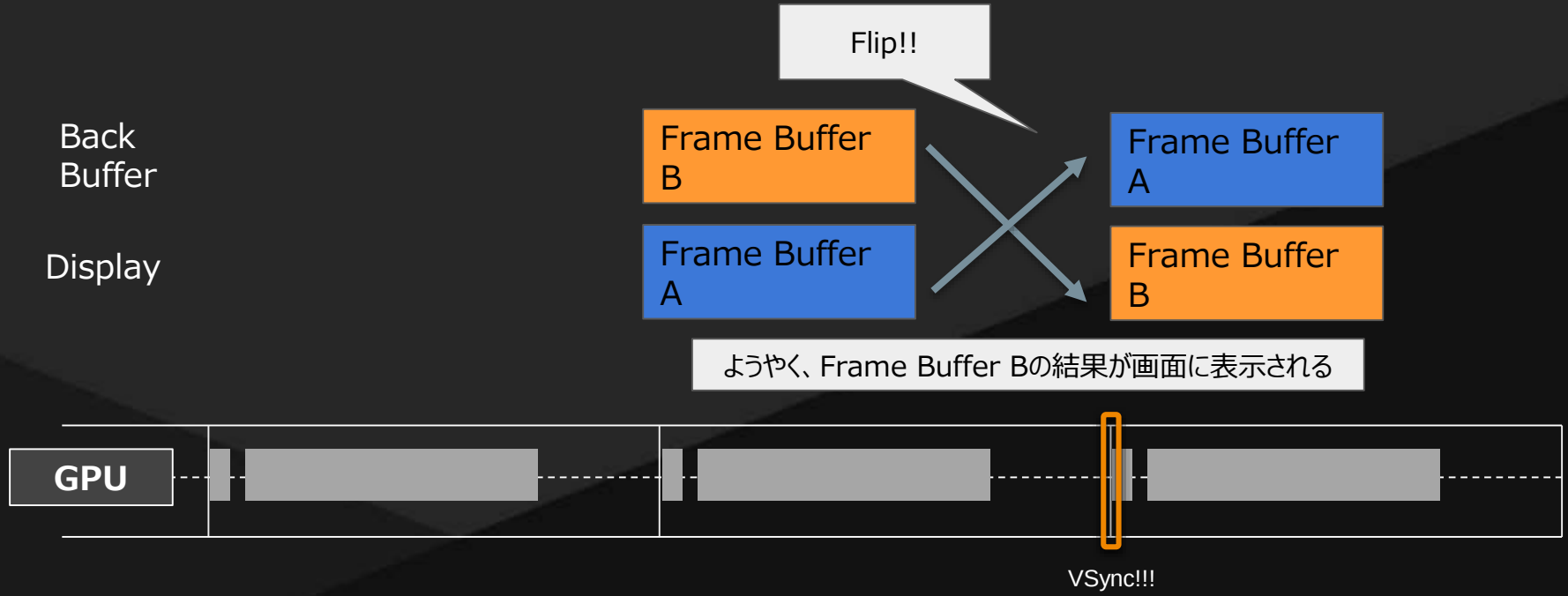

Temp Buffer
Back Buffer
Frame Buffer A
Display
Frame Buffer B
Dsiplayに表示されているのでコピーできない…
待つ
GPU

