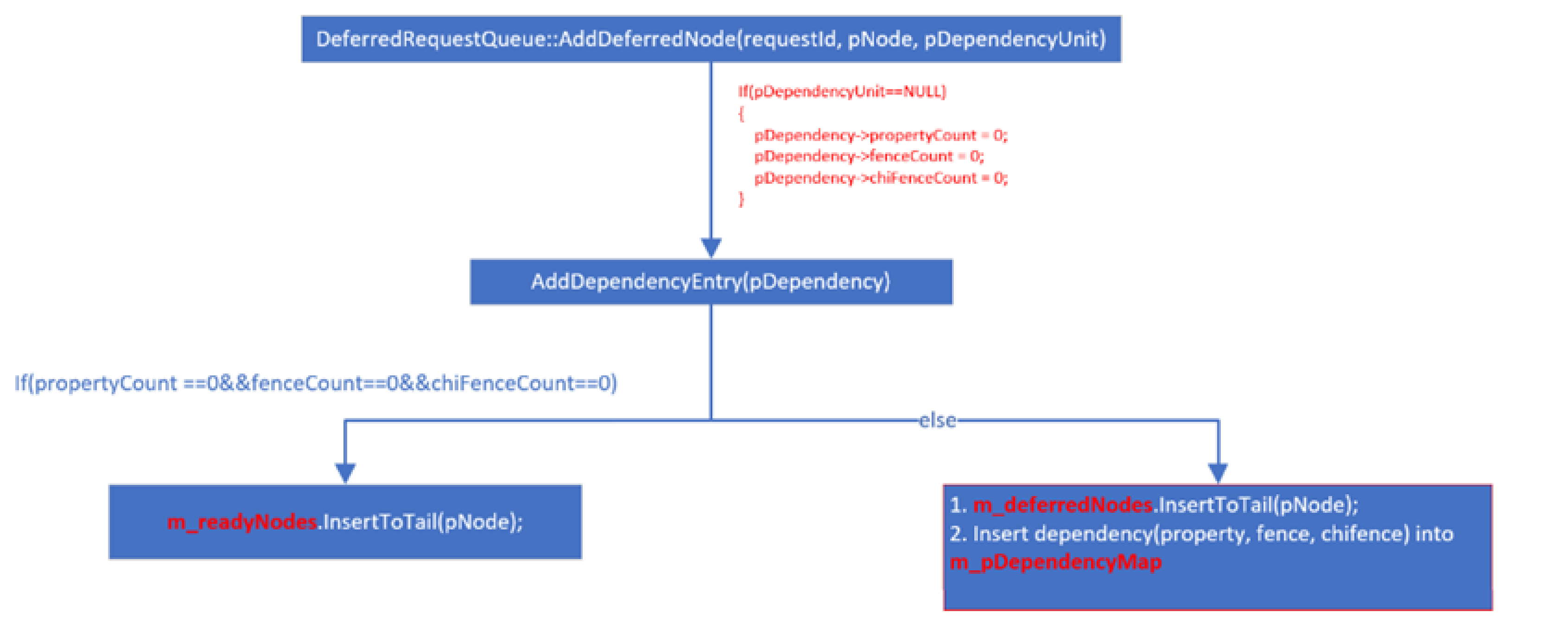
DRQ什么时候调度Node去填写dependency

Pipeline调度Node的sequenceId 0执行

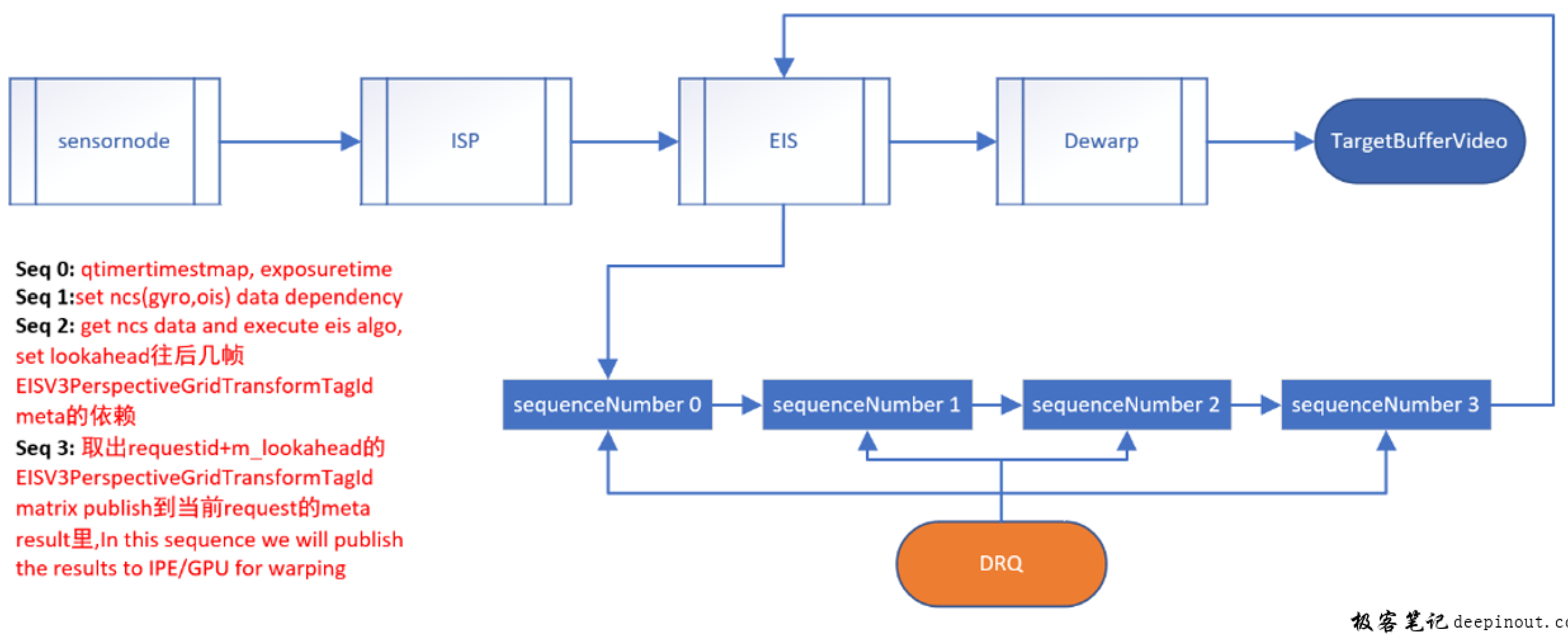
```
Pipeline::ProcessRequest()
{
    for (UINT nodeIndex = 0; nodeIndex  < m_orderedNodeCount ; nodeIndex++)
        m_pDeferredRequestQueue->AddDeferredNode(requestId, m_ppOrderedNodes[nodeIndex], NULL);//最后一个参数pDependencyUnit为NULL
    m_pDeferredRequestQueue->DispatchReadyNodes();
}
```

