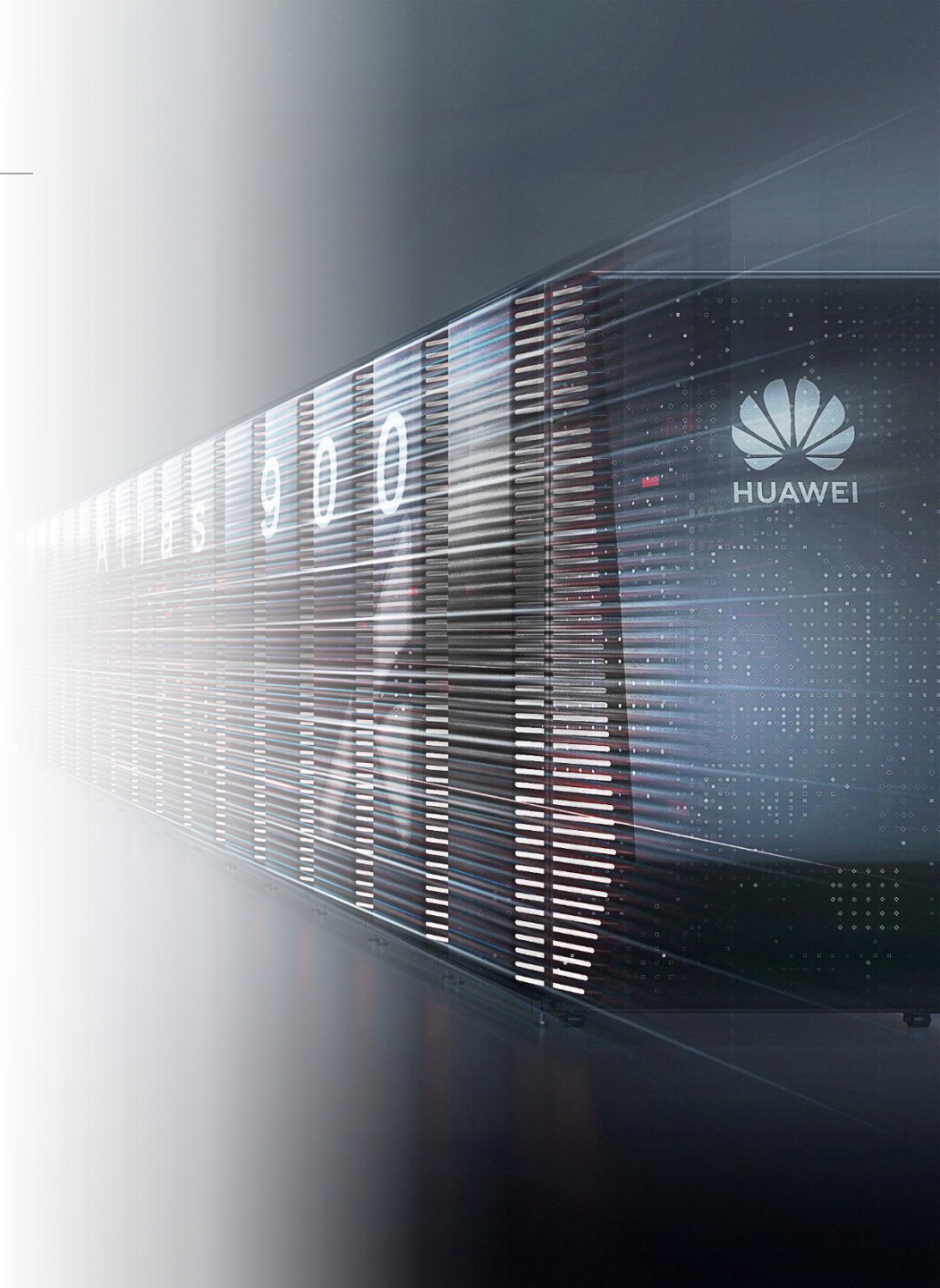


A child wearing a brown aviator helmet and goggles is sitting on the shoulder of a large, white, humanoid robot. The child is pointing their right index finger towards a large, stylized globe in the background. The globe is composed of a grid of dots and is set against a light blue sky with several bright, diagonal streaks of light. The robot's head is tilted back, and its right arm is raised, with the hand near the child's head. The robot's body is white with some mechanical details visible.

