GPU 的探究

Tags	计算机
URL	
注	

第一部分：计算过程

为什么**gpu**会将**if**和**else**都执行

一组线程，她们执行的指令都是相同的，也就是锁步执行，只是数据不同。

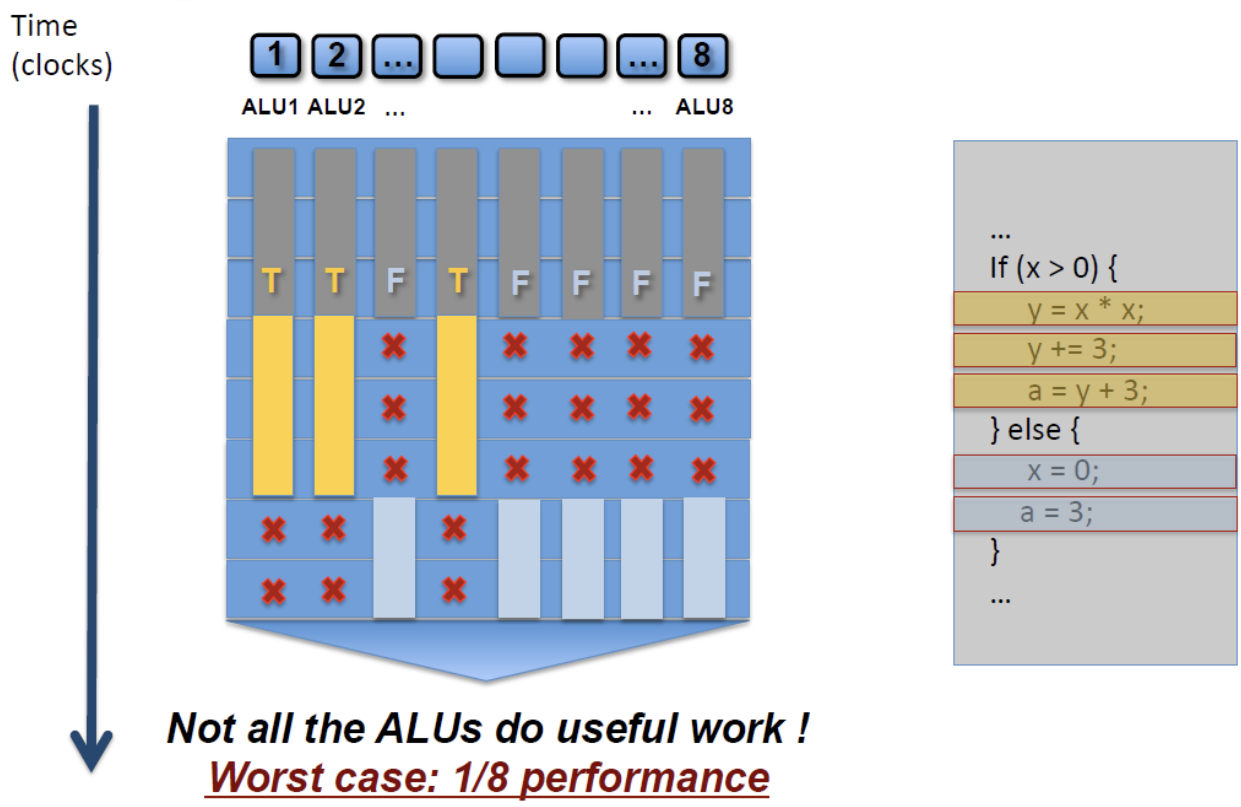
运行到if时，一些线程满足条件，继续运行，另一些不满足，被遮蔽；然后接着处理else。