> 每个node seq id 0基本都是填写自己的depependency

添加deferred node流程

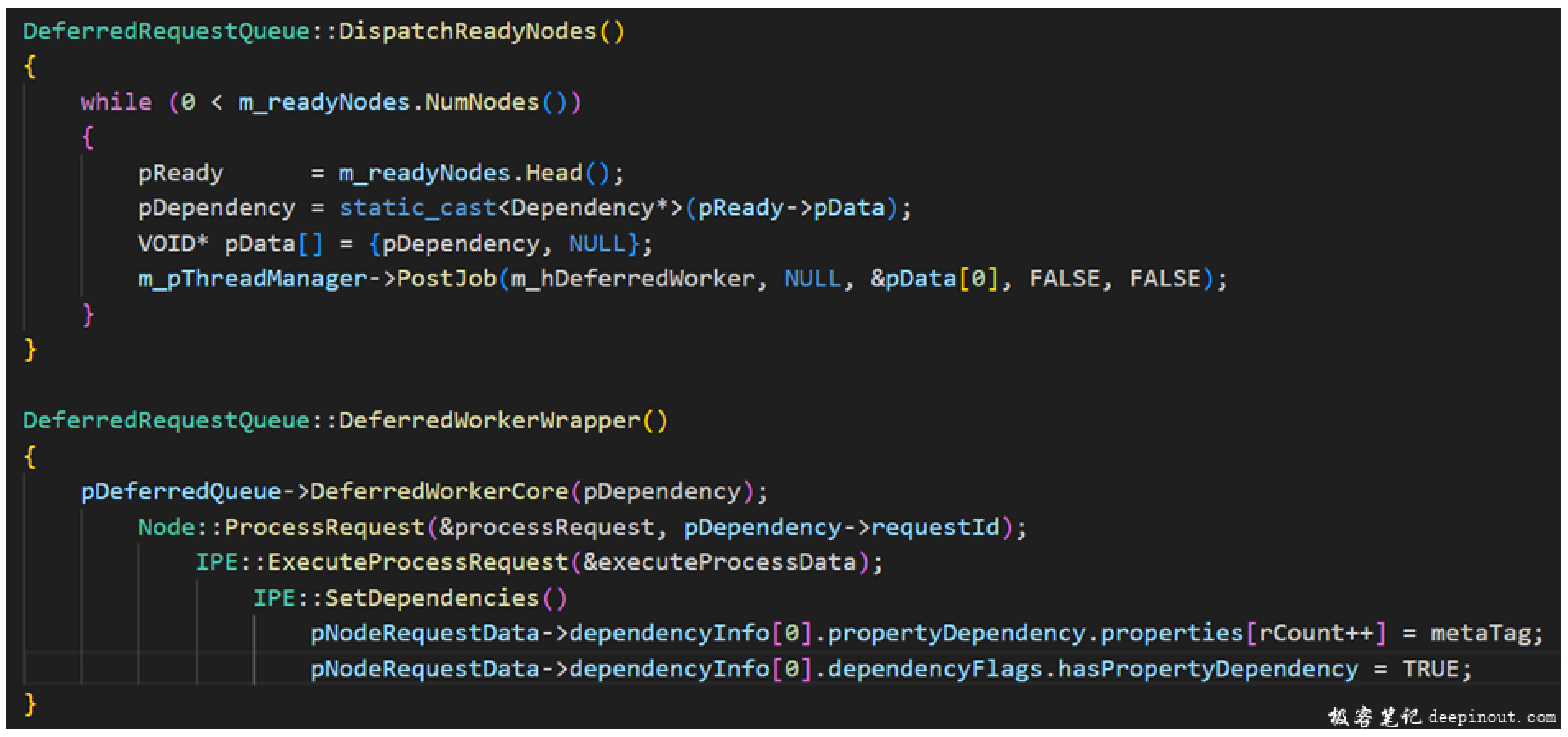

DeferredRequestQueue::AddDeferredNode(requestId, pNode, pDependencyUnit)
If(pDependencyUnit==NULL)
{
pDependency->propertyCount = 0;
pDependency->fenceCount = 0;
pDependency->chiFenceCount = 0;
}
AddDependencyEntry(pDependency)
If(propertyCount ==0&&fenceCount==0&&chiFenceCount==0)
else
m_readyNodes.InsertToTail(pNode);
1. m_deferredNodes.InsertToTail(pNode);
2. Insert dependency(property, fence, chifence) into m_pDependencyMap

Node内部的ProcessSequenceId


sensornode
ISP
EIS
Dewarp
TargetBufferVideo
Seq 0: qtimertimestmap, exposuretime
Seq 1:set ncs(gyro,ois) data dependency
Seq 2: get ncs data and execute eis algo, set lookahead往后几帧 EISV3PerspectiveGridTransformTagId meta的依赖
Seq 3: 取出requestid+m_lookahead的 EISV3PerspectiveGridTransformTagId matrix publish到当前request的meta result里,In this sequence we will publish the results to IPE/GPU for warping
sequenceNumber 0
sequenceNumber 1
sequenceNumber 2
sequenceNumber 3
DRQ

