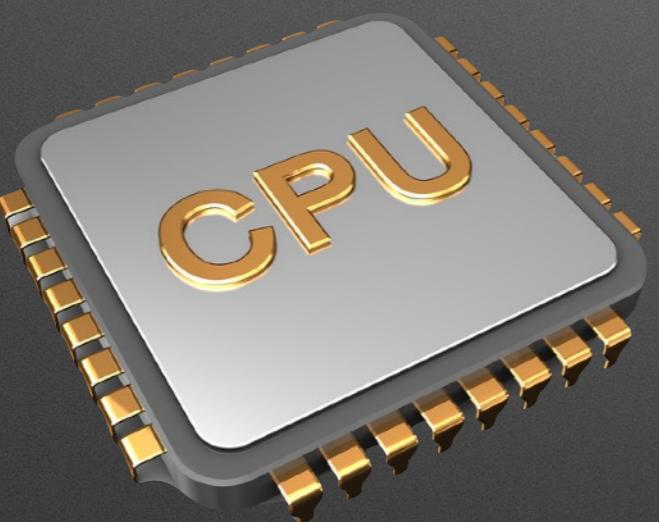


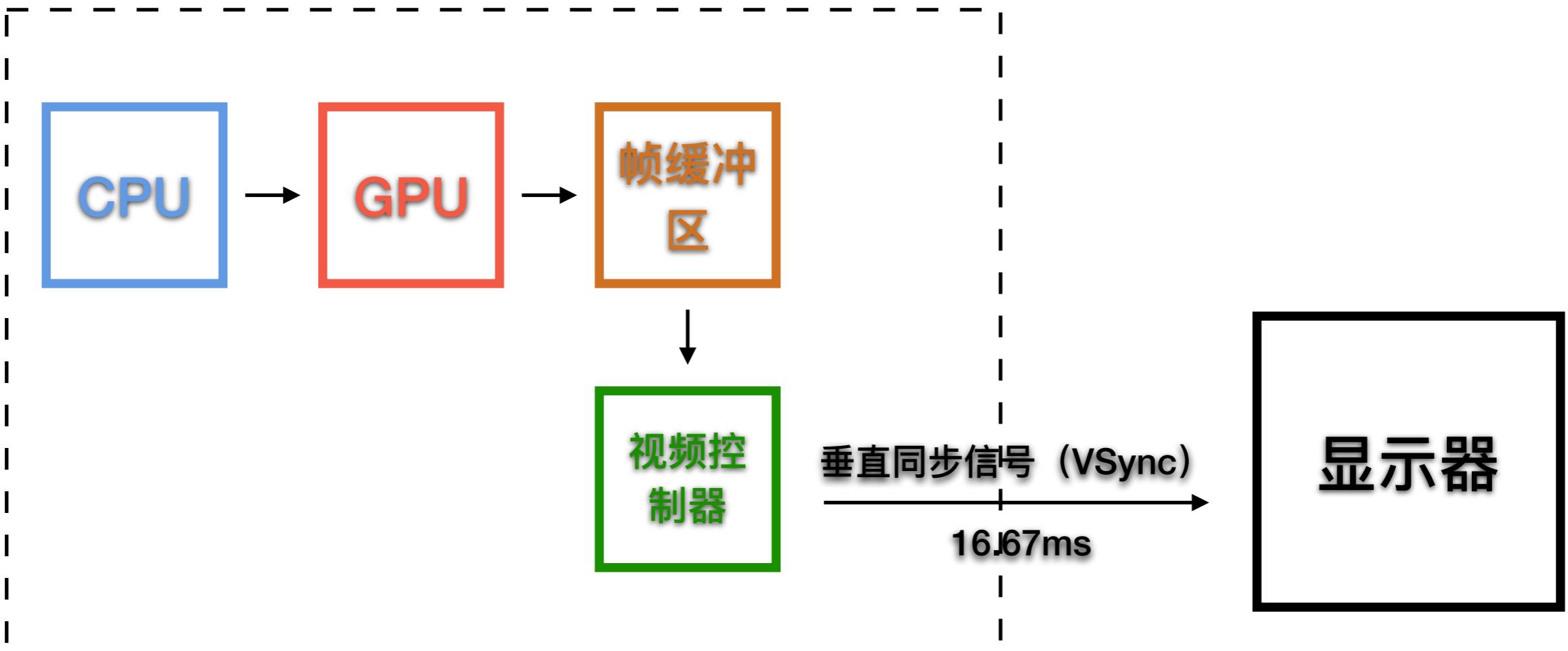
# 原理篇

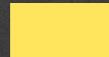