这里有两个点

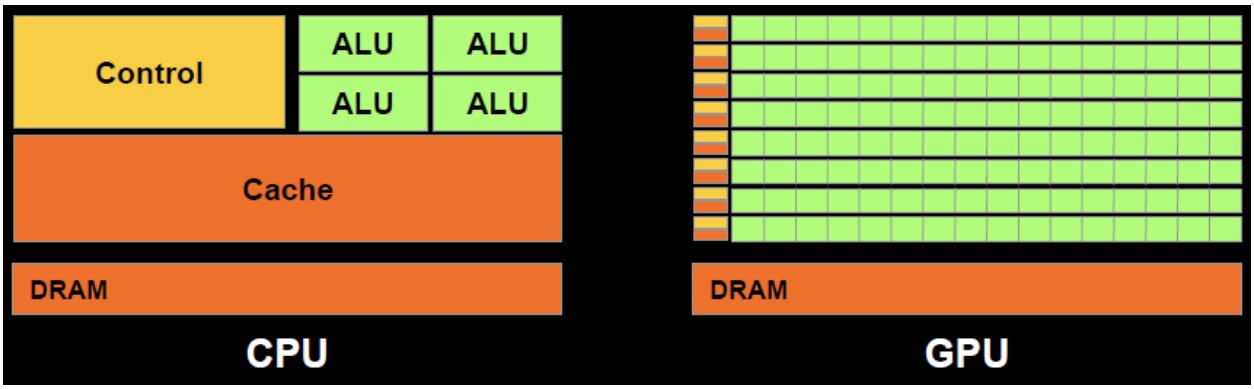
1，为什么要让一组线程同步，因为这样用于控制和解码的单元就会大幅度减少。

2，gpu的if编译结果应该和cpu不同，if和else都会走到，不需要跳转，相应的应该有遮蔽的指令（猜）

for循环也是同理

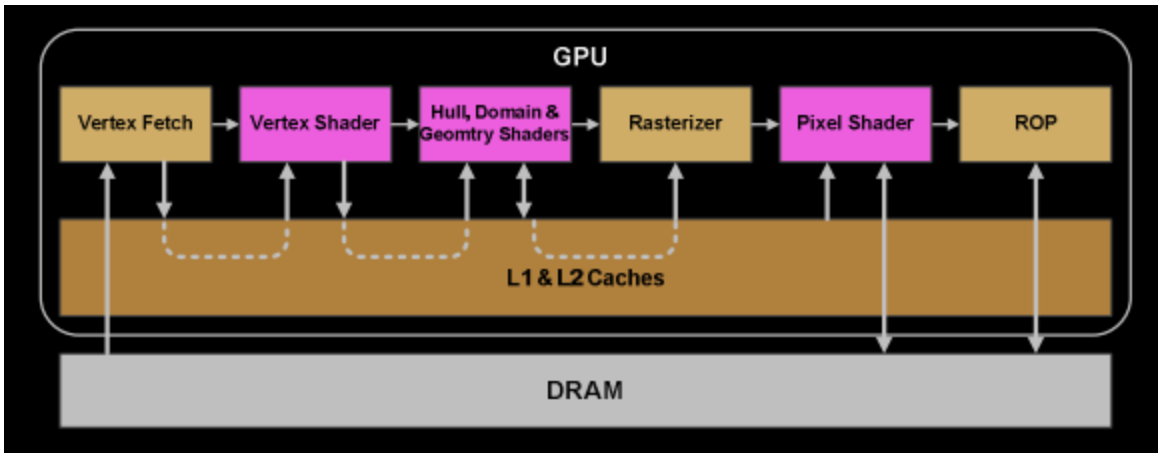


现在就能看懂这张图了