基于ACL的推理应用开发（高级篇）

——**ACL同步&异步**

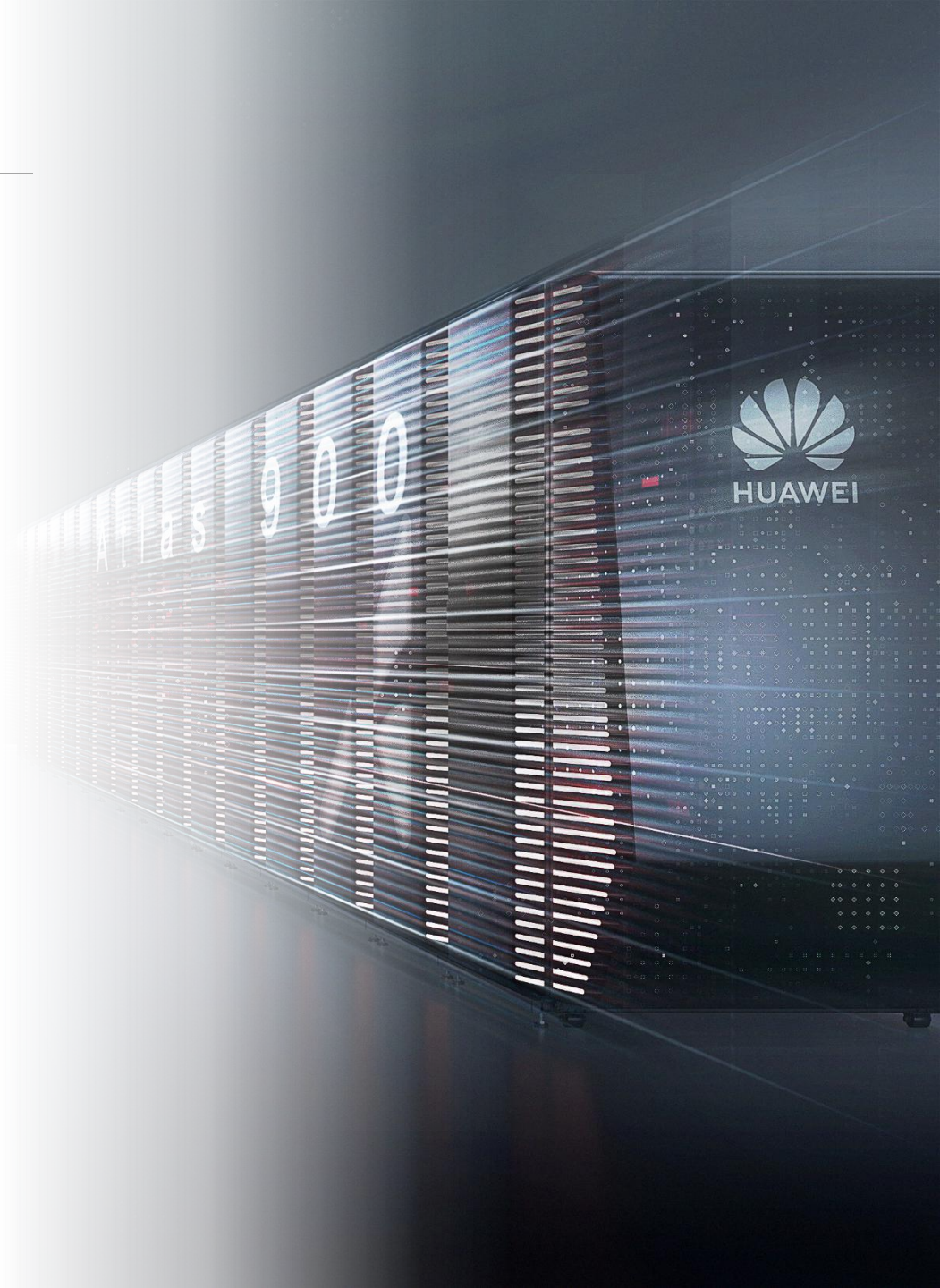
- 1 同步等待基本概念**
- 2 典型的同步等待流程**
- 3 同步等待相关接口**