# 原理篇

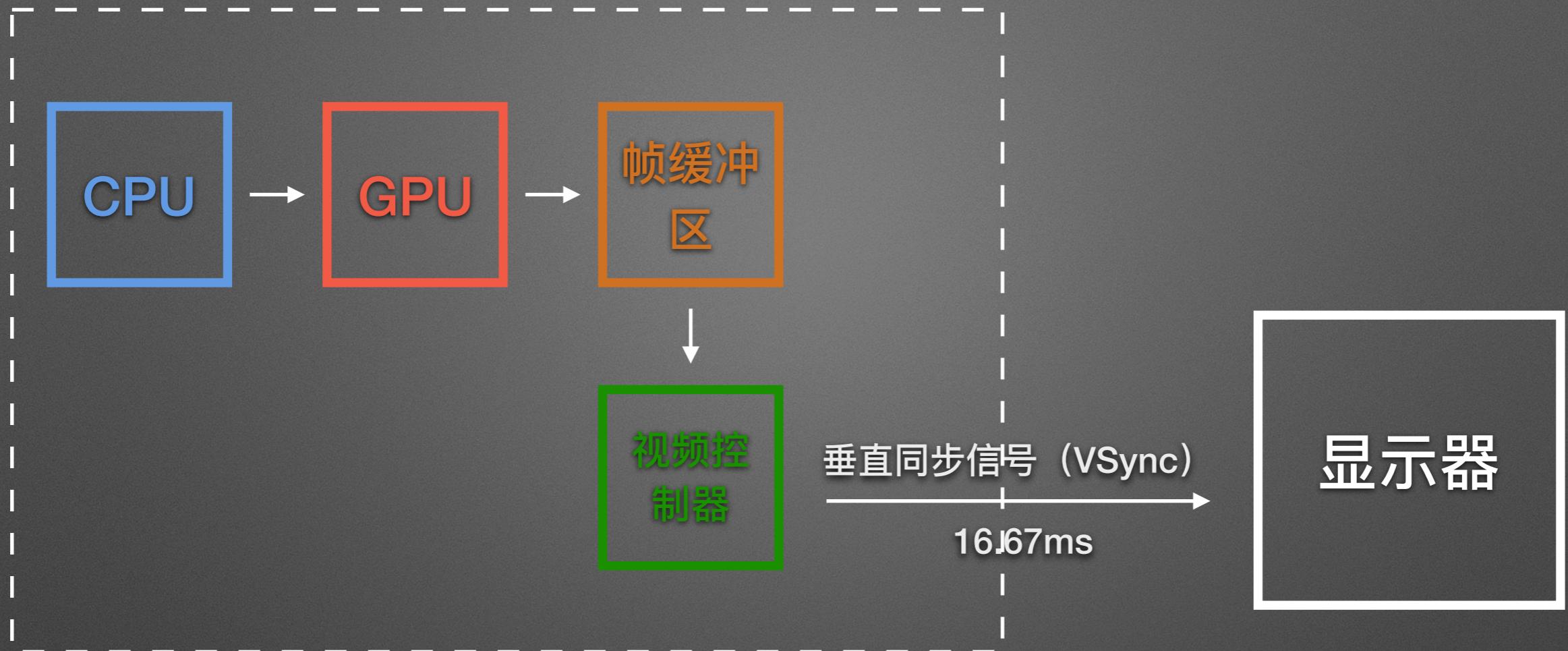




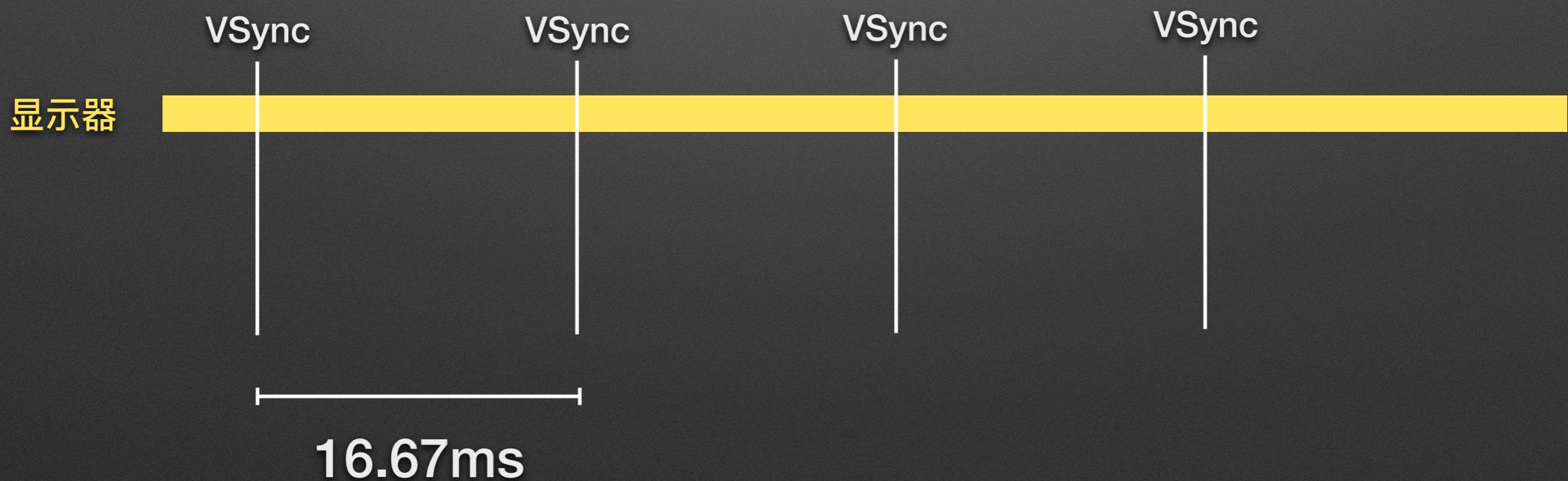
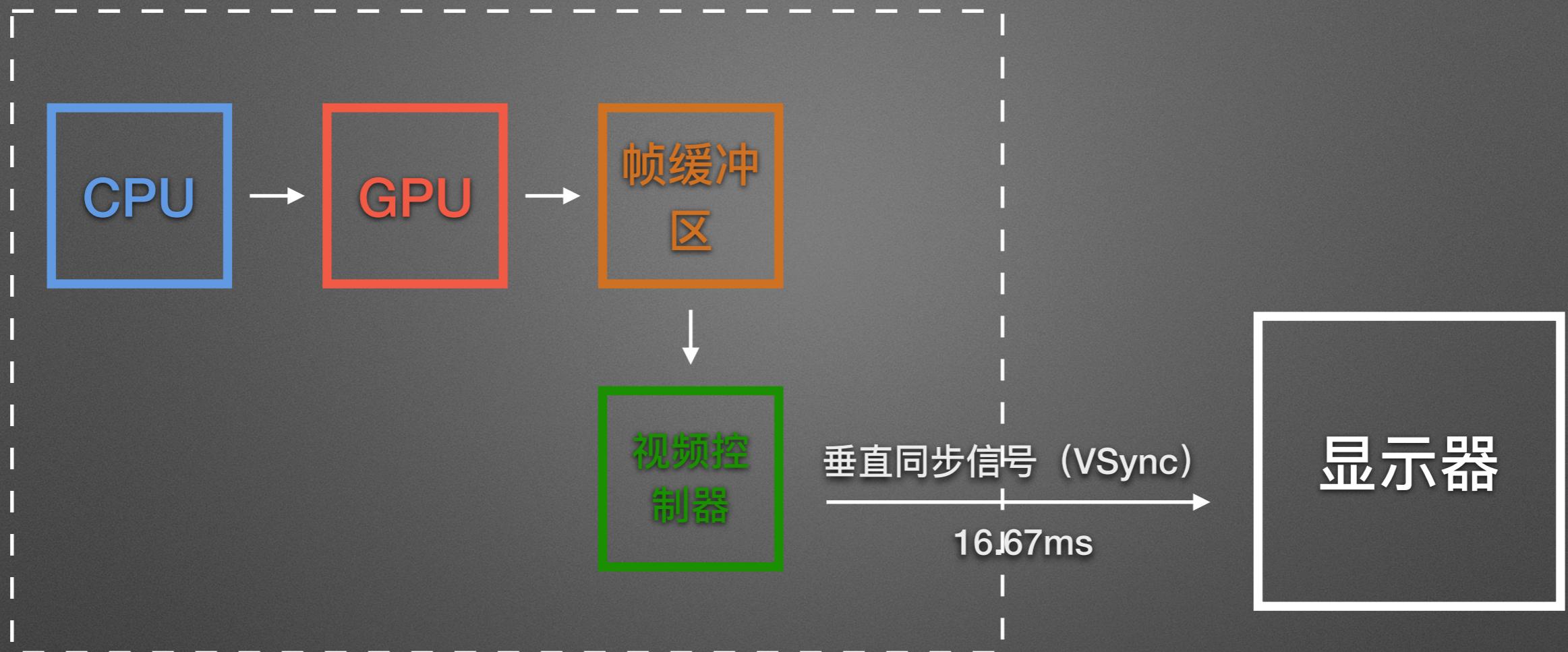
# 如何提升界面流畅性？