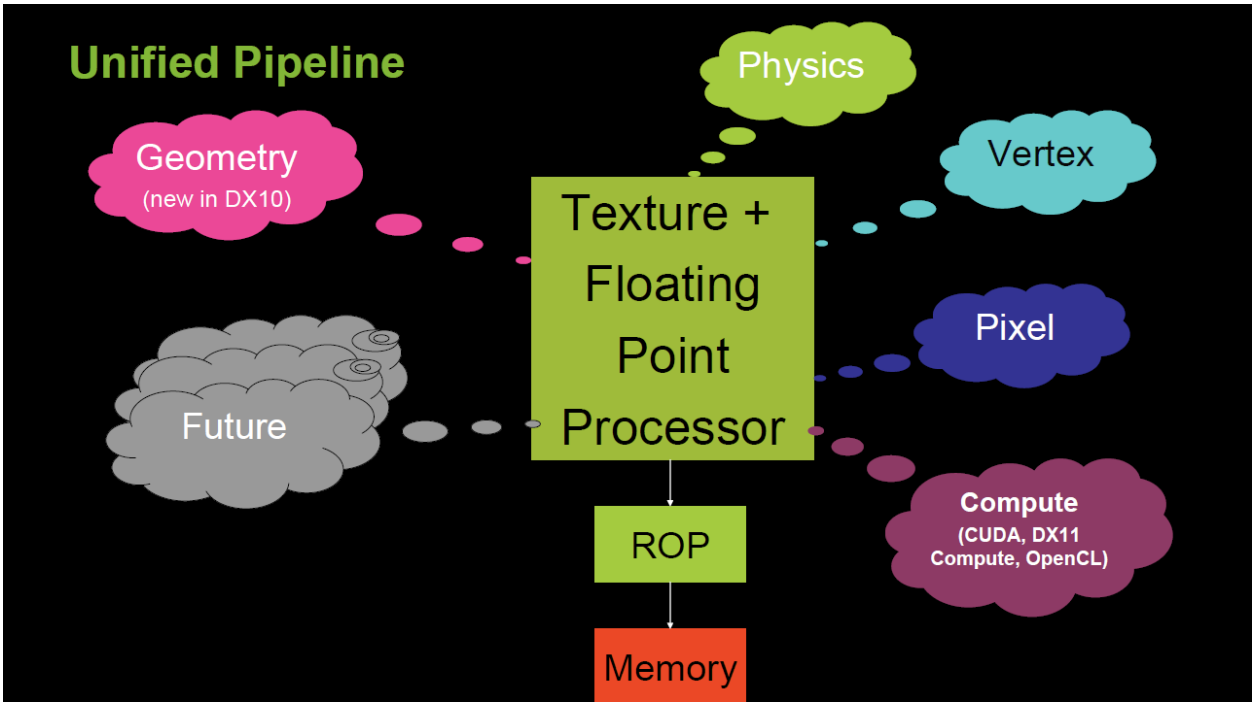


基本单元是sp流处理器，或core，thread，32个组成一个wrap。几个wrap组成一个sm，stream multiprocessor

然后：光栅化的各个阶段还真他妈被写死在显卡里了

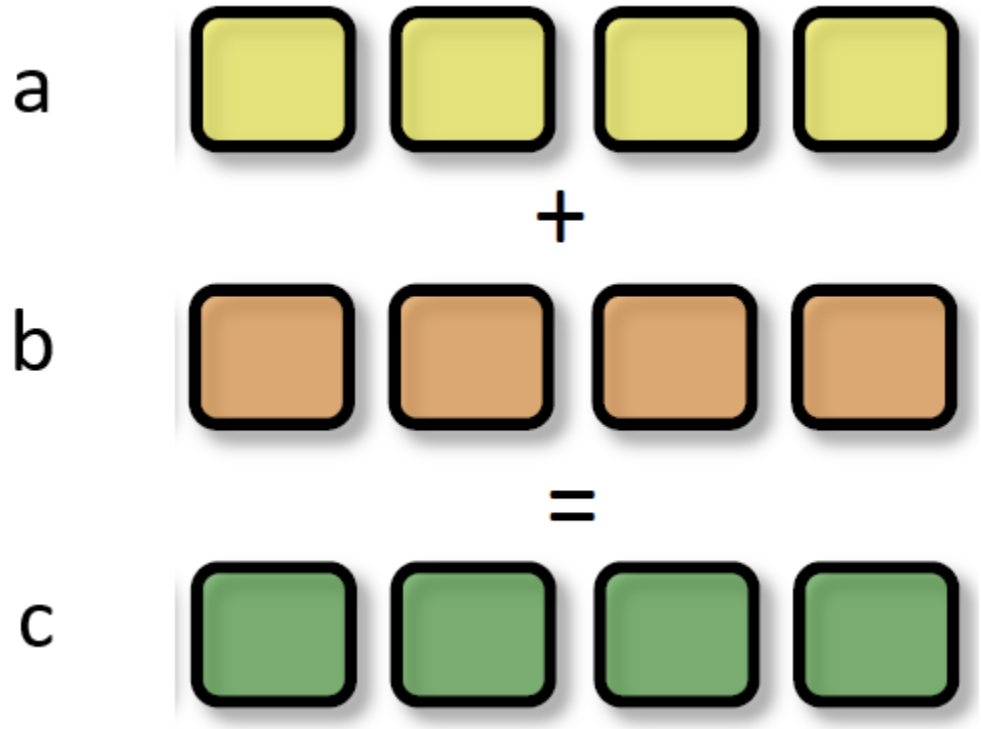


后来引入了统一着色器：vs，fs，计算着色器，几何着色器