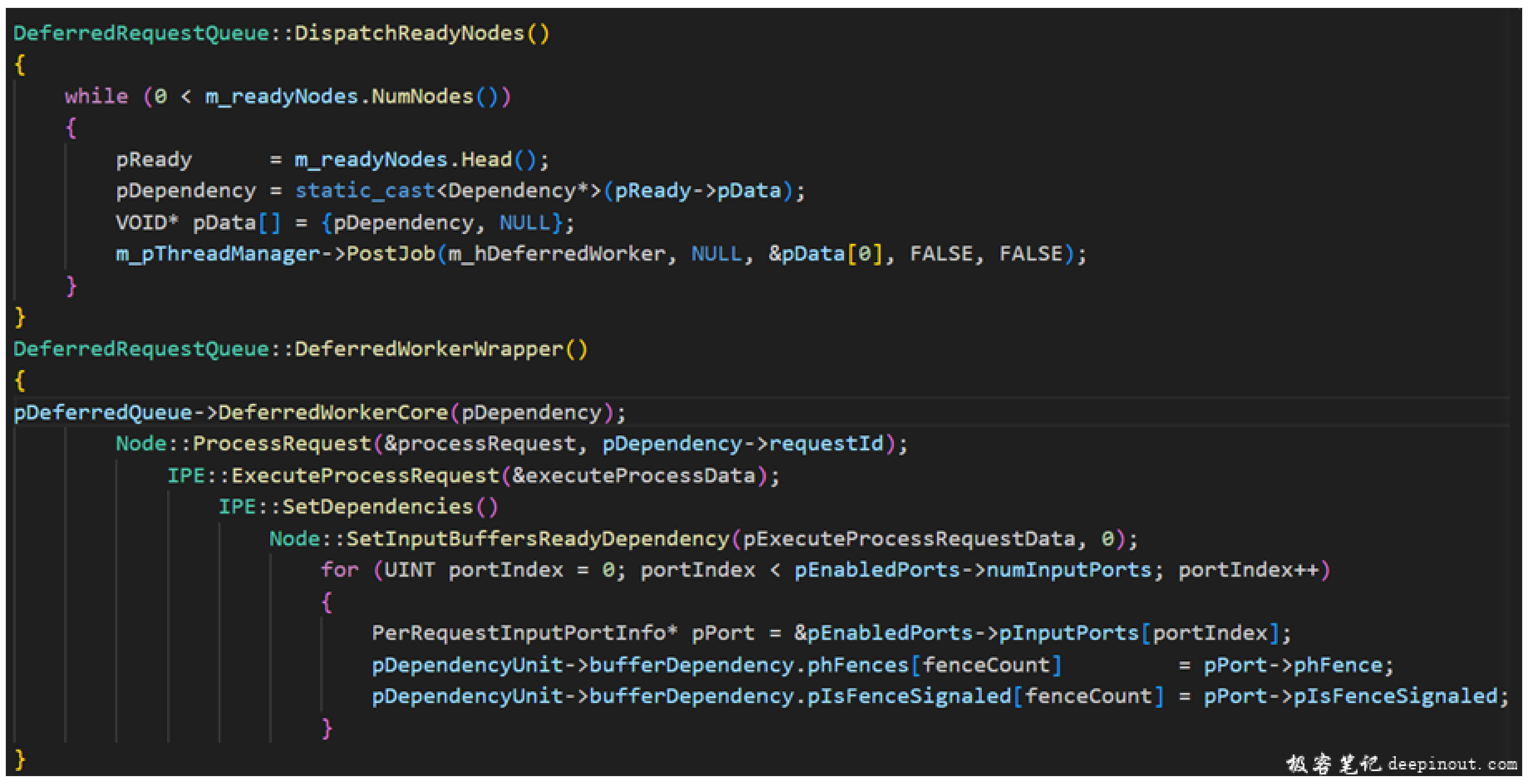
如何填写dependency

如何填写property dependency



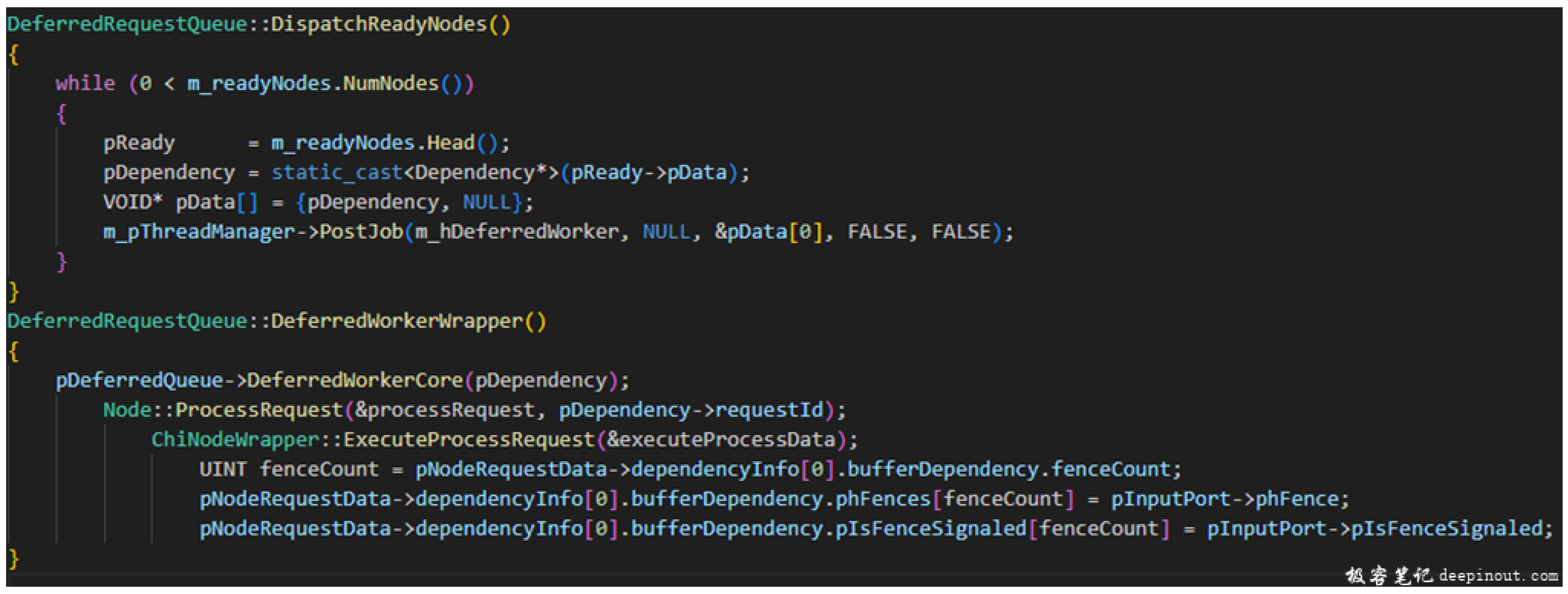
```
DeferredRequestQueue::DispatchReadyNodes()
{
    while (0 < m_readyNodes.NumNodes())
    {
        pReady      = m_readyNodes.Head();
        pDependency = static_cast<Dependency*>(pReady->pData);
        VOID* pData[] = {pDependency, NULL};
        m_pThreadManager->PostJob(m_hDeferredWorker, NULL, &pData[0], FALSE, FALSE);
    }
}

DeferredRequestQueue::DeferredWorkerWrapper()
{
    pDeferredQueue->DeferredWorkerCore(pDependency);
        Node::ProcessRequest(&processRequest, pDependency->requestId);
            IPE::ExecuteProcessRequest(&executeProcessData);
                IPE::SetDependencies()
                    pNodeRequestData->dependencyInfo[0].propertyDependency.properties[rCount++] = metaTag;
                    pNodeRequestData->dependencyInfo[0].dependencyFlags.hasPropertyDependency = TRUE;
}
```

如何填写fence dependency(camxnode)