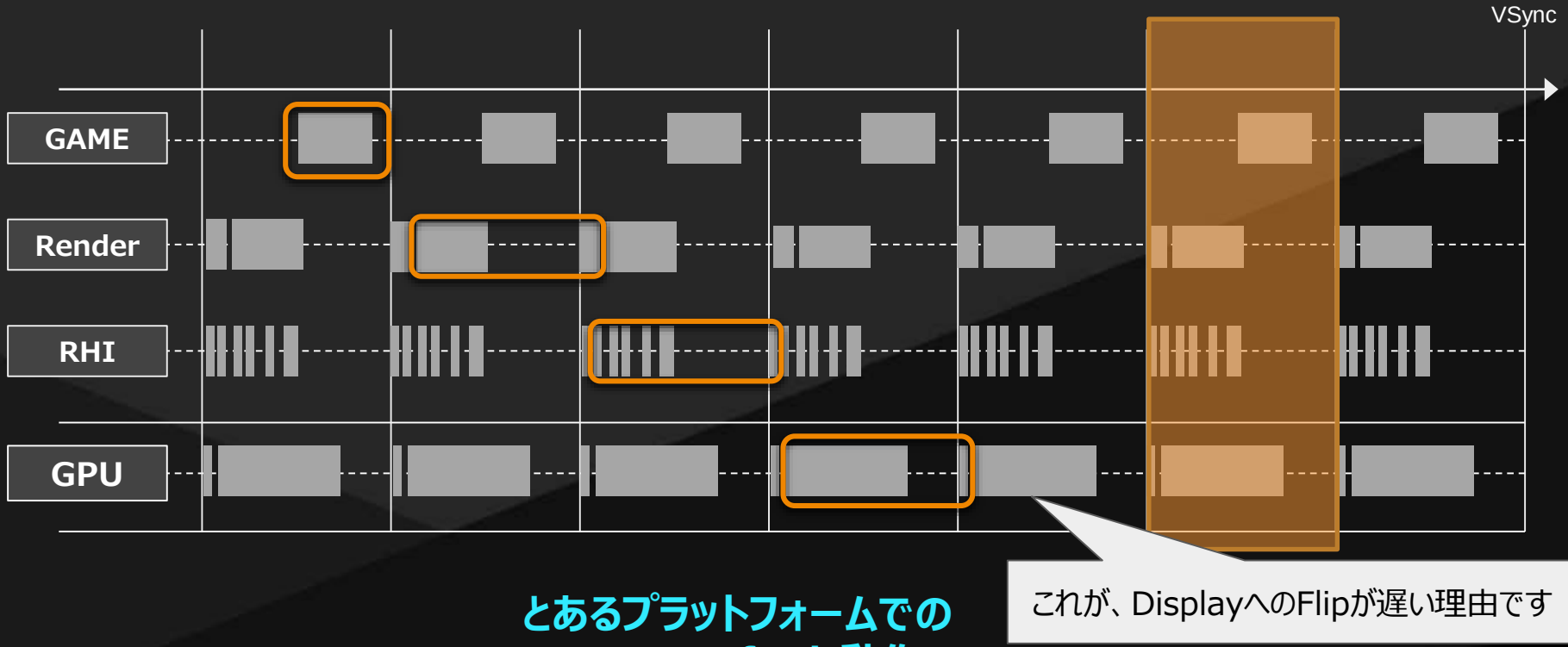

Temp Buffer
Flip!!
Back Buffer
Frame Buffer A
Frame Buffer B
Display
Frame Buffer B
Frame Buffer A


GPU
VSync!!!