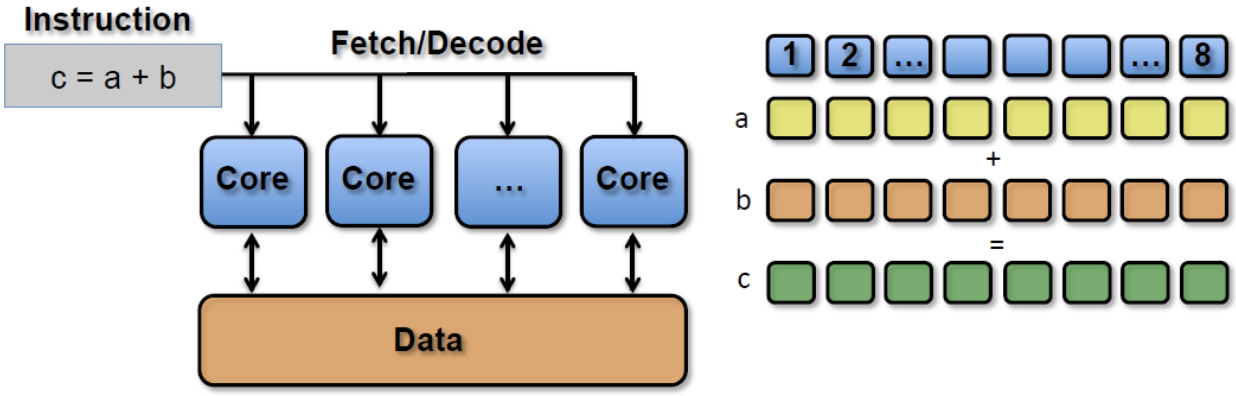


simd：单指令多数据

比如做向量加法，这里的指令就是加法，数据有n组，每一组对应一个向量分量

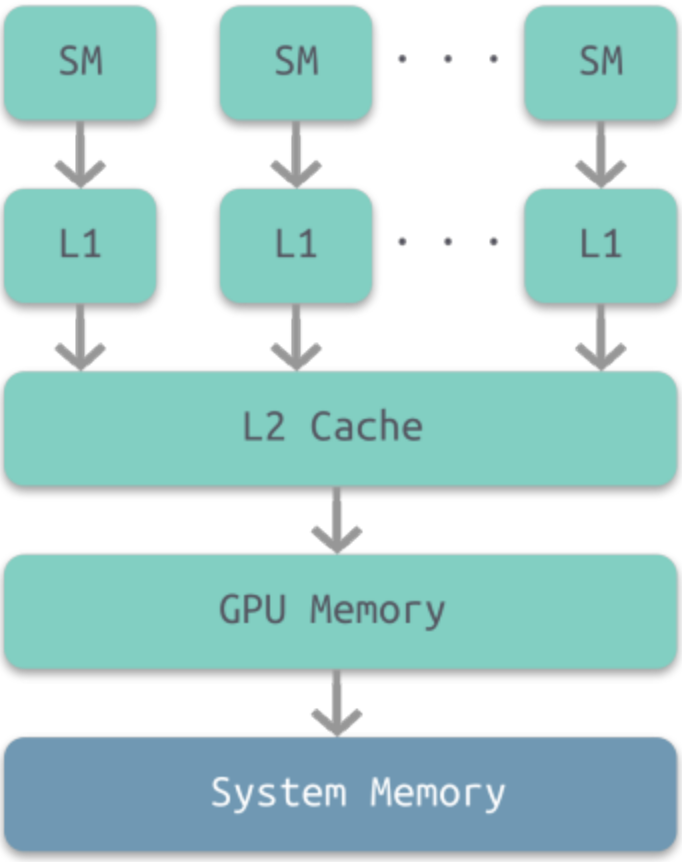


simt是simd的升级版，执行者变成了线程/cores



第二部分：存储管理

看图：gpu缓存一般是存放纹理，常量的