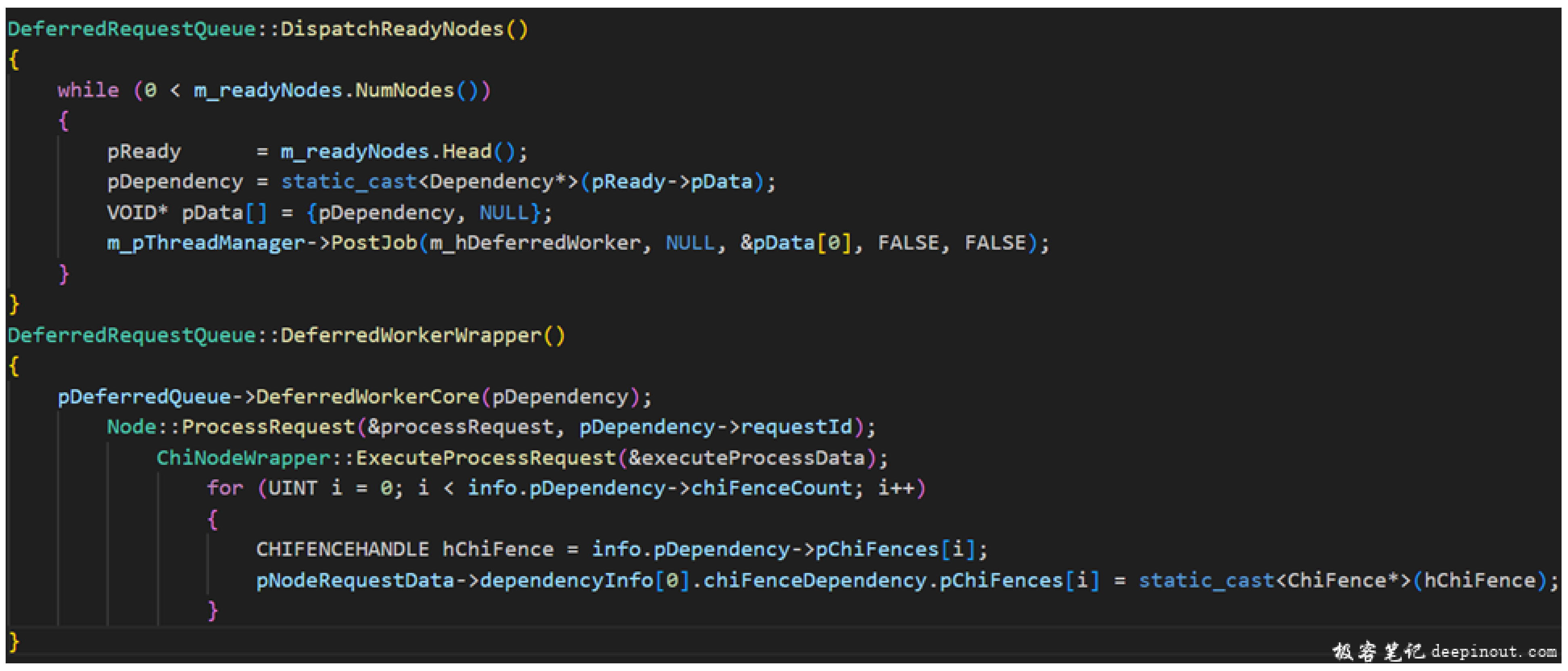
```
DeferredRequestQueue::DispatchReadyNodes()
{
    while (0 < m_readyNodes.NumNodes())
    {
        pReady      = m_readyNodes.Head();
        pDependency = static_cast<Dependency*>(pReady->pData);
        VOID* pData[] = {pDependency, NULL};
        m_pThreadManager->PostJob(m_hDeferredWorker, NULL, &pData[0], FALSE, FALSE);
    }
}
DeferredRequestQueue::DeferredWorkerWrapper()
{
pDeferredQueue->DeferredWorkerCore(pDependency);
        Node::ProcessRequest(&processRequest, pDependency->requestId);
            IPE::ExecuteProcessRequest(&executeProcessData);
                IPE::SetDependencies()
                    Node::SetInputBuffersReadyDependency(pExecuteProcessRequestData, 0);
                        for (UINT portIndex = 0; portIndex < pEnabledPorts->numInputPorts; portIndex++)
                        {
                            PerRequestInputPortInfo* pPort = &pEnabledPorts->pInputPorts[portIndex];
                            pDependencyUnit->bufferDependency.phFences[fenceCount]         = pPort->phFence;
                            pDependencyUnit->bufferDependency.pIsFenceSignaled[fenceCount] = pPort->pIsFenceSignaled;
                        }
}
```

如何填写fence dependency(chi node)



```
DeferredRequestQueue::DispatchReadyNodes()
{
    while (0 < m_readyNodes.NumNodes())
    {
        pReady      = m_readyNodes.Head();
        pDependency = static_cast<Dependency*>(pReady->pData);
        VOID* pData[] = {pDependency, NULL};
        m_pThreadManager->PostJob(m_hDeferredWorker, NULL, &pData[0], FALSE, FALSE);
    }
}
DeferredRequestQueue::DeferredWorkerWrapper()
{
    pDeferredQueue->DeferredWorkerCore(pDependency);
        Node::ProcessRequest(&processRequest, pDependency->requestId);
            ChiNodeWrapper::ExecuteProcessRequest(&executeProcessData);
                UINT fenceCount = pNodeRequestData->dependencyInfo[0].bufferDependency.fenceCount;
                pNodeRequestData->dependencyInfo[0].bufferDependency.phFences[fenceCount] = pInputPort->phFence;
                pNodeRequestData->dependencyInfo[0].bufferDependency.pIsFenceSignaled[fenceCount] = pInputPort->pIsFenceSignaled;
}
```