同步&异步基本概念

- 什么是同步/异步？
 - 为了简化课程表述，这里我们不深究同步&异步跟阻塞&非阻塞的关系，只关注同步&异步的观察角度。
 - 同步、异步是站在调用者和执行者的角度来观察的，在当前场景下，若在Host调用接口后不等待Device执行完成再返回，则表示Host的调度是异步的；若在Host调用接口后需等待Device执行完成再返回，则表示Host的调度是同步的。

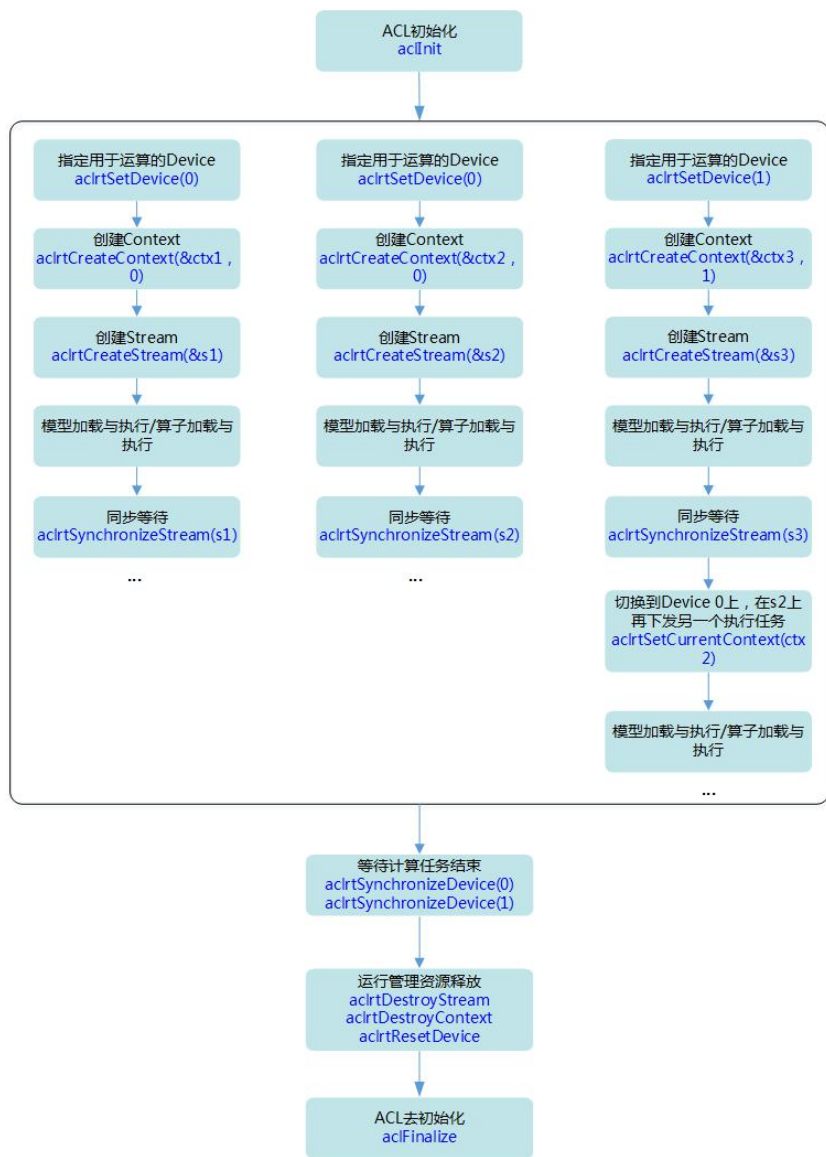
- 1 同步等待基本概念**
- 2 典型的同步等待流程**
- 3 同步等待相关接口**