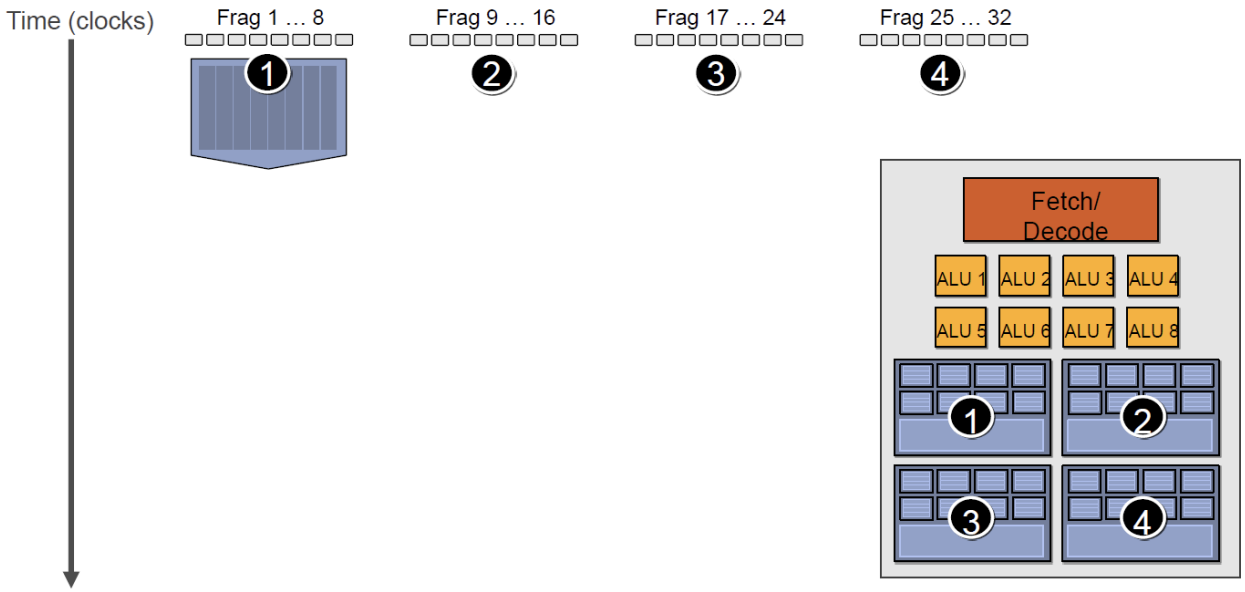


上下文单元：

上下文表示gpu进行某项计算的计算状态。

一组运算单元，比如apu，可以对应很多组上下文单元

载入一个上下文执行，遇到访存时，比如访问纹理，需要大量等待时间，就将这个context切走，换其他context上。这个样子，可以最大化地利用apu。



buffer object：