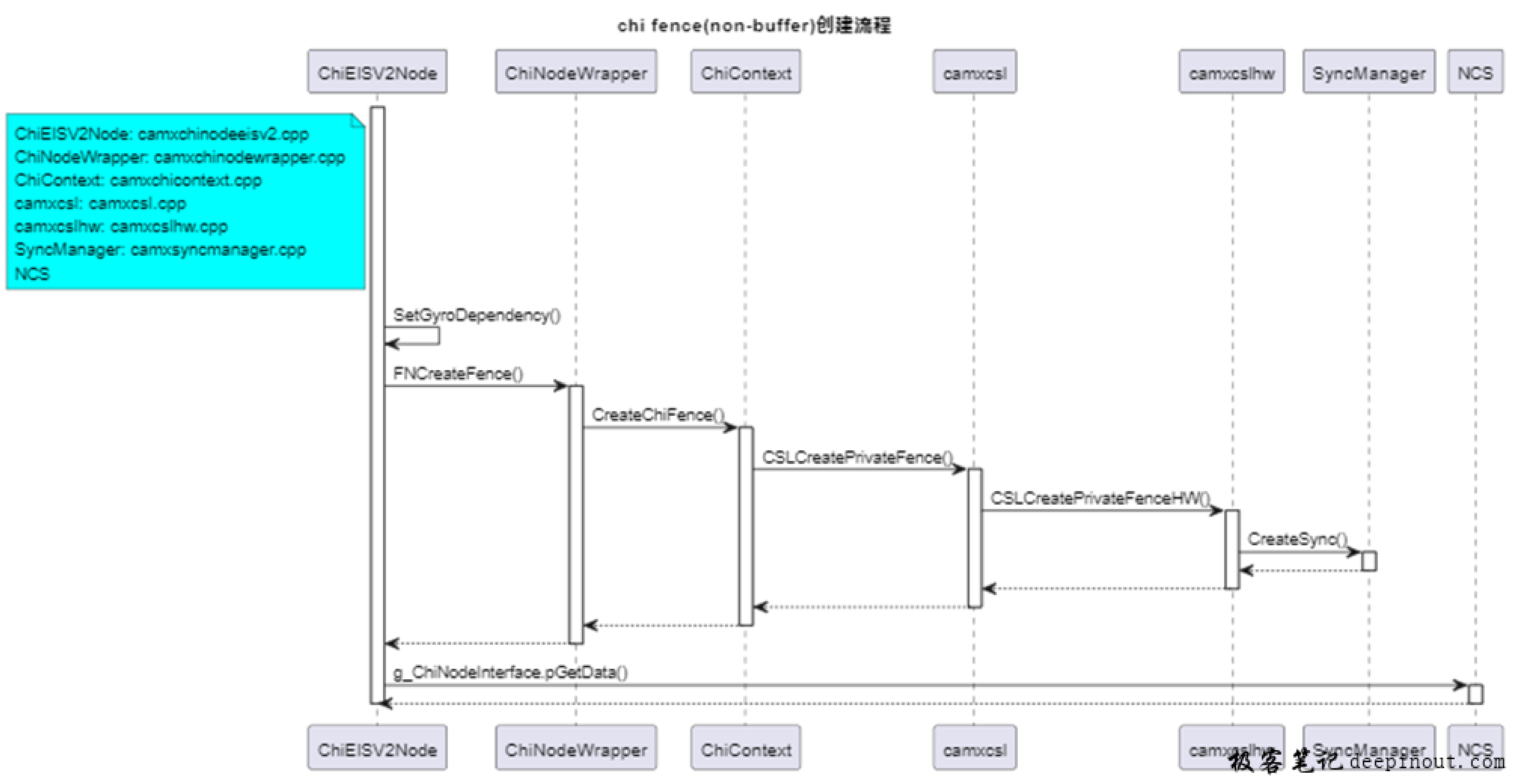
如何填写chi fence dependency



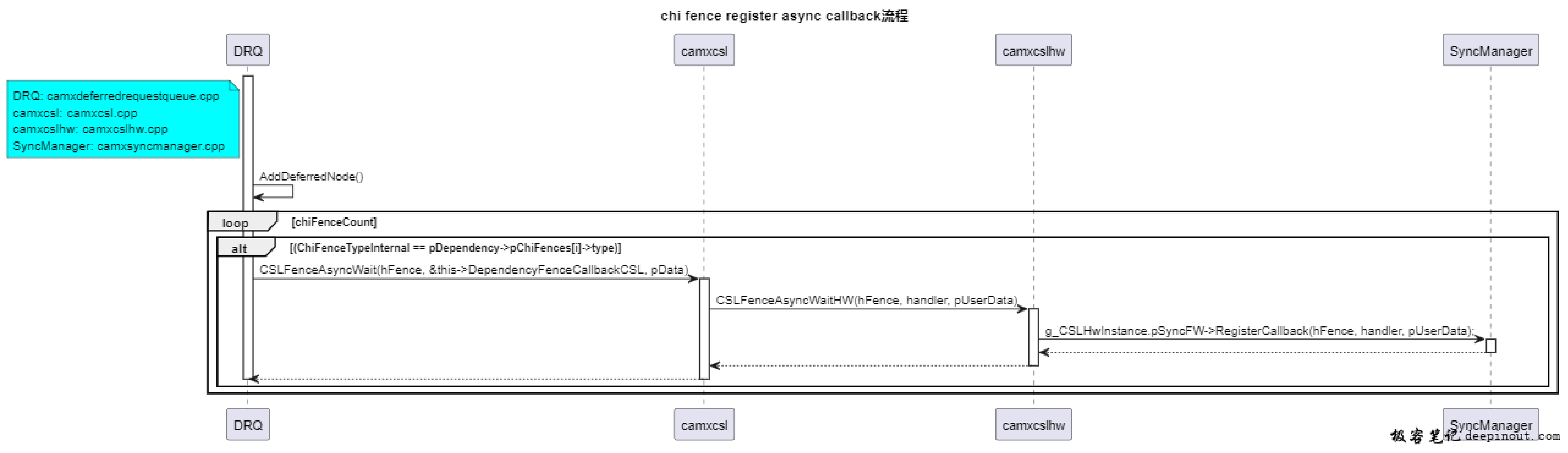
```
DeferredRequestQueue::DispatchReadyNodes()
{
    while (0 < m_readyNodes.NumNodes())
    {
        pReady      = m_readyNodes.Head();
        pDependency = static_cast<Dependency*>(pReady->pData);
        VOID* pData[] = {pDependency, NULL};
        m_pThreadManager->PostJob(m_hDeferredWorker, NULL, &pData[0], FALSE, FALSE);
    }
}
DeferredRequestQueue::DeferredWorkerWrapper()
{
    pDeferredQueue->DeferredWorkerCore(pDependency);
        Node::ProcessRequest(&processRequest, pDependency->requestId);
            ChiNodeWrapper::ExecuteProcessRequest(&executeProcessData);
                for (UINT i = 0; i < info.pDependency->chiFenceCount; i++)
                {
                    CHIFENCEHANDLE hChiFence = info.pDependency->pChiFences[i];
                    pNodeRequestData->dependencyInfo[0].chiFenceDependency.pChiFences[i] = static_cast<ChiFence*>(hChiFence);
                }
}
```

Chi Fence(non-buffer) API调用详解

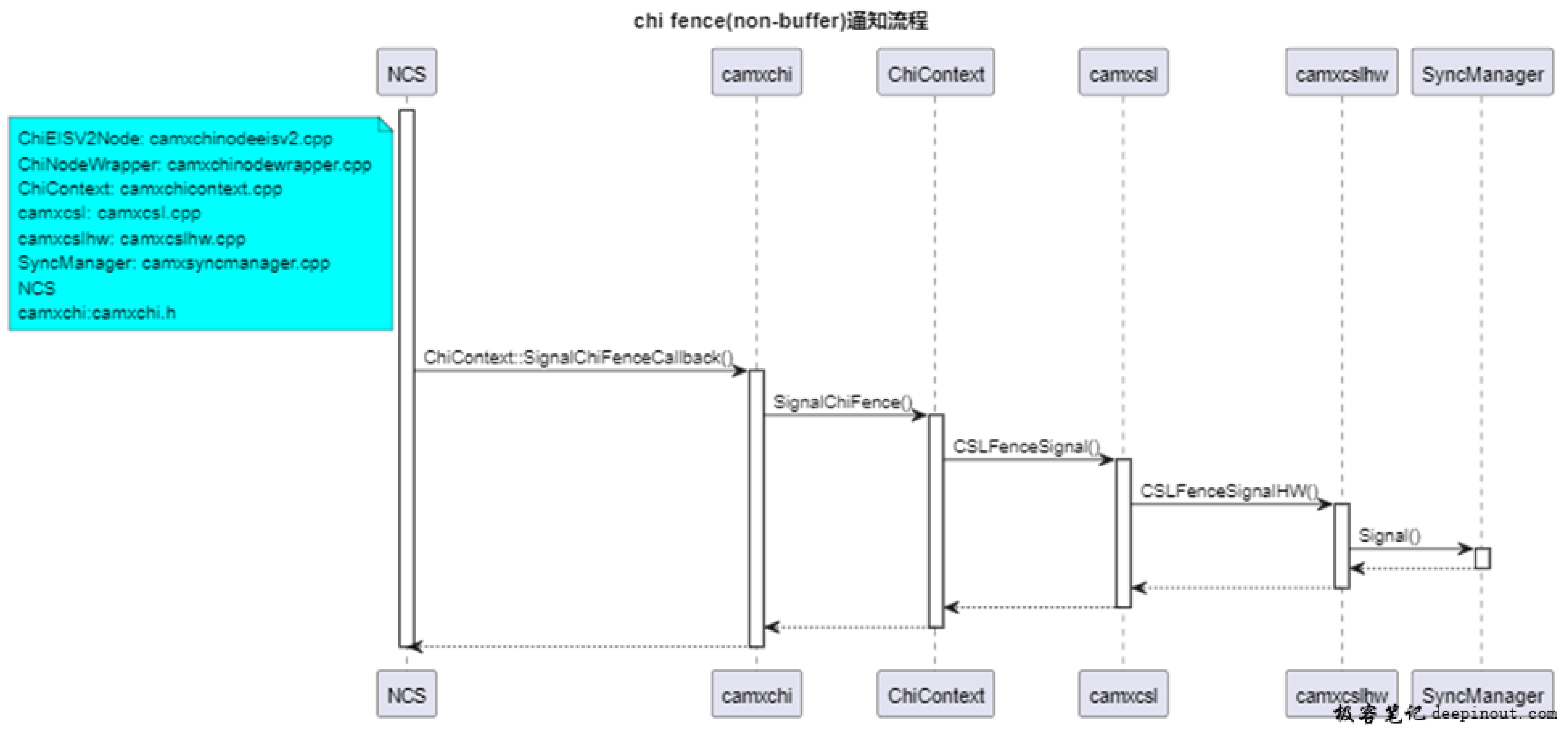
Chi Fence(non-buffer) Create举例(EISV2)