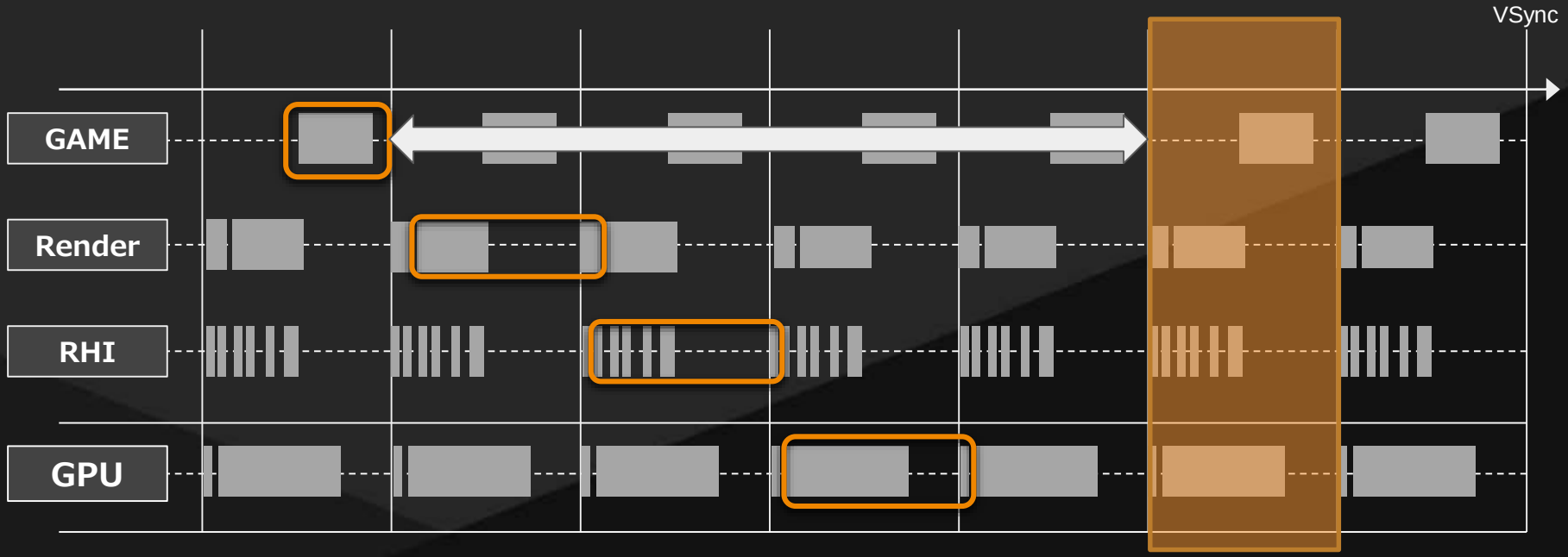

Temp Buffer
ここでコピー & Slate描画
Back Buffer
Frame Buffer A
Frame Buffer B
Display
Frame Buffer B
Frame Buffer A
GPU


Temp Buffer
次のフレームの描画がTemp Bufferに行われる
Back Buffer
Frame Buffer A
Frame Buffer B
BはFlipまで次のVSyncを待ち…
Display
Frame Buffer B
Frame Buffer A
GPU


Back
Buffer
Display
Flip!!
Frame Buffer
B
Frame Buffer
A
Frame Buffer
A
Frame Buffer
B
ようやく、Frame Buffer Bの結果が画面に表示される
GPU
VSync!!!



とあるプラットフォームでの
UE4のDefault動作



長々と説明しましたが、これが1フレームの一連の処理の流れです

Render/RHI/GPUの動作を見ながら、
各スレッドが待つ理由を確認していきました

UE4はどのスレッドも”行けるところまでなるべく早く動こう”とする
並列性が確保できるがその副作用として、

RenderはRHIを
RHIはGPUを
GPUはFlipを

と待ち、レイテンシーが増加する傾向にある

このレイテンシーを改善するための設定が4.19からできました。

Input Latency Setting

- **r.GTSyncType**
  - Determines how the game thread syncs with the render thread, RHI thread and GPU
  - 0: Sync the game thread with the render thread (default)
  - 1: Sync the game thread with the RHI thread
  - **2: Sync the game thread with the GPU swap chain flip**
- **rhi.SyncInterval**
  - Determines the frequency of VSyncs in supported RHIs.
  - 0 - Unlocked
  - 1 - 60 Hz (16.66 ms)
  - 2 - 30 Hz (33.33 ms)
- **rhi.SyncSlackMS**
  - Increases input latency by this many milliseconds, to help performance (trade-off tunable). Gamethread will be kicked off this many milliseconds before the vsync
  - Default: 10ms