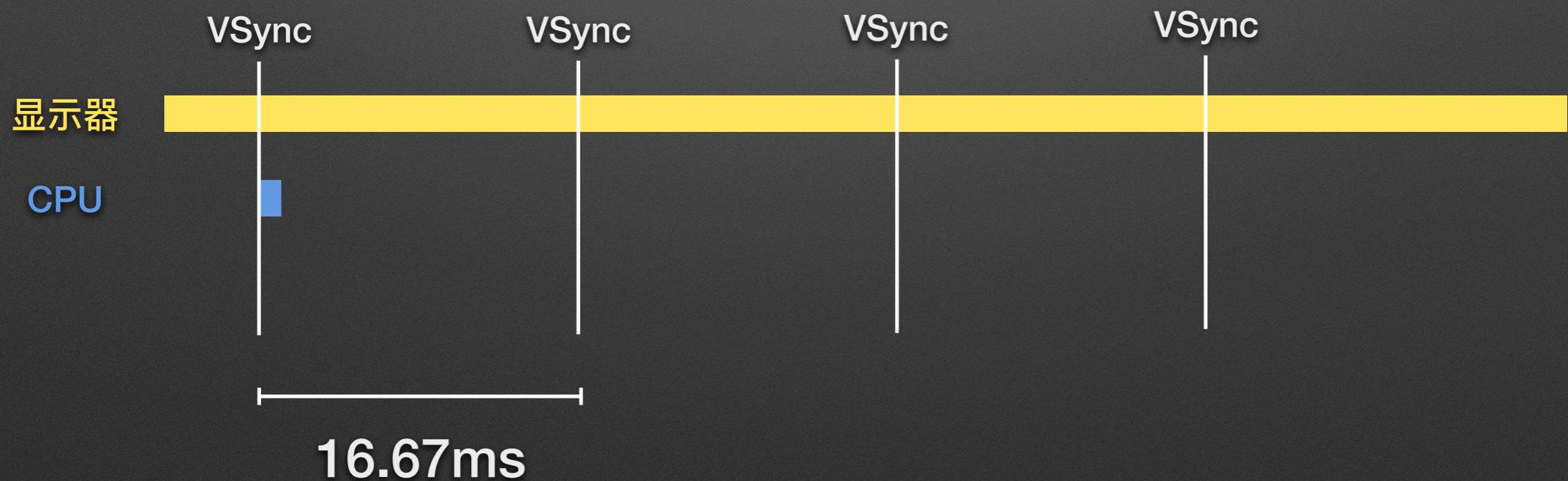
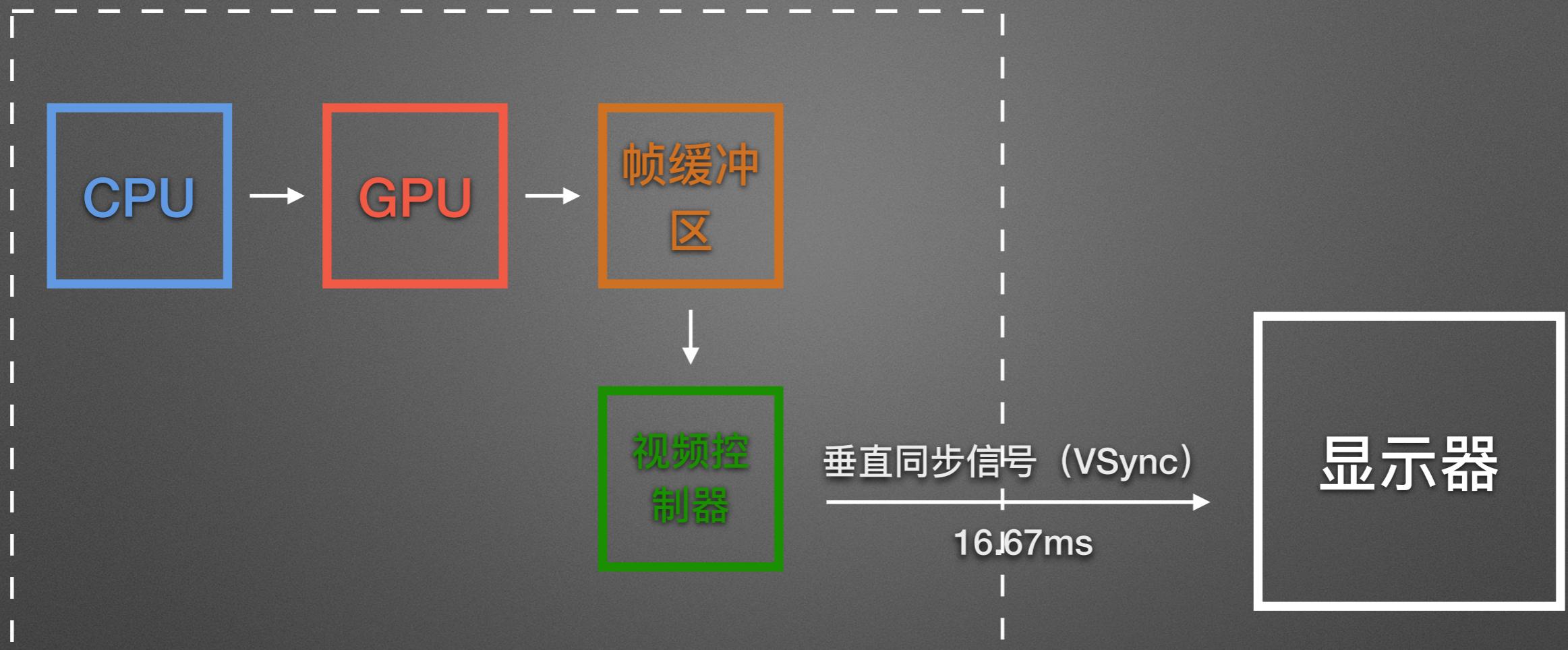


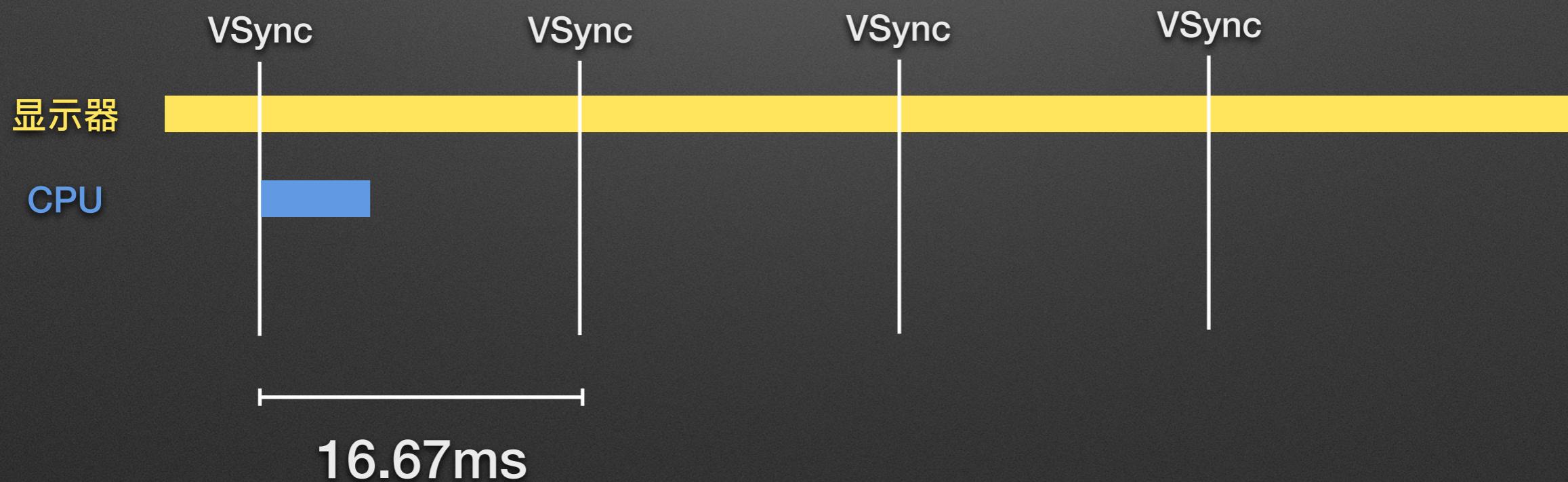
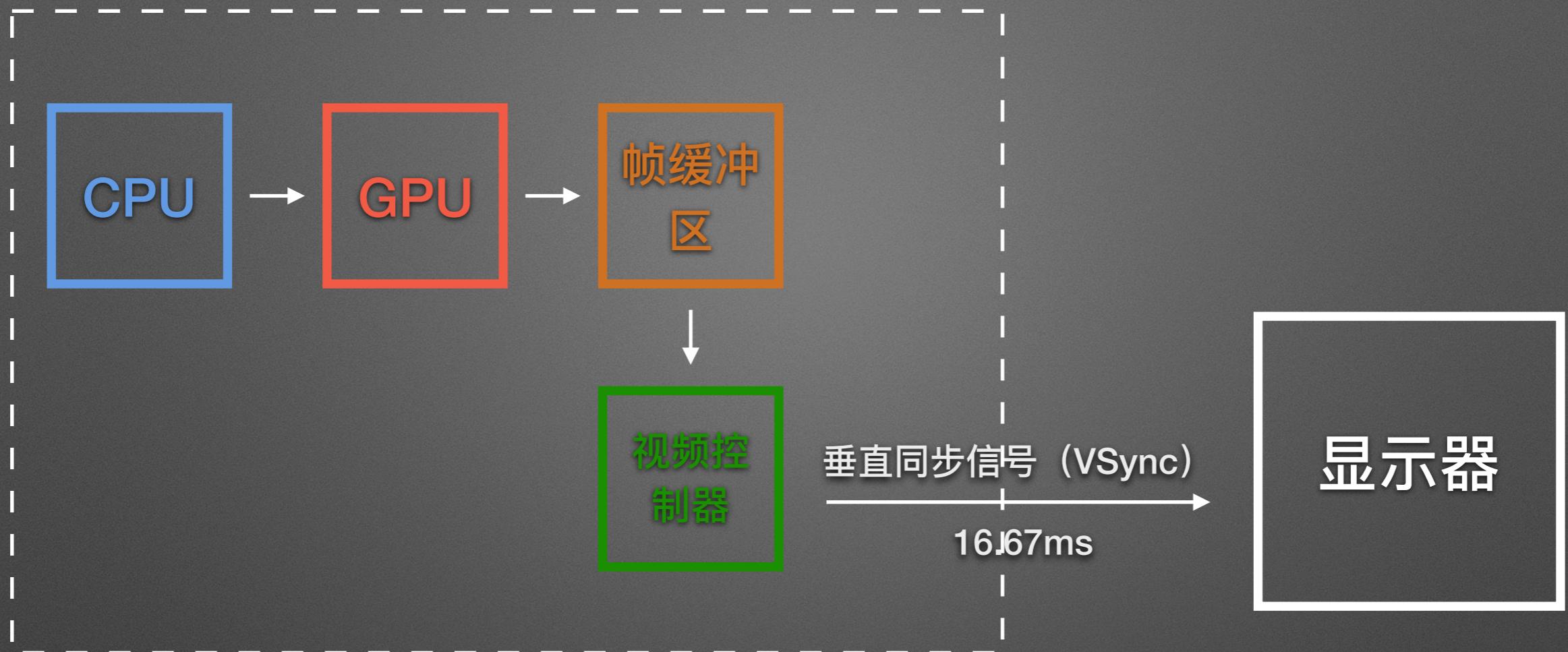
显示器 