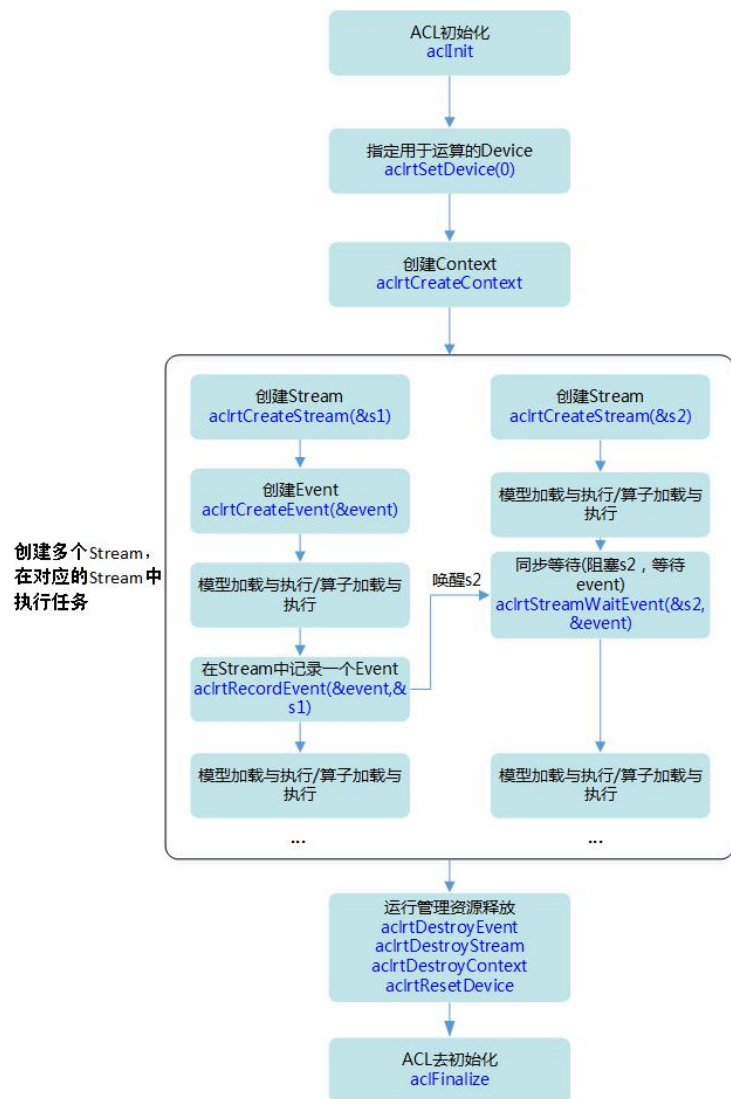
如何同时使用多个Device?

在Atlas300及以上规格的产品中，可能会涉及一个应用使用多个Device的场景，即一个Atlas设备上有多个Ascend处理器的场景。想同时使用这多个处理器（设备），通常遵循以下步骤：

1. 在各个线程中各自调用setDevice方法选择执行任务的Device
2. 每个线程创建自己的context以管理运行资源
3. 创建context的时候要指定设备编号，即context中也隐含了对设备的管理
4. 所以当想要切换设备进行任务下发时，不必调用setDevice来进行切换，选择setCurrentContext来切换context，从而切换设备就可以了，且效率比setDevice效率更高