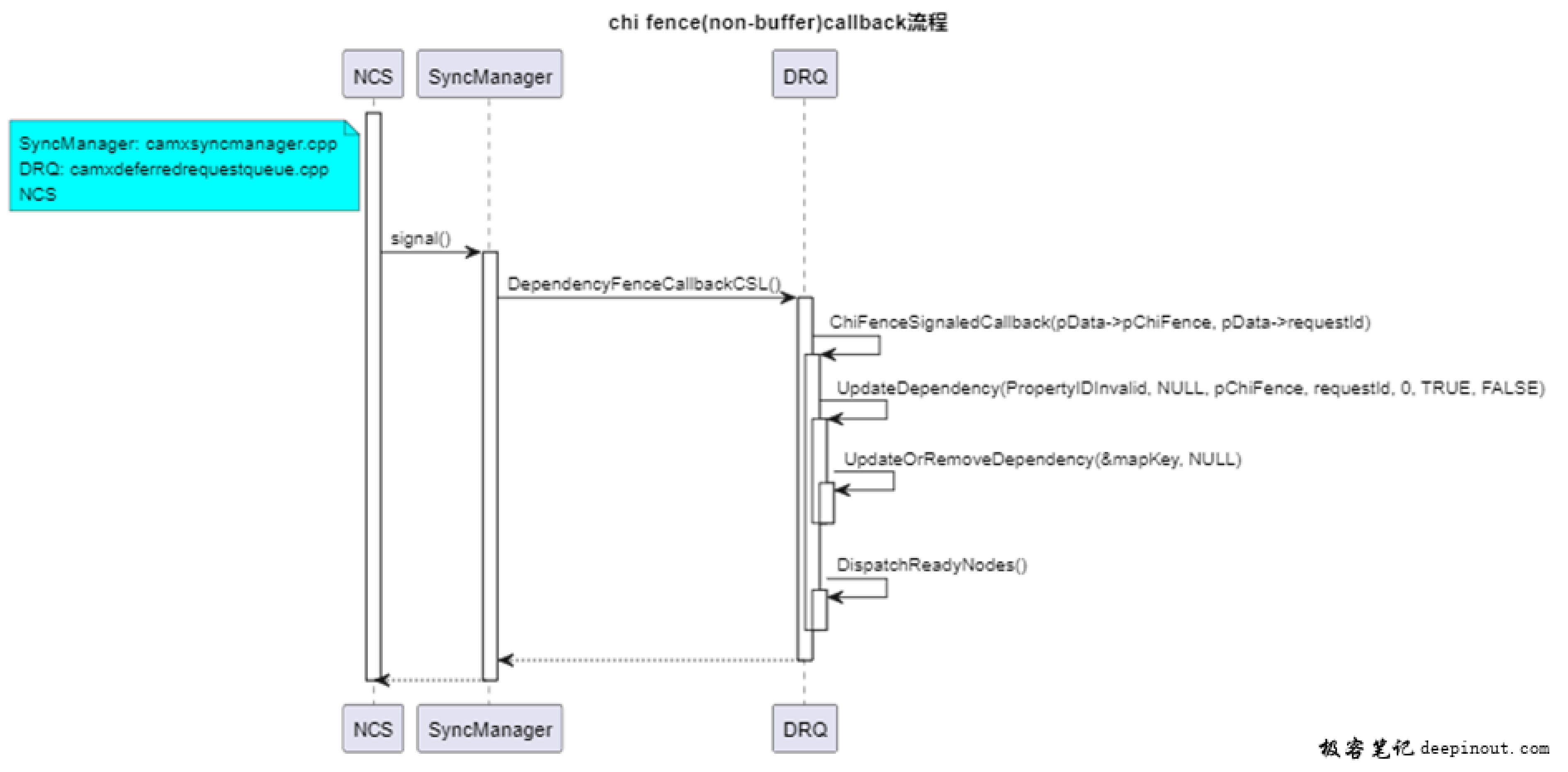
Chi Fence(non-buffer) 注册async callback

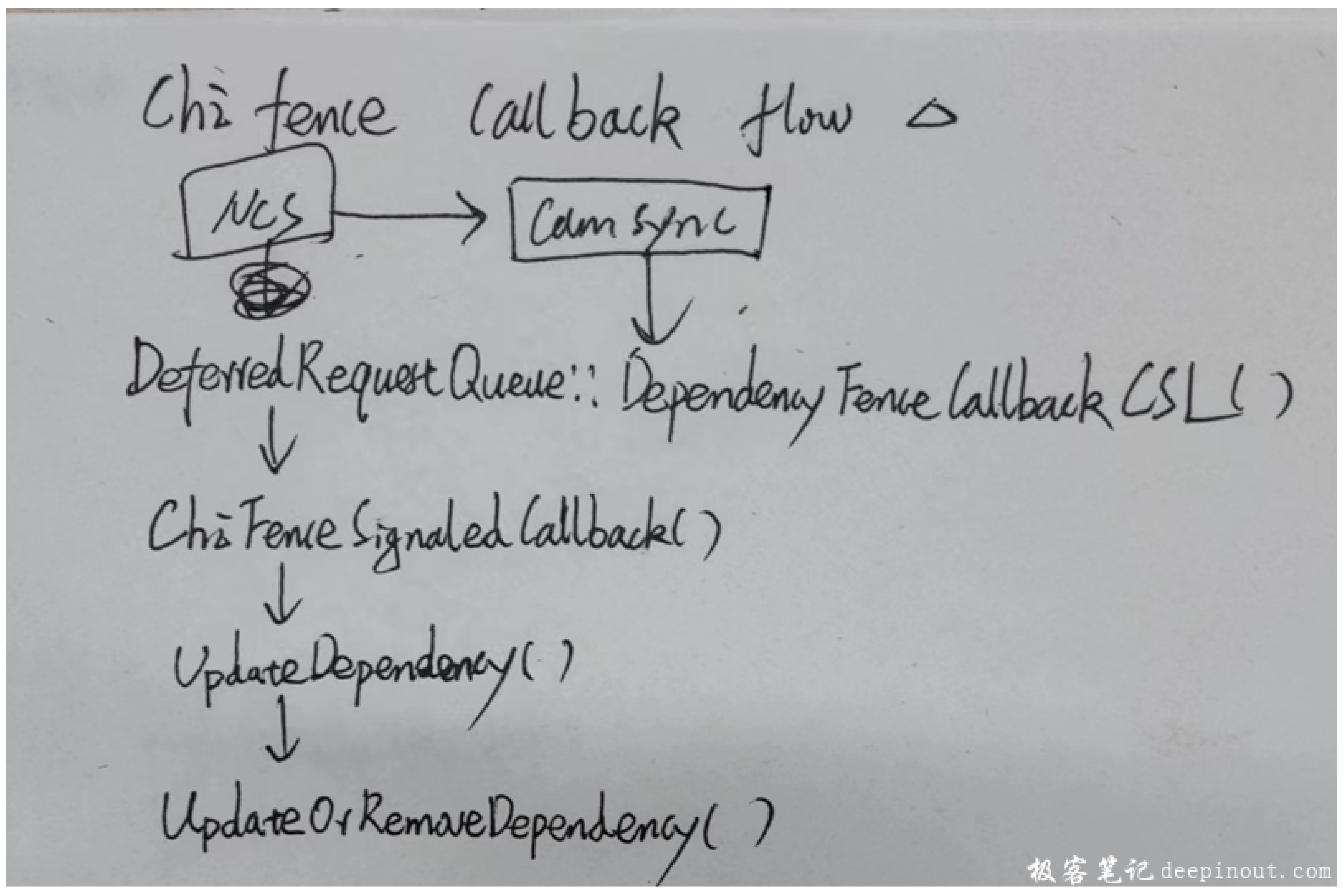

chi fence register async callback流程
DRQ
camxcsl
camxcslhw
SyncManager
DRQ: camxdeferredrequestqueue.cpp
camxcsl: camxcsl.cpp
camxcslhw: camxcslhw.cpp
SyncManager: camxsyncmanager.cpp
AddDeferredNode()
loop
[chiFenceCount]
alt
[(ChiFenceTypeInternal == pDependency->pChiFences[i]->type)]
CSLFenceAsyncWait(hFence, &this->DependencyFenceCallbackCSL, pData)
CSLFenceAsyncWaitHW(hFence, handler, pUserData)
g_CSLHwInstance.pSyncFW->RegisterCallback(hFence, handler, pUserData);
极客笔记 deepinout.com

Chi Fence(non-buffer) Signal举例(EISV2)


chi fence(non-buffer)通知流程
NCS
camxchi
ChiContext
camxcsl
camxcslhw
SyncManager
ChiEISV2Node: camxchinodeeisv2.cpp
ChiNodeWrapper: camxchinodewrapper.cpp
ChiContext: camxchicontext.cpp
camxcsl: camxcsl.cpp
camxcslhw: camxcslhw.cpp
SyncManager: camxsyncmanager.cpp
NCS
camxchi:camxchi.h
ChiContext::SignalChiFenceCallback()
SignalChiFence()
CSLFenceSignal()
CSLFenceSignalHW()
Signal()
极客笔记 deepinout.com