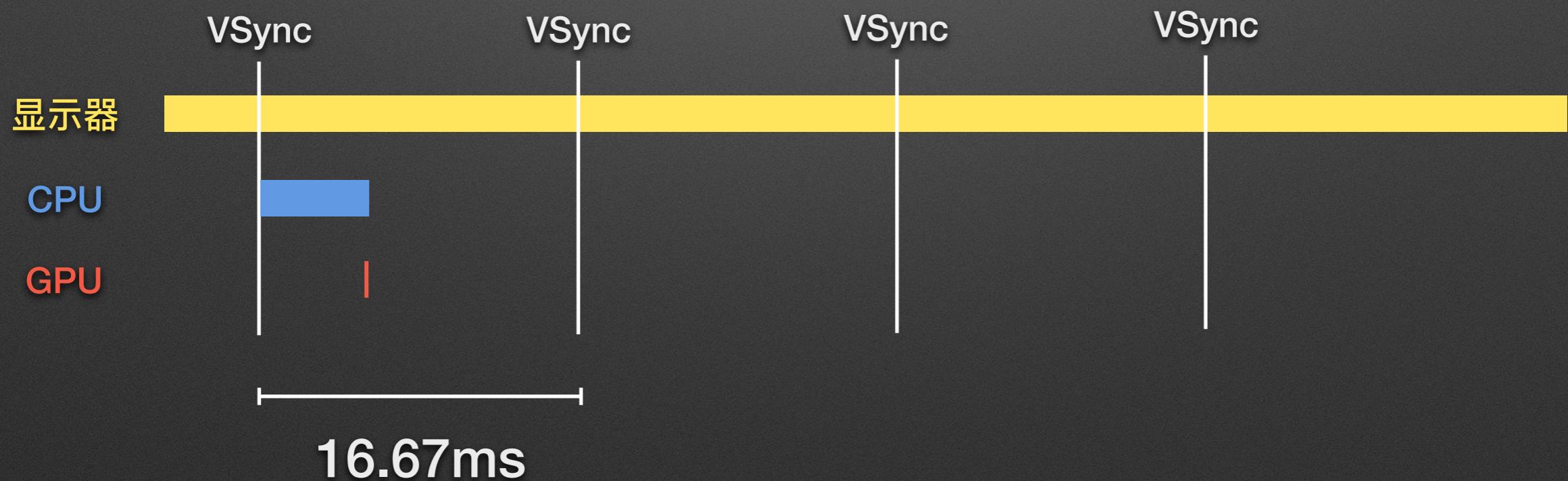
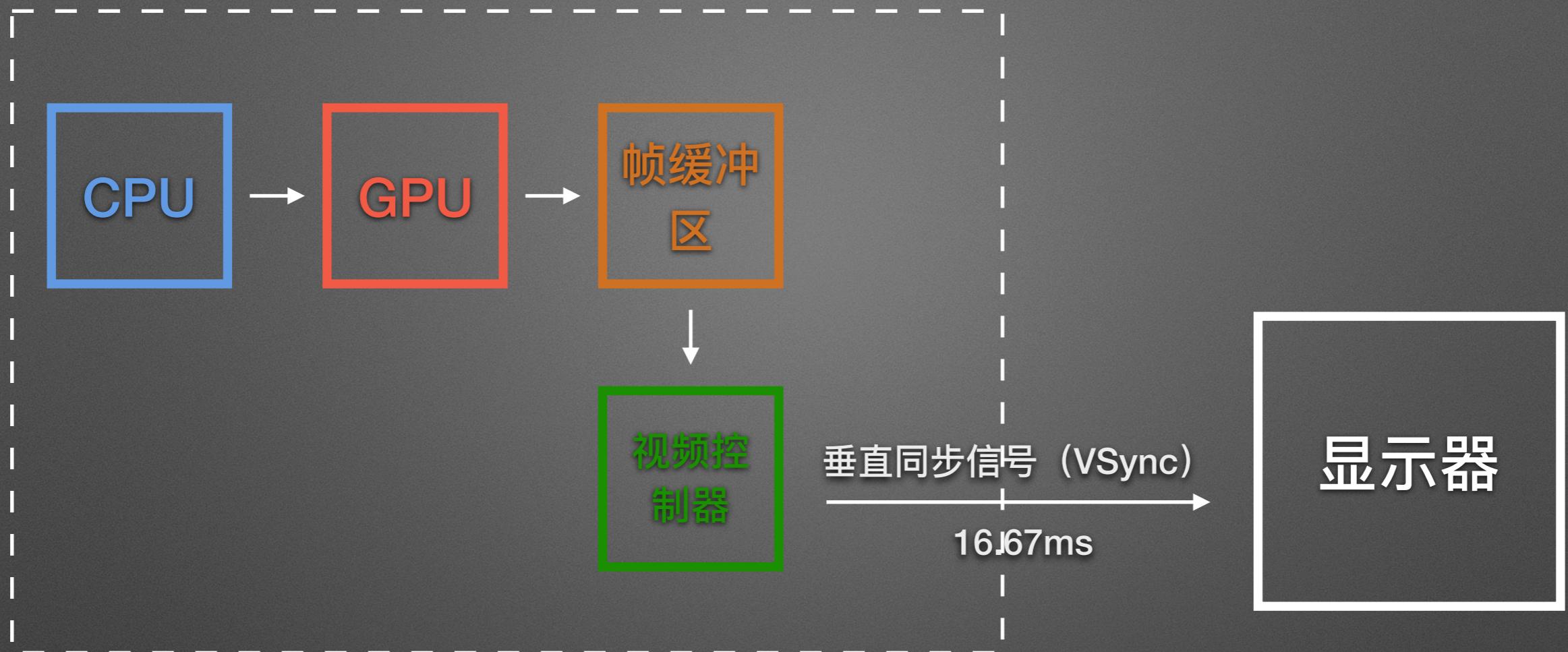