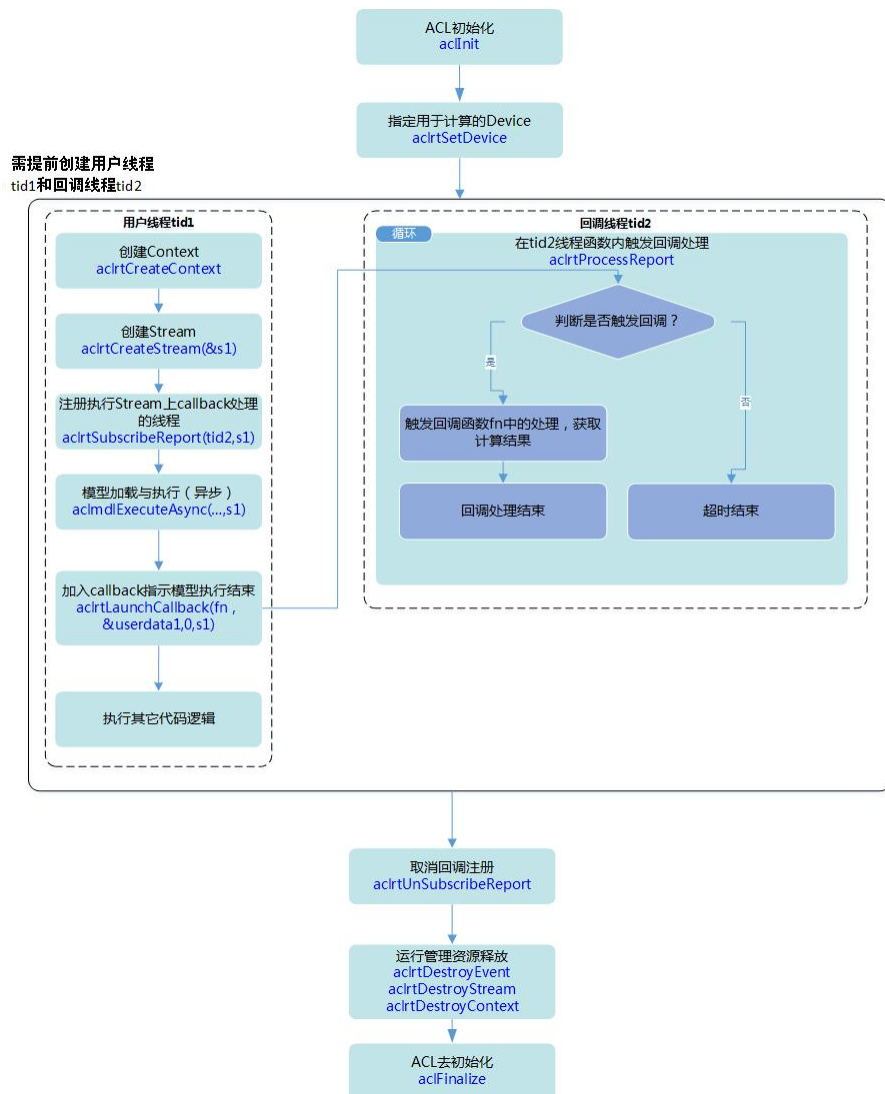


多个Stream之间的事件同步



1. 假设我们当前有2个stream (1,2)，其中stream2中的任务b，依赖stream1中的任务a执行结束
2. 此时我们需要event来介入，进行stream之间的事件同步
3. 在stream1的任务a**结束之后**，调用recordEvent来声明一个event发生了
4. 在stream2的任务b**执行之前**，调用waitEvent来等待event发生
5. 这样才能在a和b两个任务之间保序

Callback方式流程



1. 执行异步接口调用时，处理调用完毕后得到的结果可以有两种方式，即本线程处理和其他线程处理
2. 如果是本线程处理，只需等待异步接口调用时使用的stream内全部任务执行完毕后，拿出异步接口执行结果使用即可
3. 如果是其他线程处理，则需要定义一个额外的方法，专门用于处理异步调用的结果，这个“额外的方法”有个专业一点的名字，叫做“Callback”

什么是Callback?

- Callback 是一种特殊的函数，这个函数(a)被作为参数传给另一个函数(b)去调用。这样的函数(a)就是回调函数。
- Callback函数往往不是由定义方（我）来调用的，而是把函数指针传给谁就由谁来调。

```
Function a(string str){print "Hello " + str;}
```

```
Function b(void* (*func)())
```

```
{
```

```
    string str = "World." ;
```

```
    a(str );
```

```
}
```

```
b(a); // Hello World.
```


生活这么美好，我们为什么要使用Callback?

```
Function a(string str){print "Hello " + str;}
Function b(void* (*func)())
{
    string str = "World." ;
    a(str );
}
b(a); // Hello World.
```