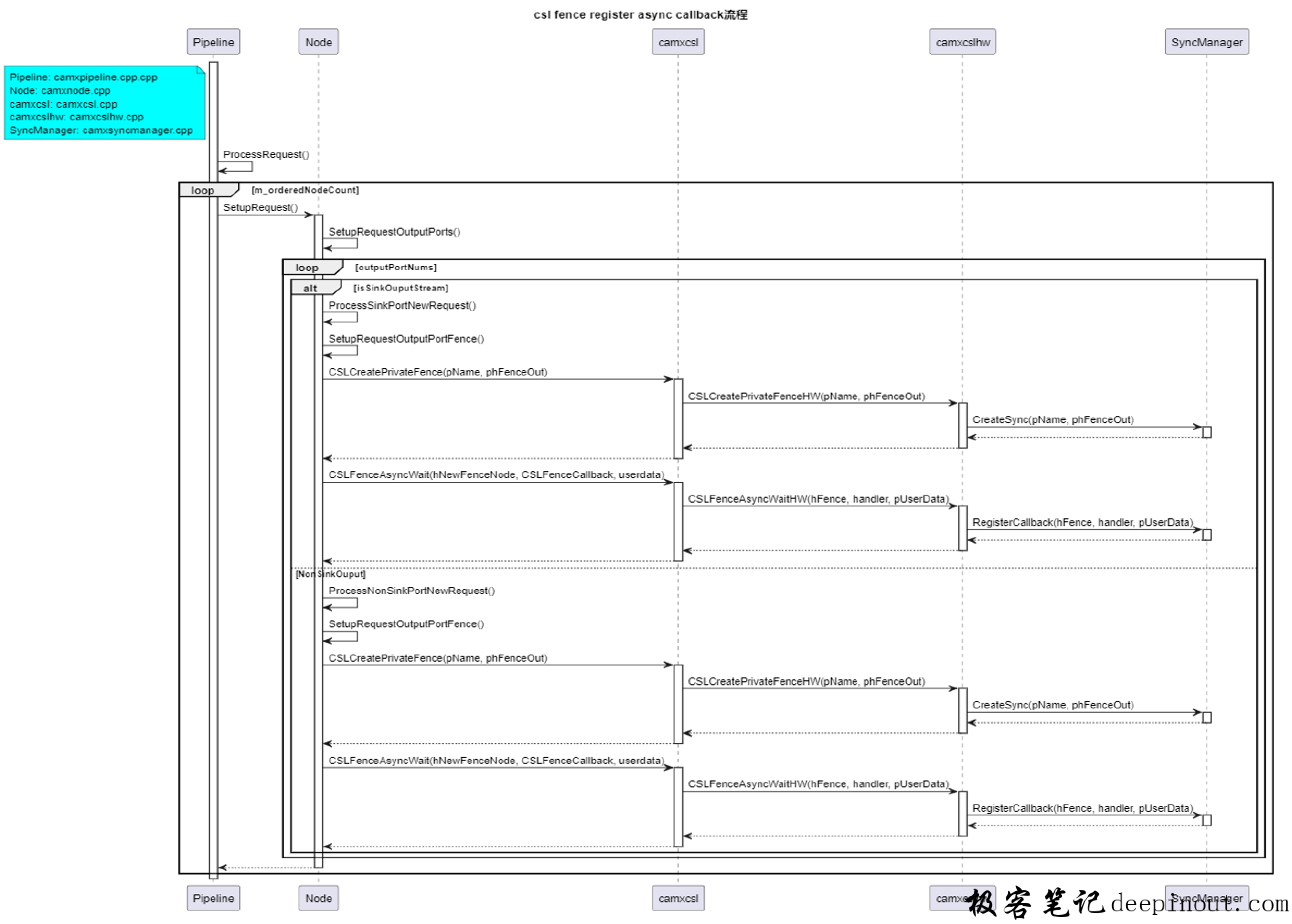
CHI Fence (non-buffer) callback flow

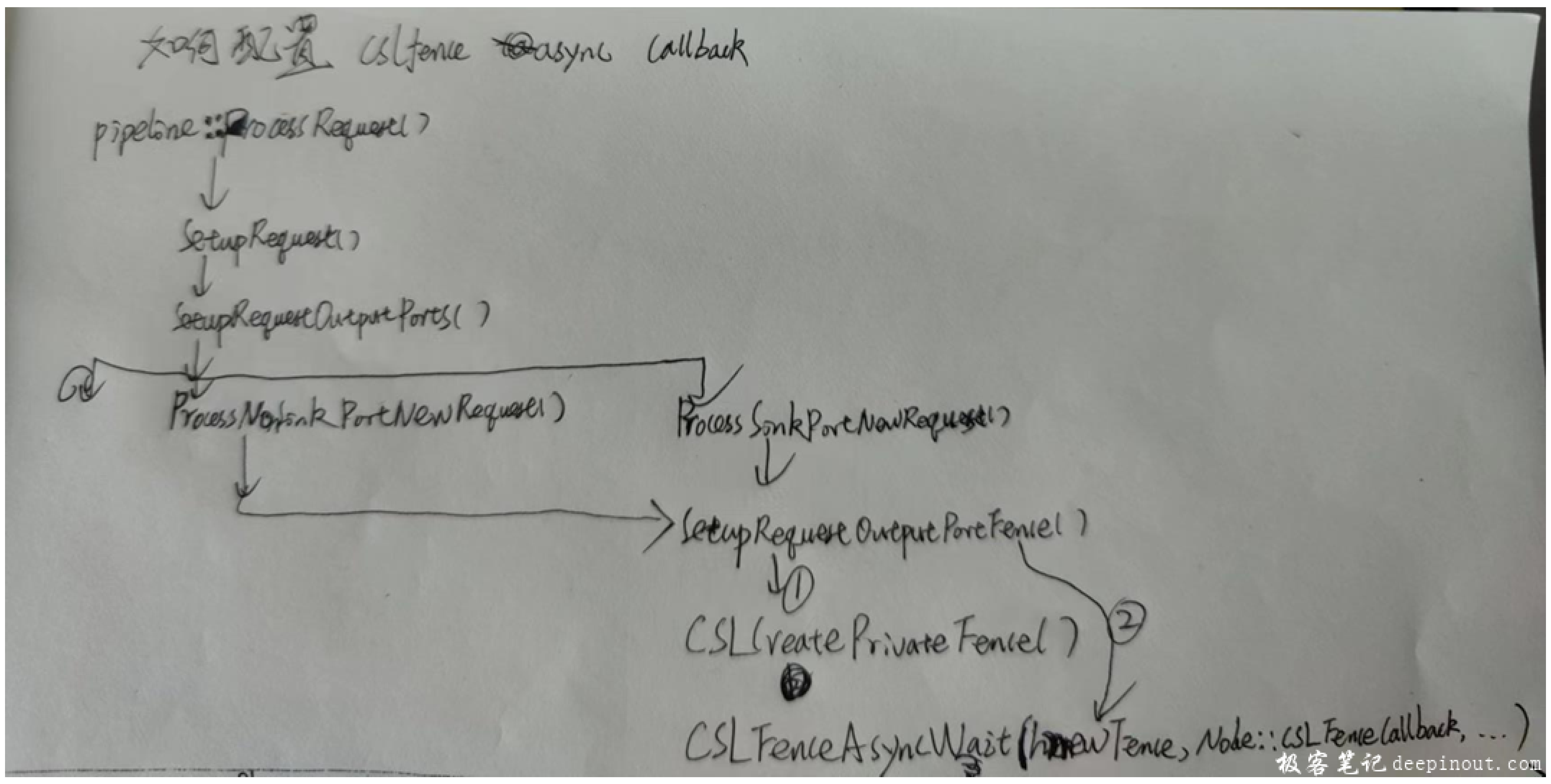

chi fence(non-buffer)callback流程
NCS
SyncManager
DRQ
SyncManager: camxsyncmanager.cpp
DRQ: camxdeferredrequestqueue.cpp
NCS
signal()
DependencyFenceCallbackCSL()
ChiFenceSignaledCallback(pData->pChiFence, pData->requestId)