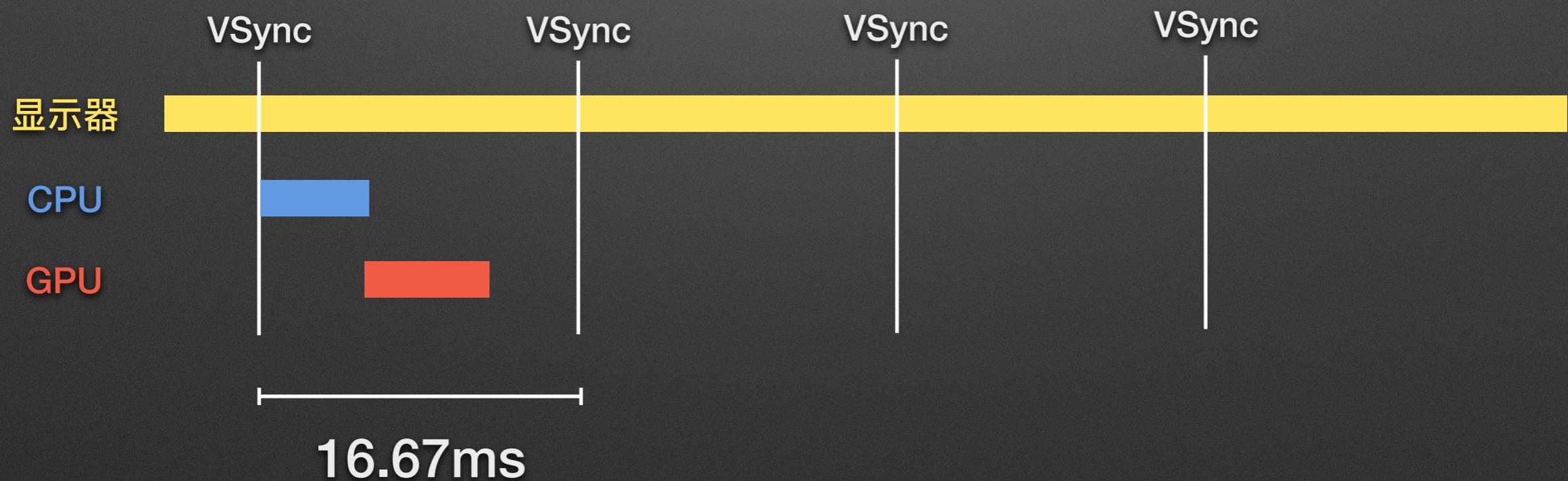
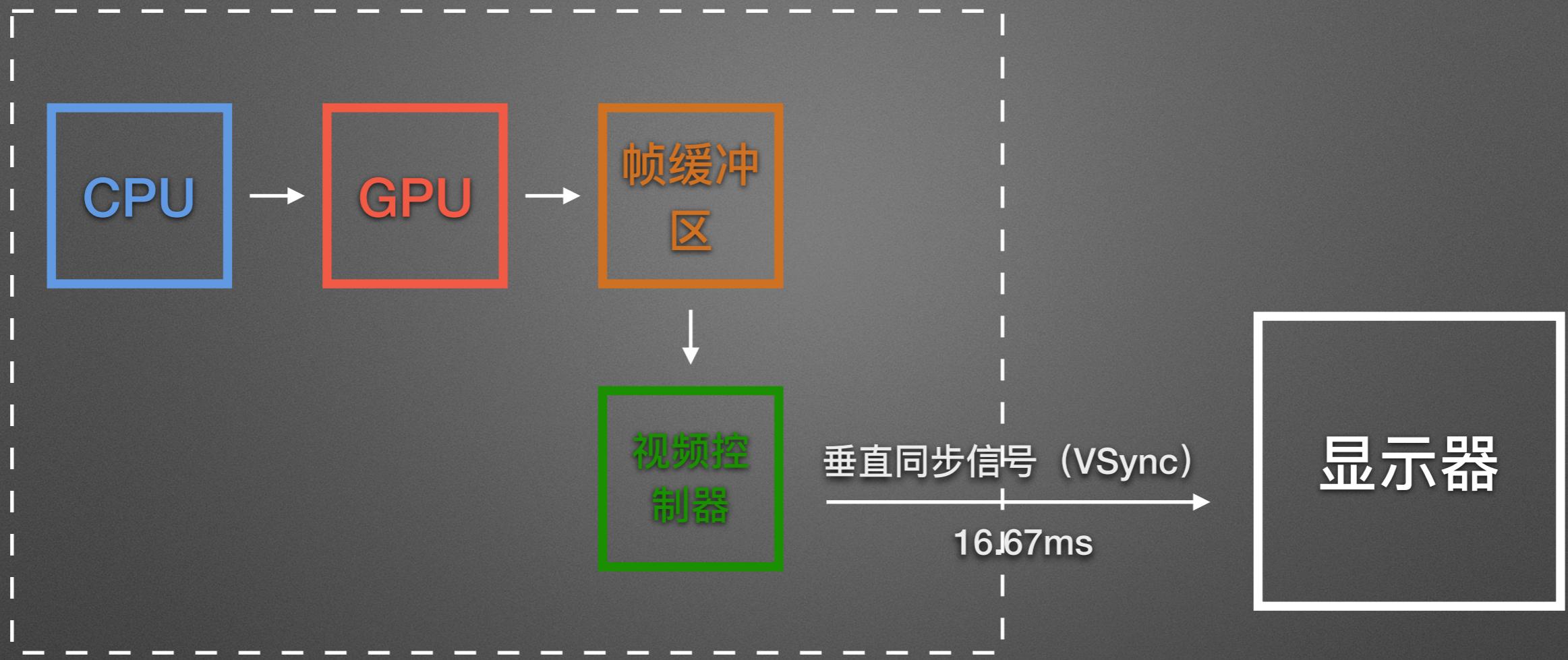
显示器

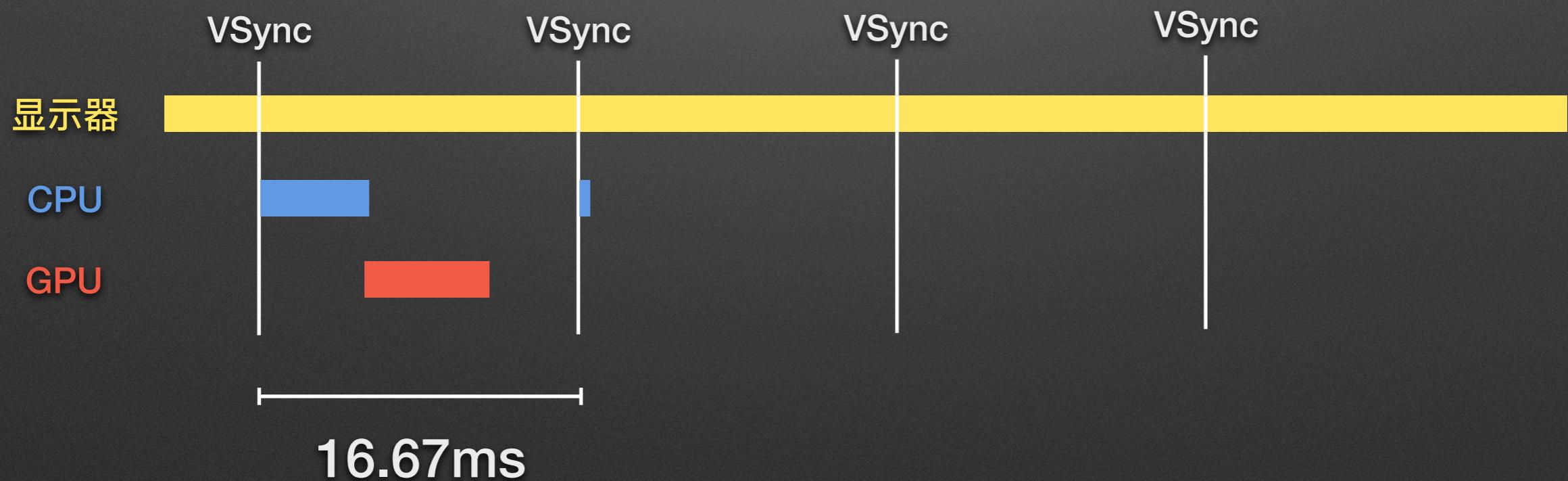
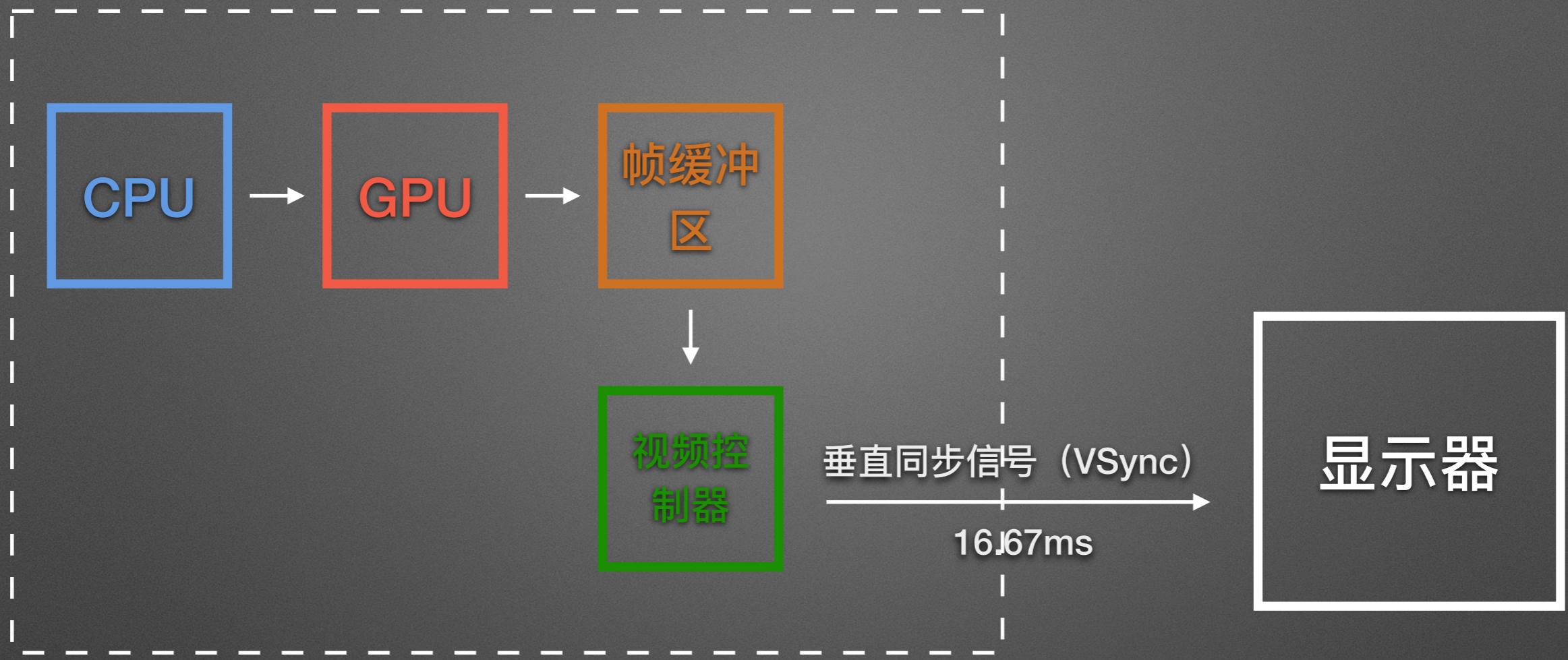