首先要明确一个问题，那就是当我们提到传参的时候，我们传给一个函数的参数到底指代的是什么呢？
是值吗？

只能是值吗？

当我们传递指针给函数的时候，这个指针指向的内存存储的一定是一个数值吗？

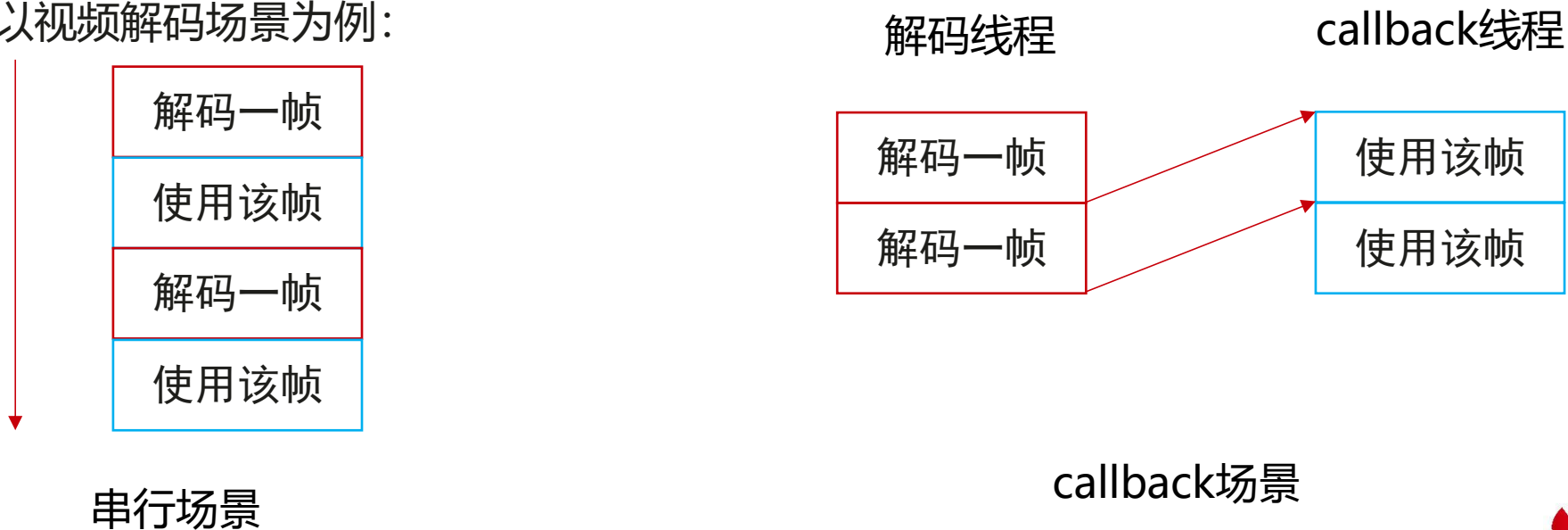
能不能是一个函数呢？

如果答案是能，那我们就可以把函数/方法/行为（a）当做参数传递给另一个函数（b），由b来决定对这个特殊的参数的使用。

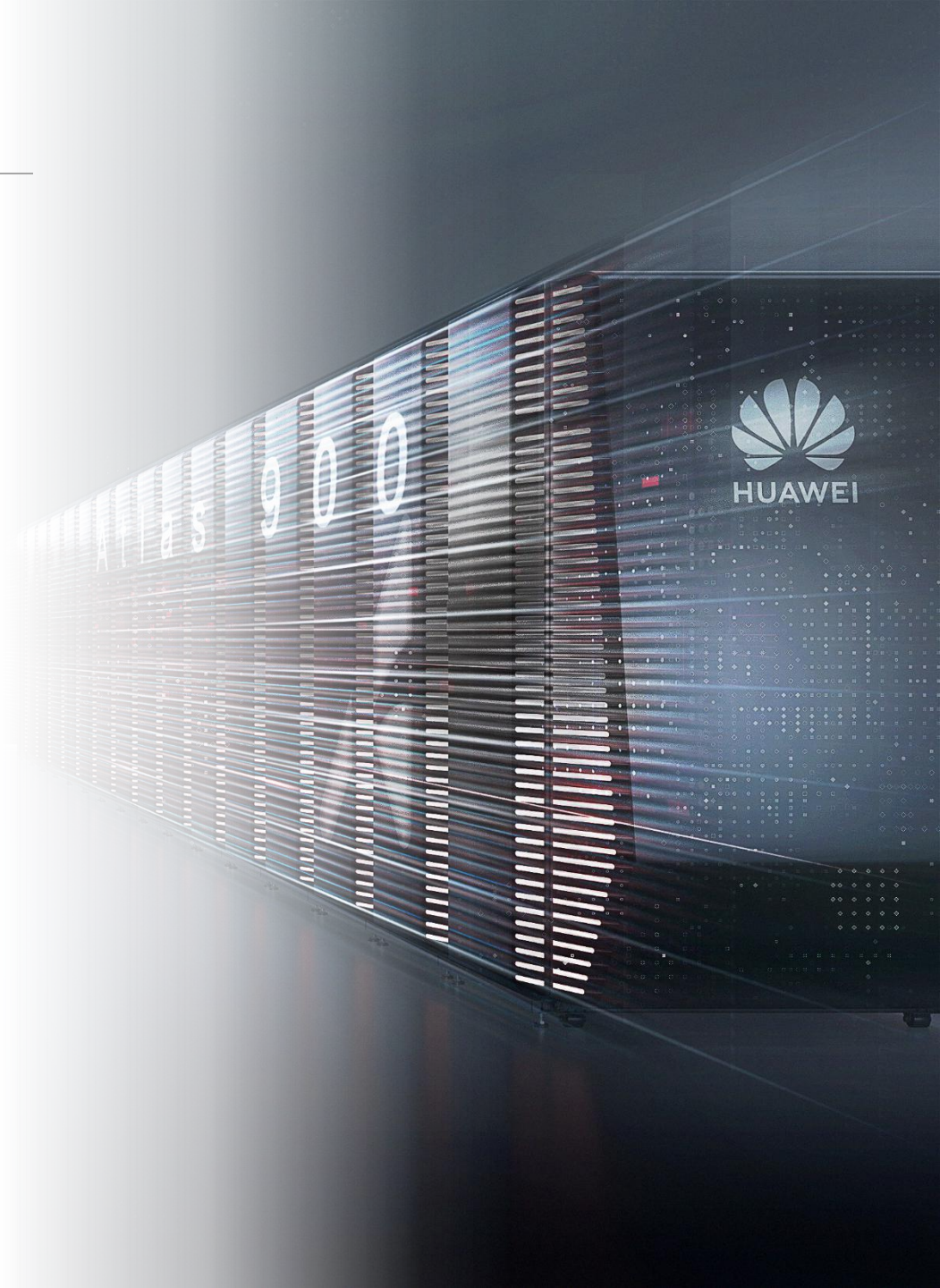


生活这么美好，我们为什么要使用Callback?

- 使用Callback的好处至少有以下2个：
- 1. 因为可以把数据生成方法（比如视频解码，结果是很多图像帧）和处理方法（对一帧一帧图片进行推理）分开并灵活组合。
- 2. 从某种意义上理解，可以达成更高的性能。其实是我们这个场景中，Callback一定要用另一个线程去执行，线程的并行能够提升性能，Callback是促成了这一点。
- 以视频解码场景为例：