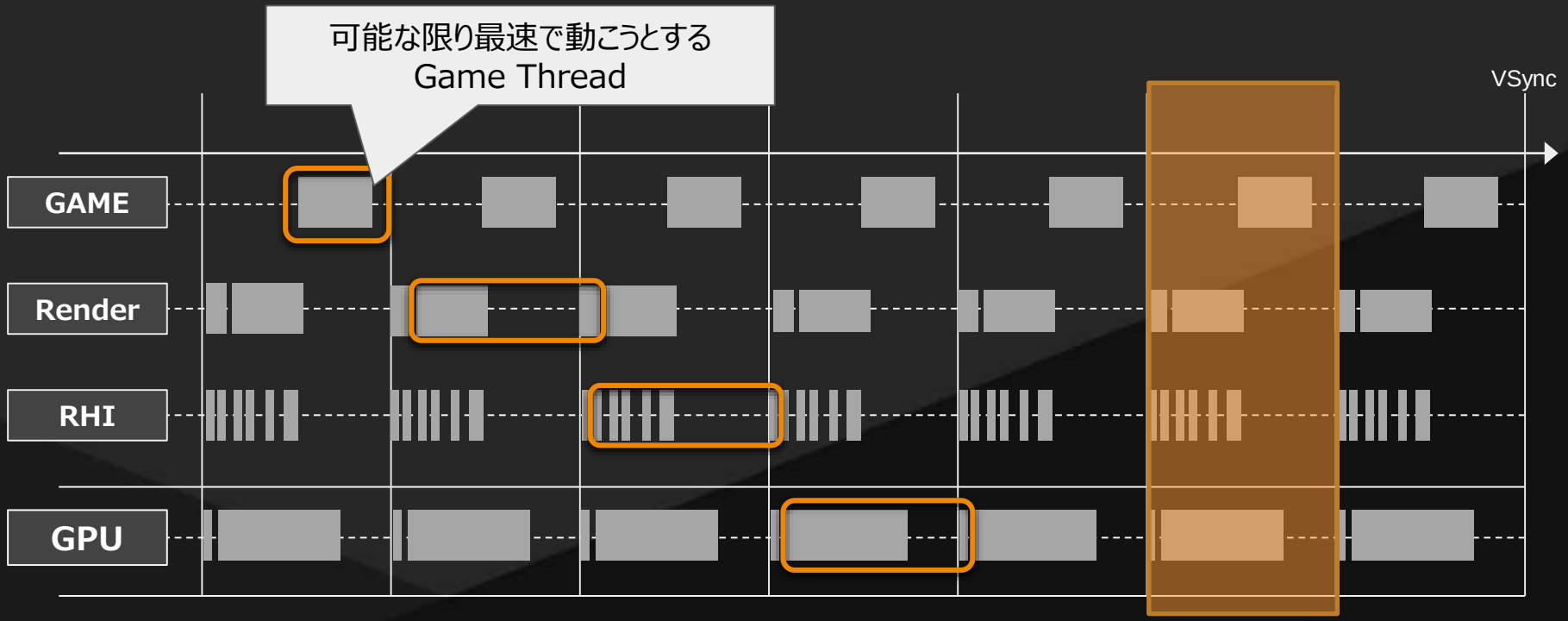
公式ドキュメントが用意されているので、詳しくはそちらを参照してください
https://docs.unrealengine.com/ja/Platforms/LowLatencyFrameSyncing/index.html

このInput Latencyの設定によって、
全ての起点であるGame Threadの開始を明示的に遅れらます
結果として、そうすることで各スレッドの待ち時間を減らしレイテンシーの低減を図ります