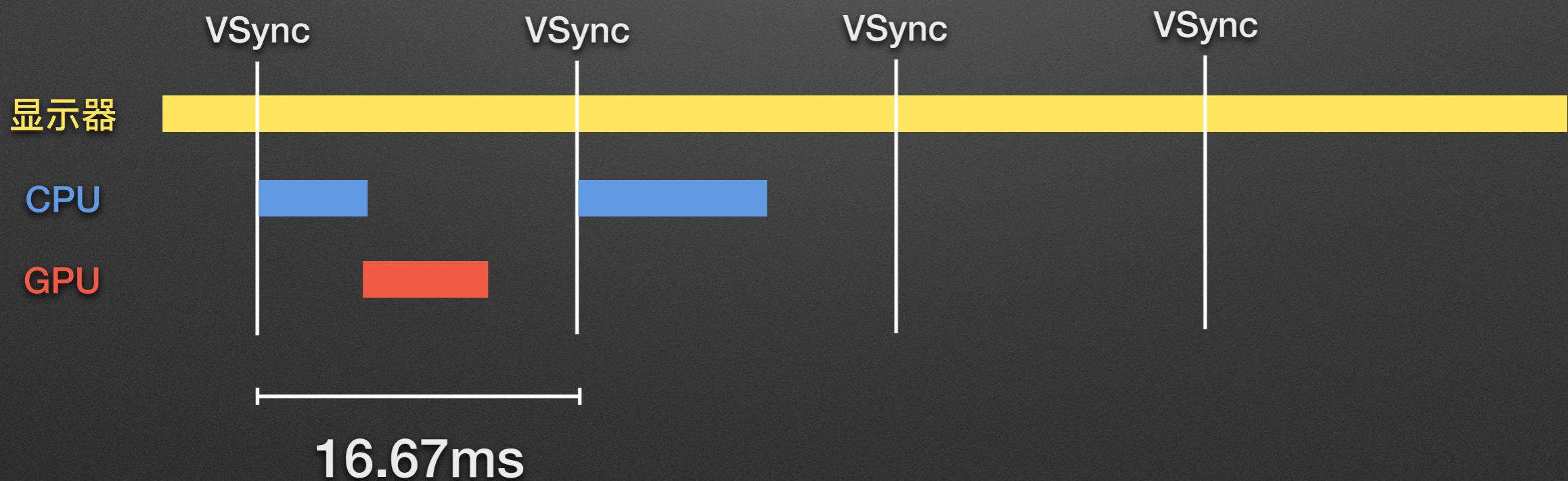
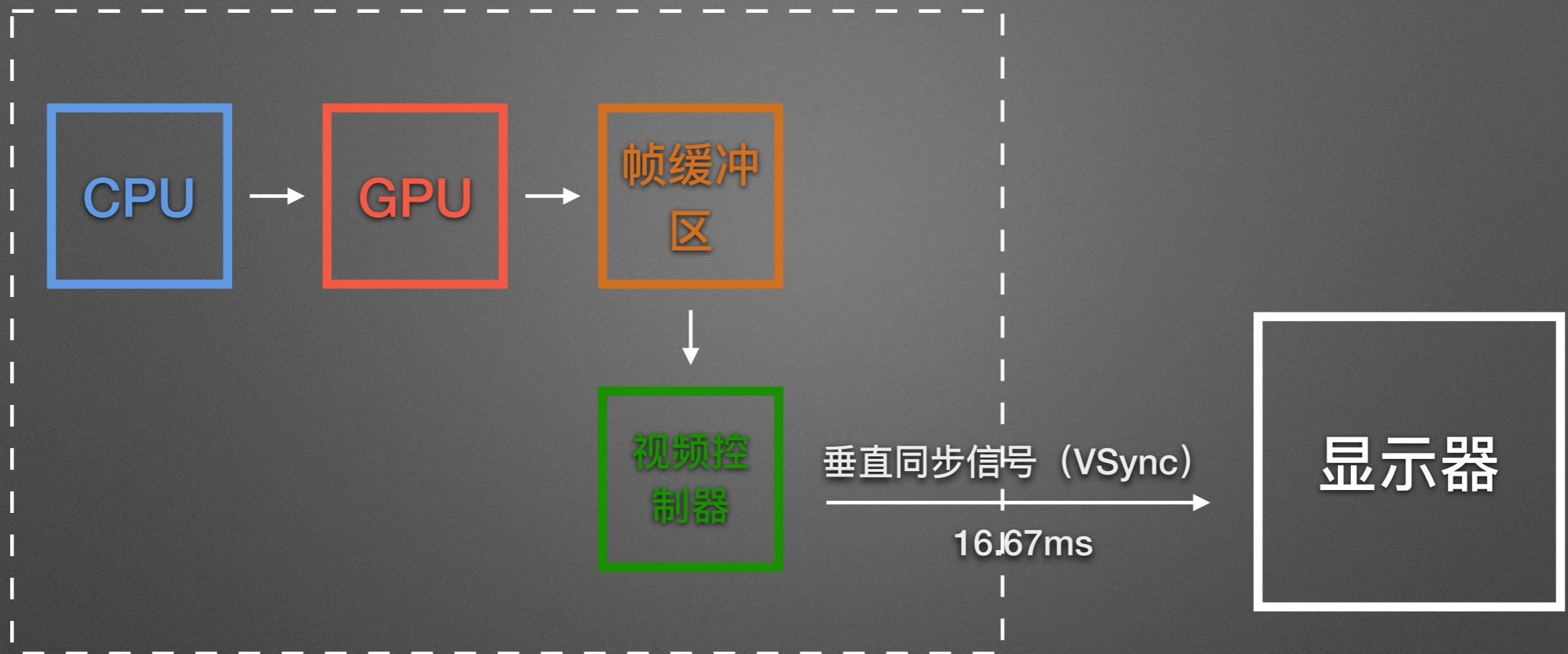