就是gpu内存中的一块区域，可以存放纹理，着色器代码等

cpu会发送指令过来，命令流会被提交到硬件单元gpu channel，每个gou channel 都会关联一个context

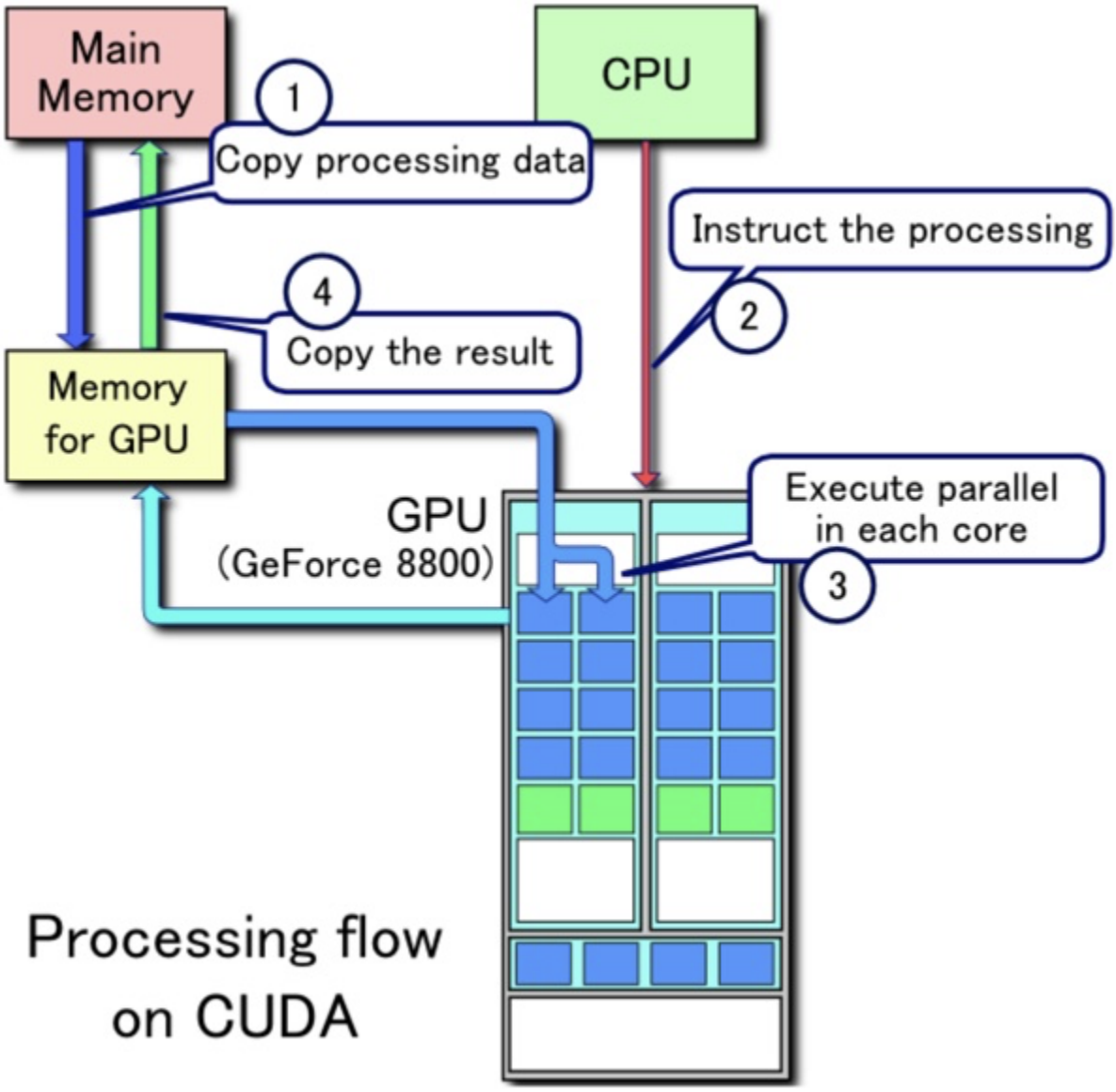
整体流程/数据流

首先是将数据copy到gpu

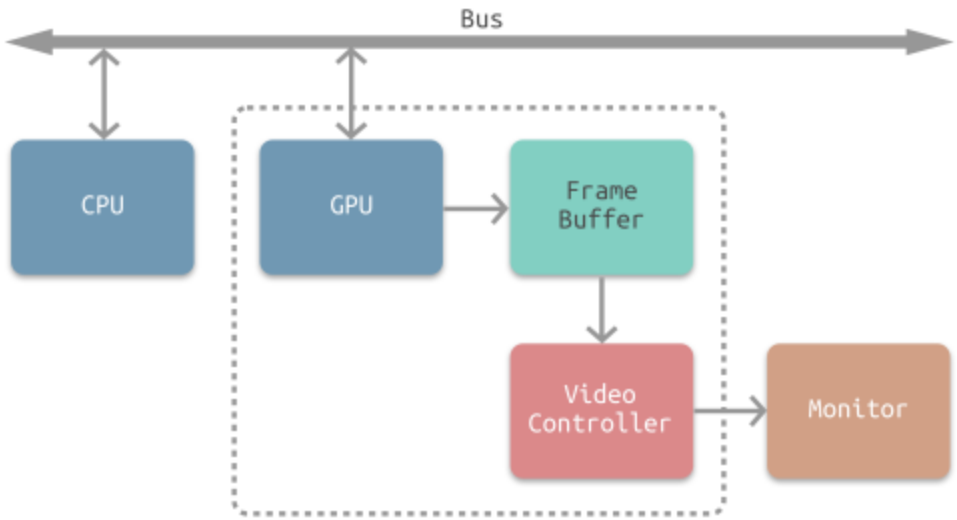
然后cpu发送指令

gpu计算