Input Latency改善のための基本設定はこの３つになるかと思います

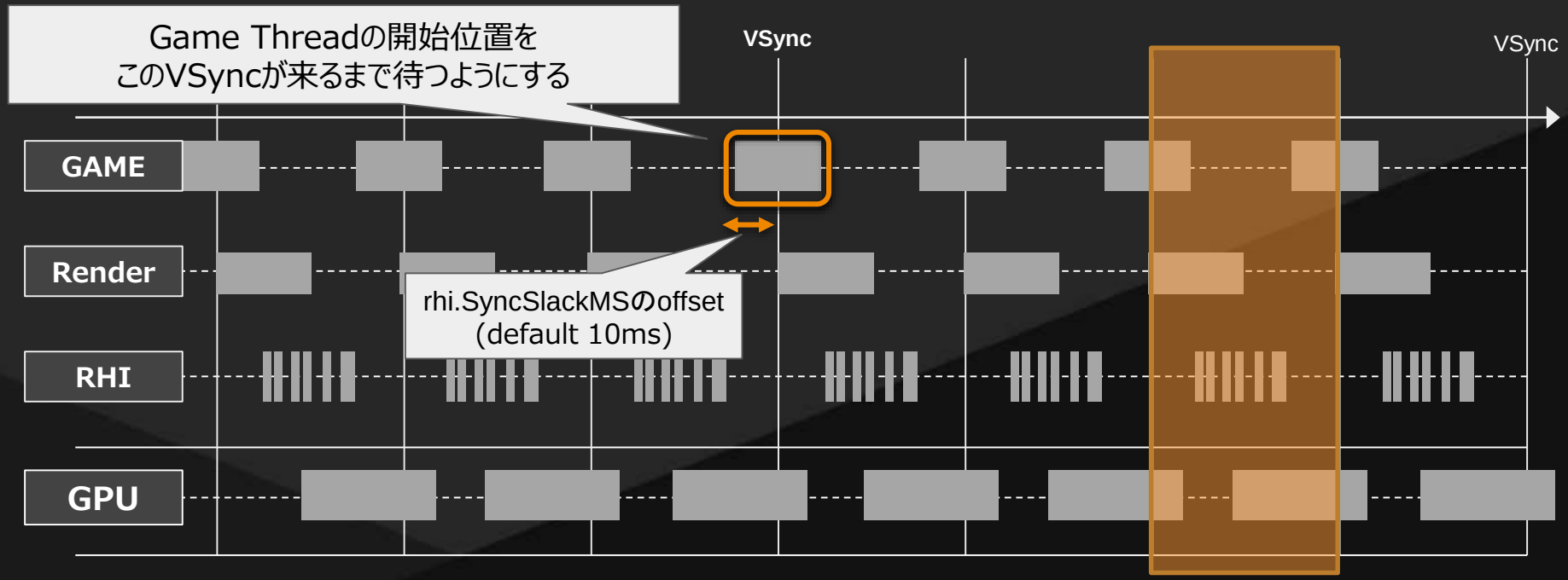
- rhi.SyncInterval 1/2 (何フレームのゲームを想定しているか？: 1:60fps, 2:30fps)
- r.GTSyncType　2 (Game Threadの開始をVSyncを待つ)
- rhi.SyncSlackMS 0 (Game Threadの開始をVsyncより何ms早くさせるかのオフセット)