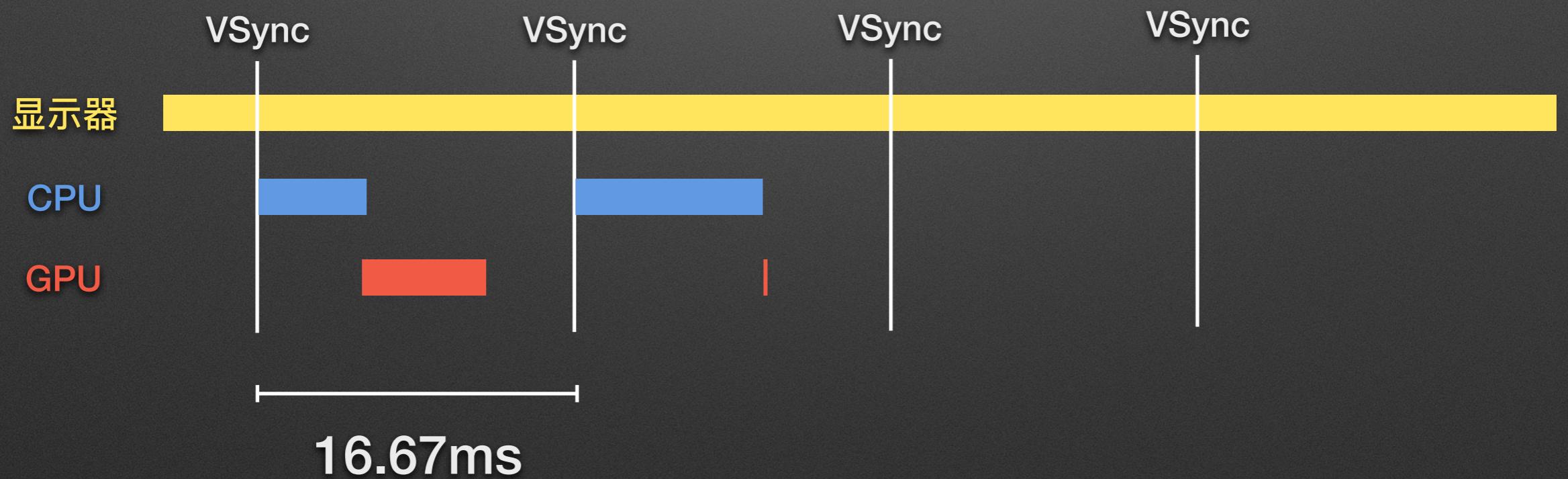
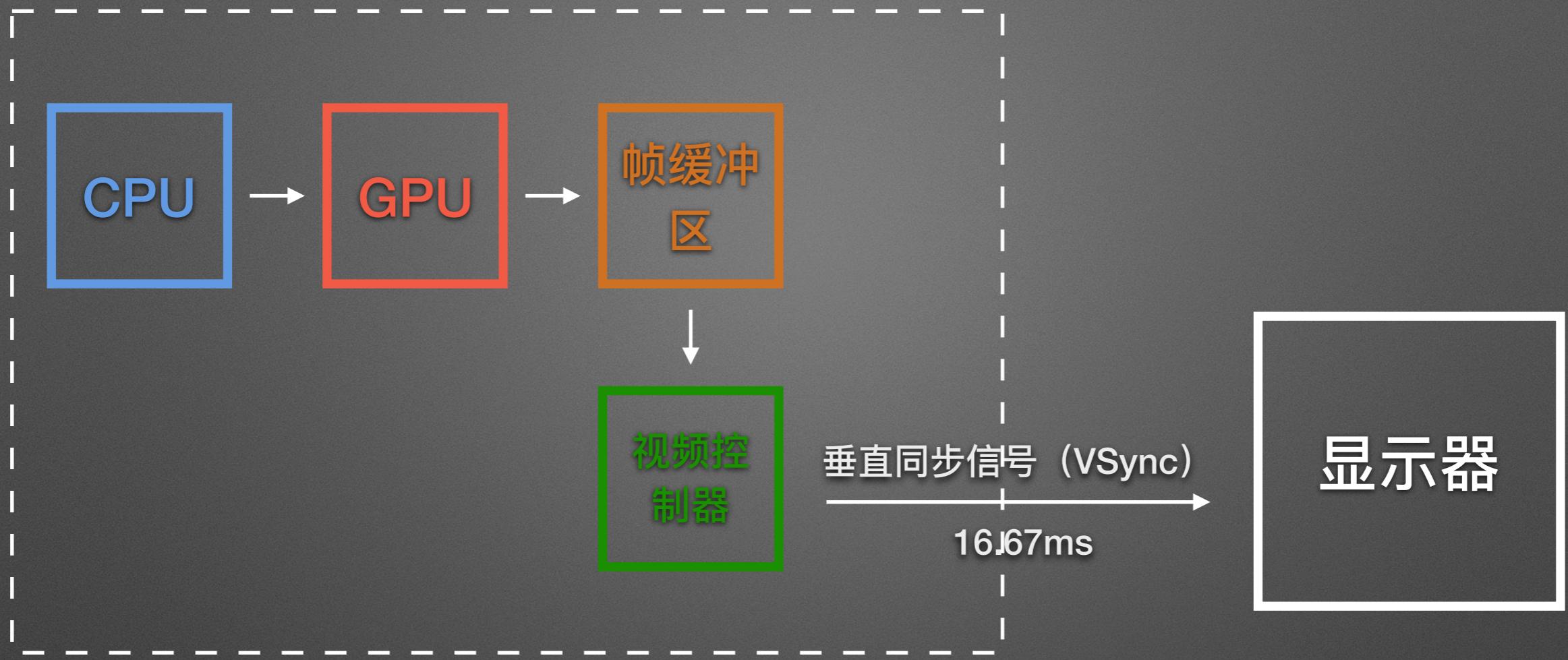


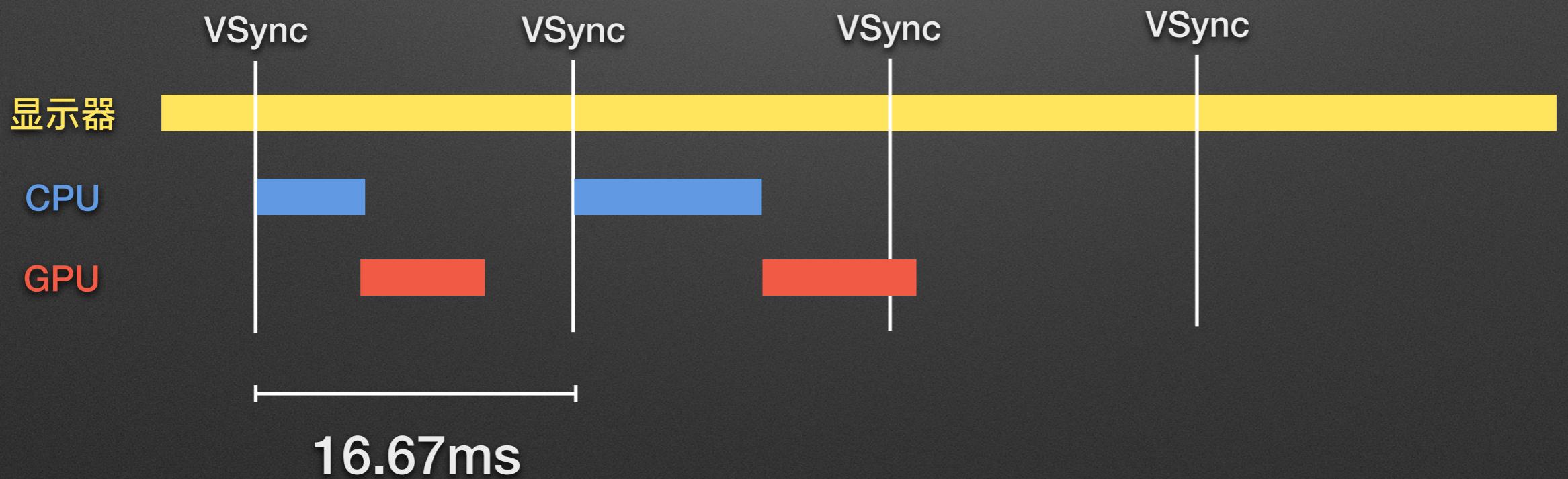
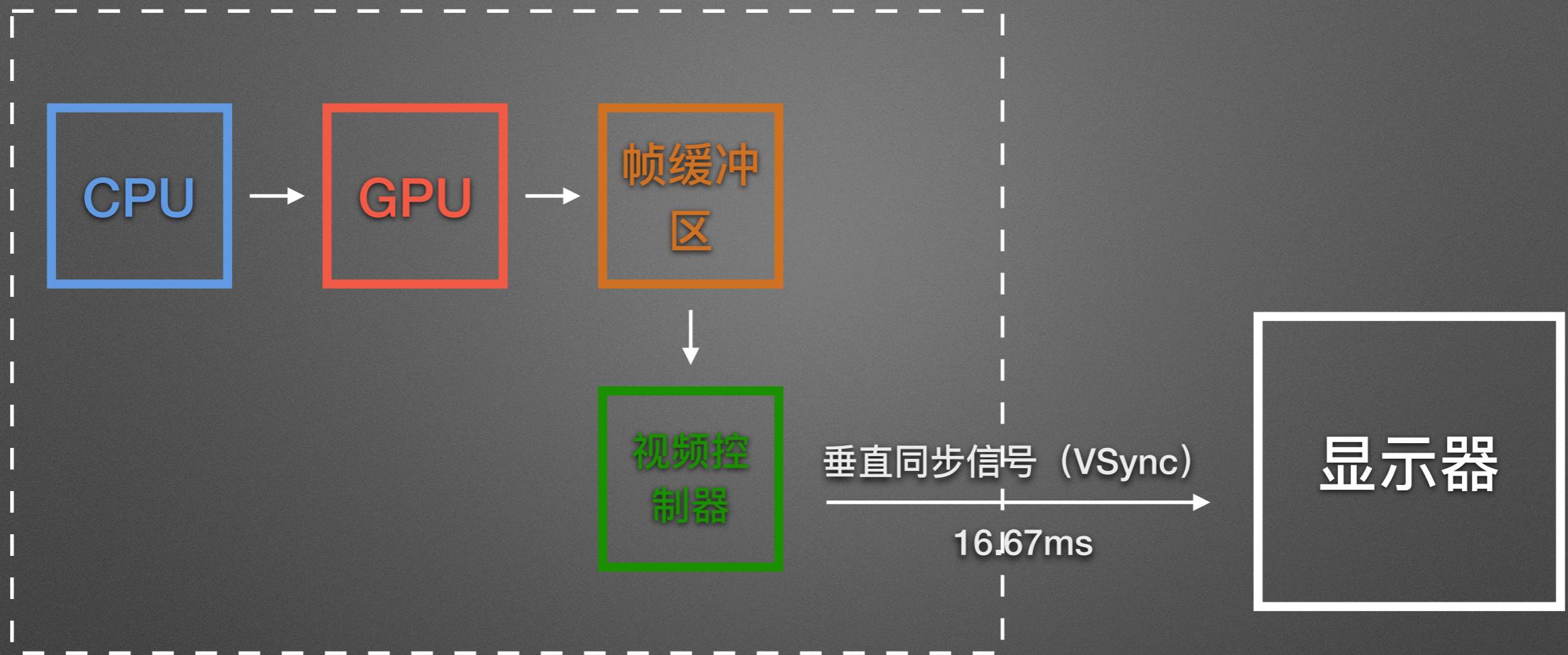