最后将结果写回帧缓存区



桢缓冲是在gpu那边的

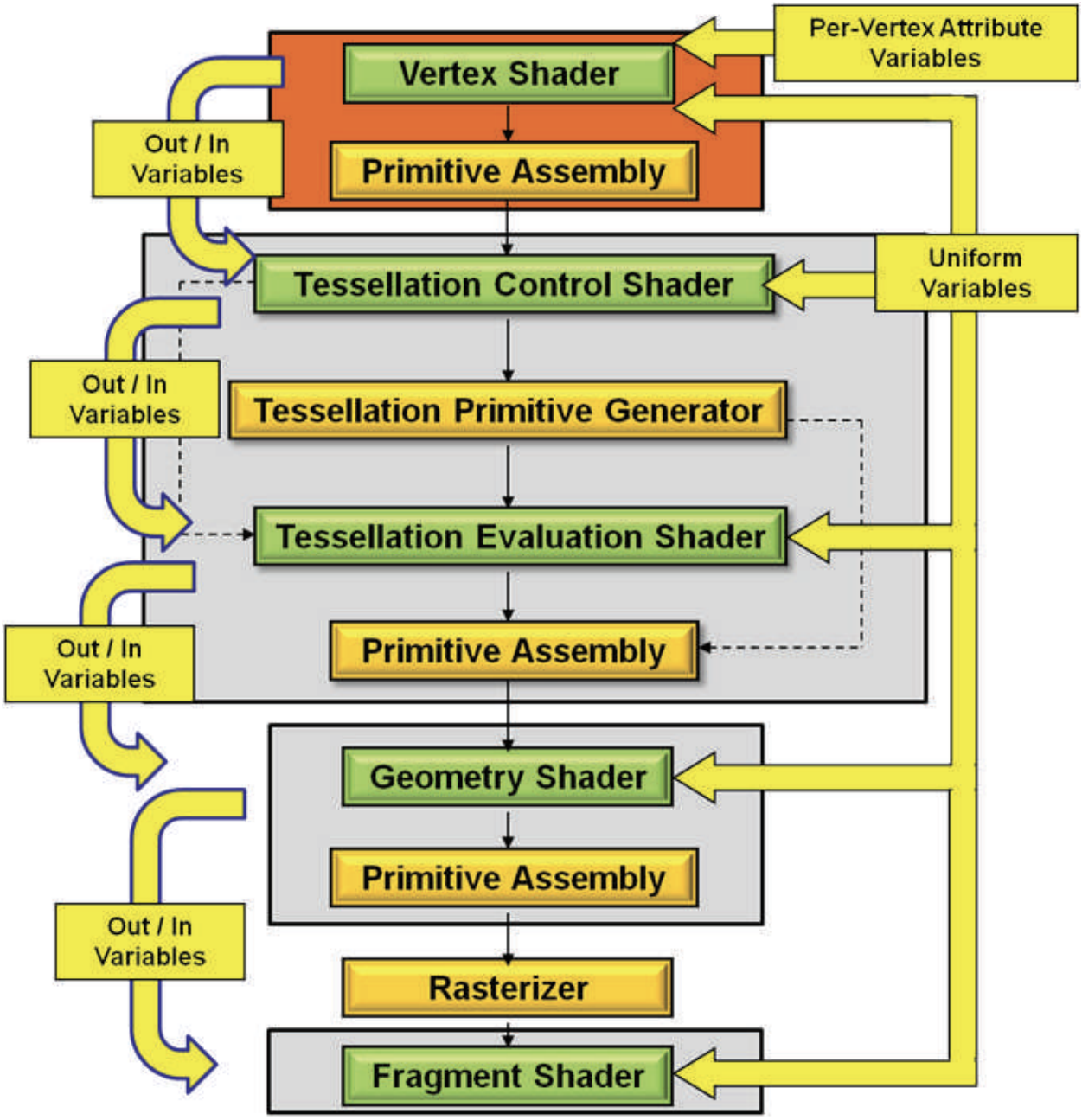


画面撕裂和垂直同步

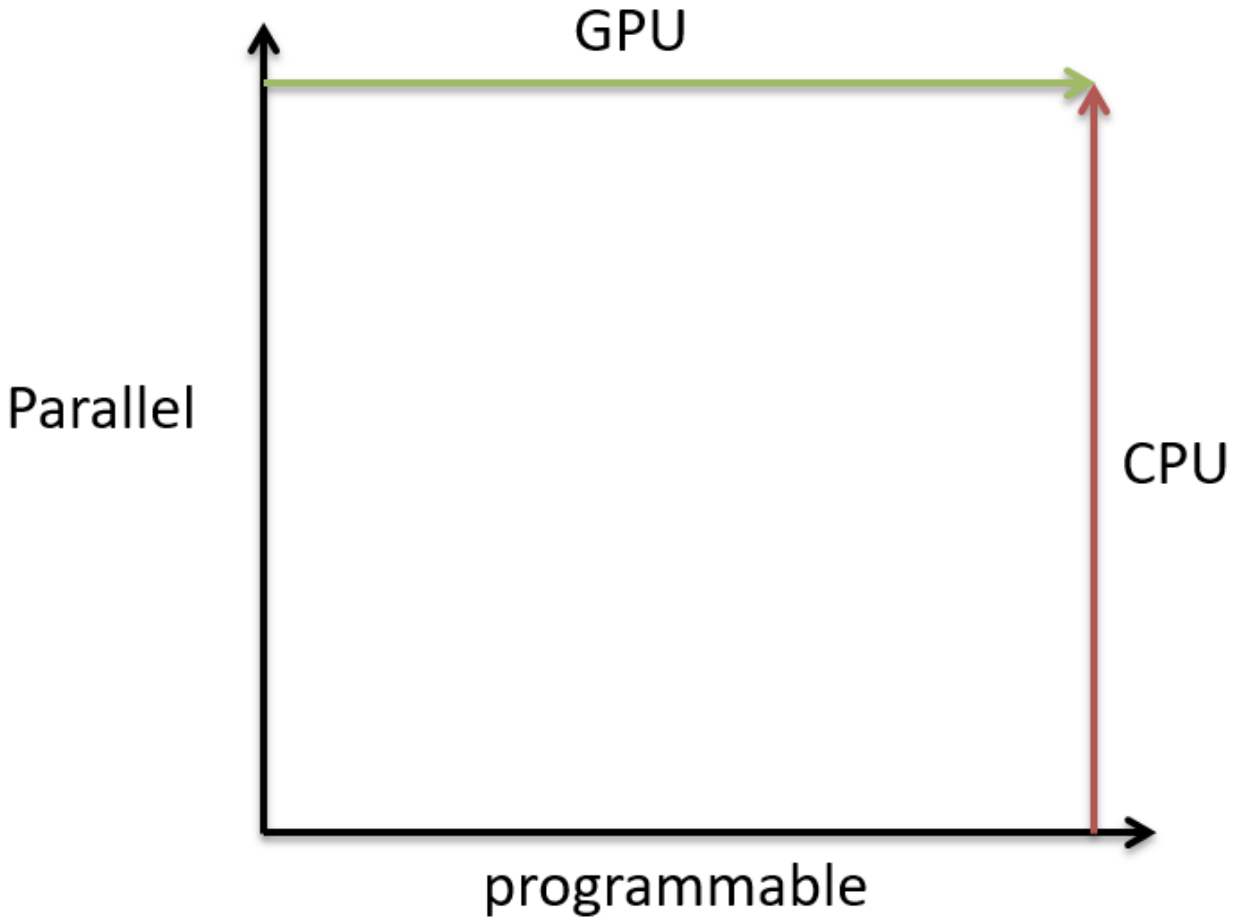
gpu渲染好后，会交换缓冲区。这个时候视频控制器才把上一帧的一半送到了显示器，由于发生了缓冲区交换，视频控制器从刚才的内存位置继续读，结果读到了下一帧的内容。最后屏幕上就会显示一半上一帧和一半下一帧，撕裂。

解决办法时，当显示器传回了当前帧显示完毕的信号，也就是垂直同步信号时，gpu才会交换缓冲区

着色器的流程，绿色是可编程的

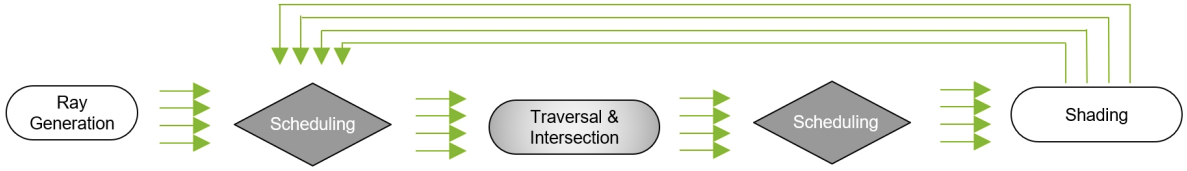


这张图可以看出来，最早gpu就是专用计算的，不是通用计算的😐，可编程的在慢慢增多



光线追踪的流程，可以参考下

RAY TRACING



RASTER