- 1 同步等待基本概念**
- 2 典型的同步等待流程**
- 3 同步等待相关接口**



Event相关接口

接口名	说明
aclrtCreateEvent	创建一个Event。
aclrtDestroyEvent	销毁一个Event，只能销毁通过aclrtCreateEvent接口创建的Event，同步接口。销毁某个Event时，用户需确保等待aclrtSynchronizeEvent接口或aclrtStreamWaitEvent接口涉及的任务都结束后，再销毁。
aclrtRecordEvent	<p>在Stream中记录一个Event，同步接口。</p> <p>aclrtRecordEvent接口与aclrtStreamWaitEvent接口配合使用时，主要用于多Stream之间同步的场景，在调用aclrtRecordEvent接口时，系统内部会申请Event资源，在调用aclrtStreamWaitEvent接口之后，请及时调用aclrtResetEvent接口释放Event资源。</p> <p>接口调用顺序：aclrtCreateEvent-->aclrtRecordEvent-->aclrtStreamWaitEvent-->aclrtResetEvent</p>
aclrtResetEvent	<p>复位一个Event。用户需确保等待Stream中的任务都完成后，再复位Event。</p> <p>在复位Event时，涉及重置Event的状态，Event的状态是在调用aclrtRecordEvent接口时记录，因此在调用aclrtResetEvent接口前，需要先调用aclrtRecordEvent接口。</p> <p>接口调用顺序：aclrtCreateEvent-->aclrtRecordEvent-->aclrtResetEvent</p>
aclrtQueryEvent	<p>查询指定Event的状态。</p> <pre>typedef enum aclrtEventStatus { ACL_EVENT_STATUS_COMPLETE = 0, //完成 ACL_EVENT_STATUS_NOT_READY = 1, //未完成 ACL_EVENT_STATUS_RESERVED = 2, //预留 } aclrtEventStatus;</pre>
aclrtSynchronizeEvent	阻塞应用程序运行，等待Event完成。
aclrtEventElapsedTime	统计两个Event之间的耗时。
aclrtStreamWaitEvent	阻塞指定Stream的运行，直到指定的Event完成。



回调场景相关接口

- **(*aclrtCallback)**
- 函数功能：
 - 被调用的回调函数，函数体由用户自己定义。
- 函数原型：
 - `typedef void (*aclrtCallback)(void *userData)`
- 参数说明：

参数名	输入/输出	说明
userData	输入	传递给回调函数的用户数据。

回调场景相关接口

- **aclrtSubscribeReport**

- 函数功能:

- 指定处理Stream上回调函数的线程。同步接口。

- 函数原型:

- **aclError aclrtSubscribeReport(uint64_t threadId, aclrtStream stream)**

- 参数说明:

参数名	输入/输出	说明
threadId	输入	指定线程的ID。
stream	输入	指定Stream。

- =====割哥鸽歌隔=====

- **aclError aclrtUnSubscribeReport(uint64_t threadId, aclrtStream stream)**

- 取消线程注册，Stream上的回调函数不再由指定线程处理。同步接口。

回调场景相关接口

- **aclrtProcessReport**

- 函数功能:

- 等待指定时间后，触发回调处理，由aclrtSubscribeReport接口指定的线程处理回调。同步接口。
- 用户需新建一个线程，在线程函数内调用aclrtProcessReport接口。

- 函数原型:

- **aclError aclrtProcessReport(int32_t timeout)**

- 参数说明:

参数名	输入/输出	说明
timeout	输入	超时时间，单位为ms。 取值范围： -1：表示无限等待 大于0（不包含0）：表示等待的时间

回调场景相关接口

- **aclrtLaunchCallback**

- 函数功能:

- › 在Stream的任务队列中增加一个需要在Host/Device上执行的回调函数。异步接口。

- 函数原型:

- › **aclError aclrtLaunchCallback(aclrtCallback fn, void *userData, aclrtCallbackBlockType blockType, aclrtStream stream)**

- 参数说明:

参数名	输入/输出	说明
fn	输入	指定要增加的回调函数。
userData	输入	待传递给回调函数的用户数据。
blockType	输入	指定回调函数调用是否阻塞Device运行。 typedef enum aclrtCallbackBlockType { ACL_CALLBACK_NO_BLOCK, //非阻塞 ACL_CALLBACK_BLOCK, //阻塞 } aclrtCallbackBlockType;
stream	输入	指定Stream。

Thank you.

昇腾开发者社区



<http://ascend.huawei.com>

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

**Bring digital to every person, home, and
organization for a fully connected,
intelligent world.**

**Copyright©2020 Huawei Technologies Co., Ltd.
All Rights Reserved.**

The information in this document may contain
predictive
statements including, without limitation, statements
regarding
the future financial and operating results, future
product
portfolio, new technology, etc. There are a number of
factors that
could cause actual results and developments to differ
materially
from those expressed or implied in the predictive
statements.
Therefore, such information is provided for reference
purpose
only and constitutes neither an offer nor an