# 如何提升界面流畅性？

- 导致卡顿的原因
- 哪些操作容易导致 CPU 消耗？

对象创建

对象销毁

计算布局

文本渲染

图片解码

图像绘制

对象创建

对象销毁

计算布局

文本渲染

图片解码

图像绘制

对象创建

对象销毁

计算布局

文本渲染

图片解码

图像绘制

CALayer

对象创建

对象销毁

计算布局

文本渲染

图片解码

图像绘制

对象创建

对象销毁

计算布局

文本渲染

图片解码

图像绘制

```
NSArray *tmp = self.array;
self.array = nil;
dispatch_async(dispatch_get_global_queue
(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    [tmp class];
});
```

对象创建

对象销毁

计算布局

文本渲染

图片解码

图像绘制

异步渲染

对象创建

对象销毁

计算布局

文本渲染

图片解码

图像绘制

对象创建

对象销毁

计算布局

文本渲染

图片解码

图像绘制

**CGBitmapContext**

对象创建

对象销毁

计算布局

文本渲染

图片解码

图像绘制

# 对象创建

# 对象销毁

# 计算布局

# 文本渲染

# 图片解码

# 图像绘制

```
dispatch_async(dispatch_get_global_queue(
    DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    UIImage *image = nil;
    @autoreleasepool {
        UIGraphicsBeginImageContextWithOptions(size, NO, 0.0);
        CGContextRef context = UIGraphicsGetCurrentContext();

        // CGContext Draw ...

        image = UIGraphicsGetImageFromCurrentImageContext();
        UIGraphicsEndImageContext();
        dispatch_async(dispatch_get_main_queue(), ^{
            layer.contents = (__bridge id)(image.CGImage);
        });
    });
});
```

# 如何提升界面流畅性？

- 导致卡顿的原因
- 哪些操作容易导致 CPU 消耗？
- 哪些操作容易导致 GPU 消耗？

纹理的渲染

视图的混合

图形的生成

纹理的渲染

视图的混合

图形的生成

纹理的渲染  
视图的混合  
图形的生成

**合成技术**  
多张图合成为一张进行显示

纹理的渲染

视图的混合

图形的生成

纹理的渲染  
视图的混合  
图形的生成

**合成技术**  
多张图合成为一张进行显示

纹理的渲染

视图的混合

图形的生成

纹理的渲染  
视图的混合  
图形的生成

离屏渲染  
圆角  
shouldRasterize  
(光栅化)  
图片缩放

纹理的渲染  
视图的混合  
图形的生成

离屏渲染  
圆角  
shouldRasterize  
(光栅化)  
图片缩放

layer.mask  
ayer.masksToBounds / view.clipsToBounds  
layer.allowsGroupOpacity  
layer.opacity<1.0  
layer.shadow  
layer.shouldRasterize  
UILabel  
CATextLayer  
Core Text  
do with CGContext in drawRect:

# 纹理的渲染 视图的混合 图形的生成

综上，  
合成技术一箭三雕：  
CPU 避免了创建 UIKit 对象的资源消耗；  
GPU 避免了多张 texture 合成和渲染的消耗；  
更少的 bitmap 也意味着更少的内存占用。

## 如何提升界面流畅性？

- 导致卡顿的原因
- 哪些操作容易导致 CPU 消耗？
- 哪些操作容易导致 GPU 消耗？

# 性能优化

Performance Optimization

原理篇

实战篇

调试篇

# 性能优化

Performance Optimization

原理篇

实战篇

调试篇

# 实战篇

*Demo*

预排版（异步）

异步合成

异步绘制圆角

轻量的 UIView

替代

重量的 UIImageView

# 预排版（异步）

异步合成

异步绘制圆角

轻量的 UIView

替代

重量的 UIImageView

# 预排版（异步）

异步合成

异步绘制圆角

轻量的 UIView

替代

重量的 UIImageView

异步计算

一次缓存排版结果

预排版（异步）

异步合成

异步绘制圆角

轻量的 UIView

替代

重量的 UIImageView

预排版（异步）

异步合成

异步绘制圆角

轻量的 UIView  
替代  
重量的 UIImageView

合成技术

多张图合成为一张进行显示

预排版（异步）

异步合成

异步绘制圆角

轻量的 UIView  
替代  
重量的UIImageView

合成技术

多张图合成为一张进行显示

手动计算 frame 绘制

预排版（异步）

异步合成

异步绘制圆角

轻量的 UIView

替代

重量的 UIImageView

合成技术

多张图合成为一张进行显示

手动计算 frame 绘制

Injection Plugin

预排版（异步）

异步合成

异步绘制圆角

轻量的 UIView

替代

重量的 UIImageView

预排版（异步）

异步合成

异步绘制圆角

轻量的 UIView

替代

重量的 UIImageView

# 预排版（异步）

异步合成

异步绘制圆角

轻量的 UIView

替代

重量的 UIImageView

预排版（异步）

都是异步，如果控制并发量？

异步合成

异步绘制圆角

轻量的 UIView

替代

重量的 UIImageView

预排版（异步）

都是异步，如果控制并发量？

异步合成

VWebo

异步绘制圆角

轻量的 UIView

替代

重量的 UIImageView

预排版（异步）

都是异步，如果控制并发量？

异步合成

VWebo

异步绘制圆角

YYAsyncLayer

轻量的 UIView

替代

重量的 UIImageView

预排版（异步）

都是异步，如果控制并发量？

异步合成

VWebo

异步绘制圆角

YYAsyncLayer

轻量的 UIView

原子计数器 OSAtomicIncrement32

替代

重量的 UIImageView

预排版（异步）

都是异步，如果控制并发量？

异步合成

VWebo

异步绘制圆角

YYAsyncLayer

轻量的 UIView  
替代

重量的 UIImageView

原子计数器 OSAtomicIncrement32

“libkern/OSAtomic.h”