Input Lateny改善の前後を比較してみます



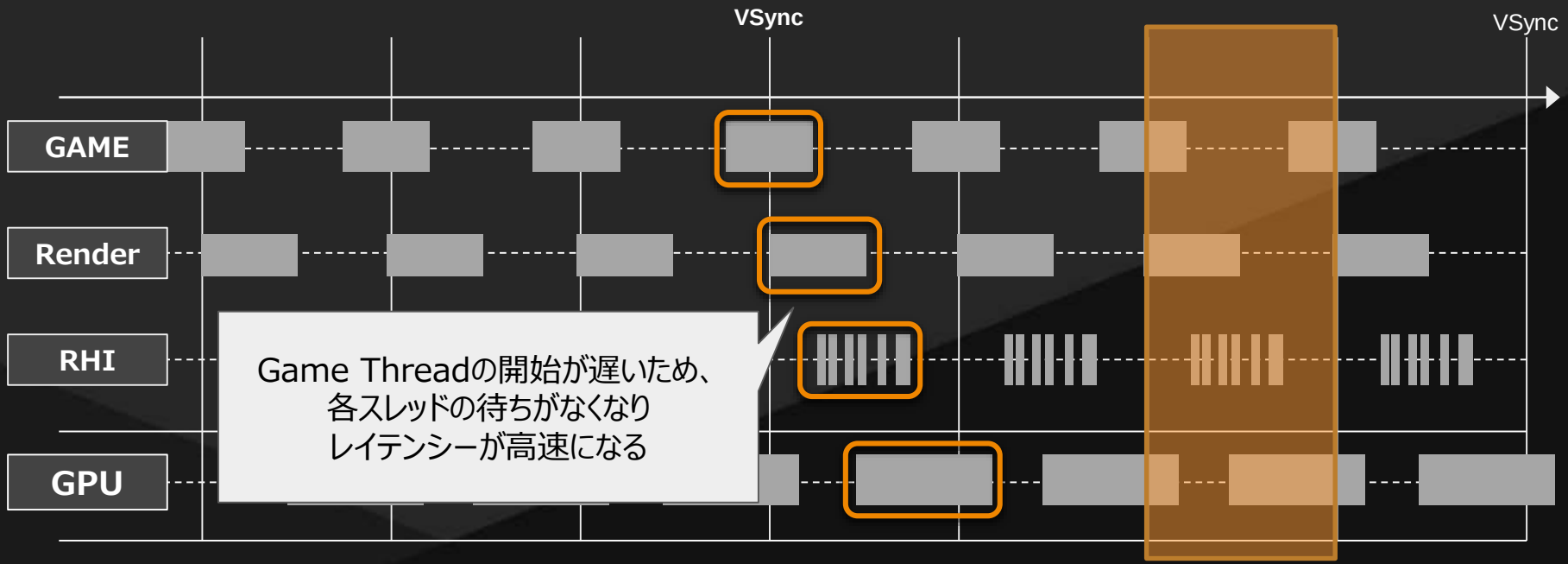
Input Latency改善前の動作

GTSyncType=2



Input Latency改善後の動作

GTSyncType=2



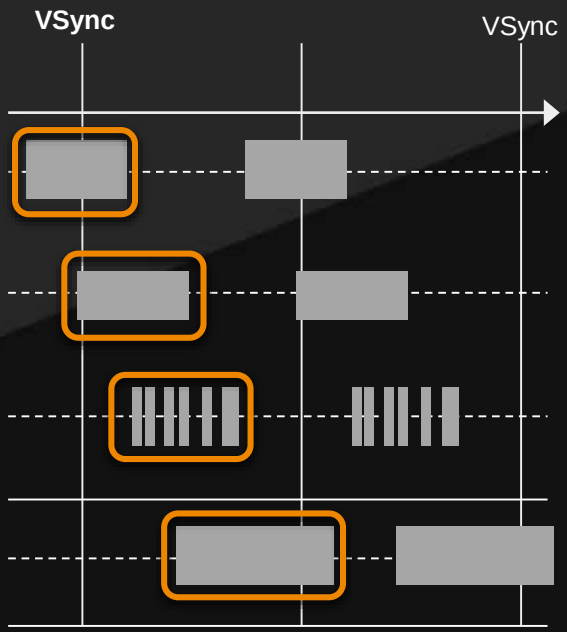
Input Latency改善後の動作

GTSyncType=2

しかし、このLatencyの改善には副作用があります

Latencyの改善により、同一フレームの各スレッドが並列で走ります
そのため各スレッドの依存関係が生じ、
各スレッドでの処理負荷が想定フレームに
収まっていても処理落ちしてしまうかもしれません

**使用する際は各プラットフォームのプロファイラなどを用いて、
各種スレッドの依存関係と処理負荷を確認するようにしてください**



Input Latency改善の副作用

UE4.19でのInput Latencyの改善

1. 使うと新たな制約が出て処理負荷をよりシビアにおさえないといけない。
2. r.GTSyncTypeで動的変更可能なので、カットシーンなどではオフにできる。
3. rhi.SyncSlackMSでGameThreadの開始位置を変更可能。
   デフォルトではVSyncの10ms前からGameThreadがスタートする

ありがとうございました！