UpdateDependency(PropertyIDInvalid, NULL, pChiFence, requestId, 0, TRUE, FALSE)
UpdateOrRemoveDependency(&mapKey, NULL)
DispatchReadyNodes()
极客笔记 deepinout.com

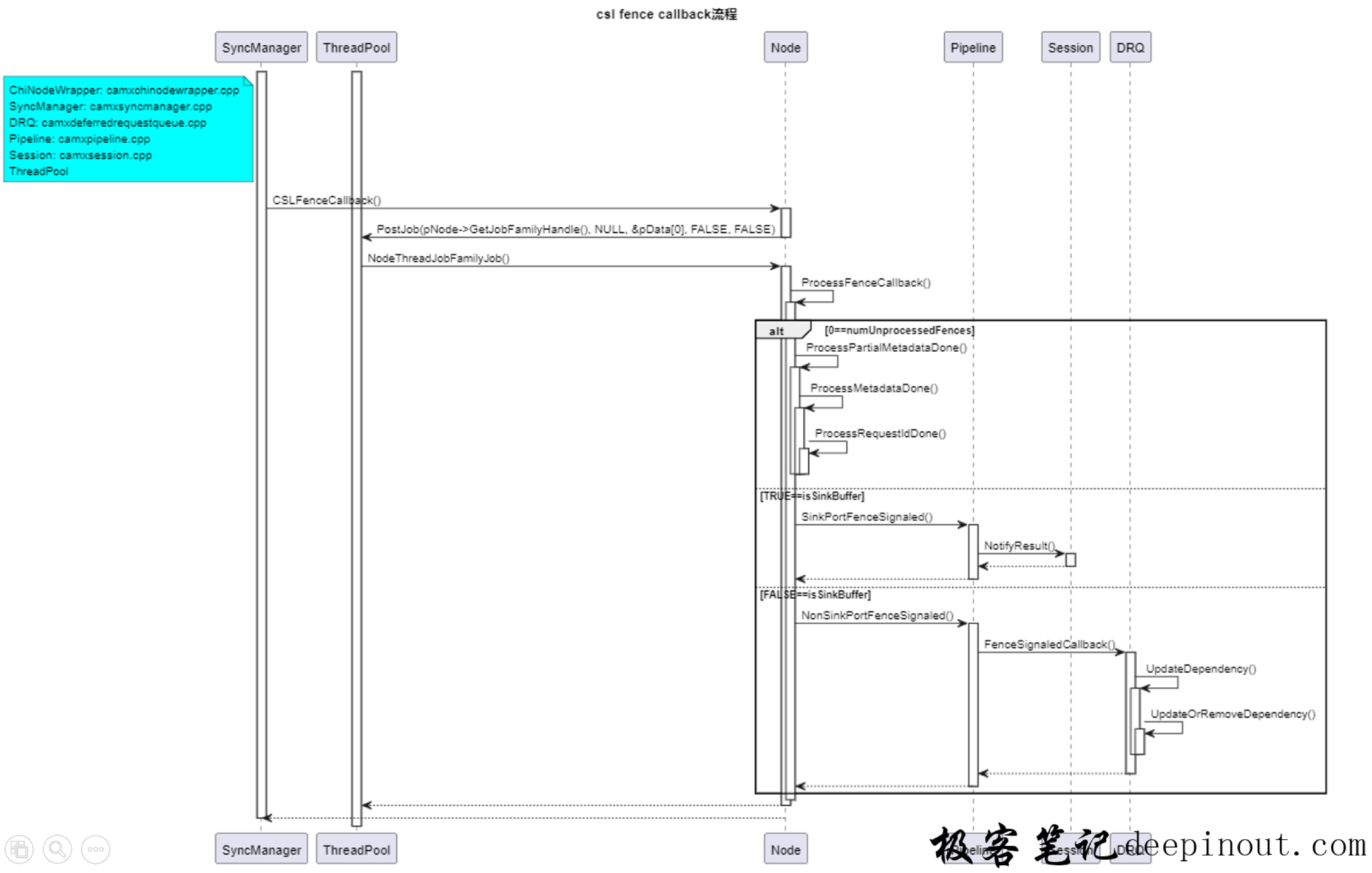

Chi fence Callback flow
NCS
Cam sync
DeferredRequestQueue:: DependencyFenceCallbackCSL( )
ChiFenceSignaledCallback( )
UpdateDependency( )
UpdateOrRemoveDependency( )
极客笔记 deepinout.com

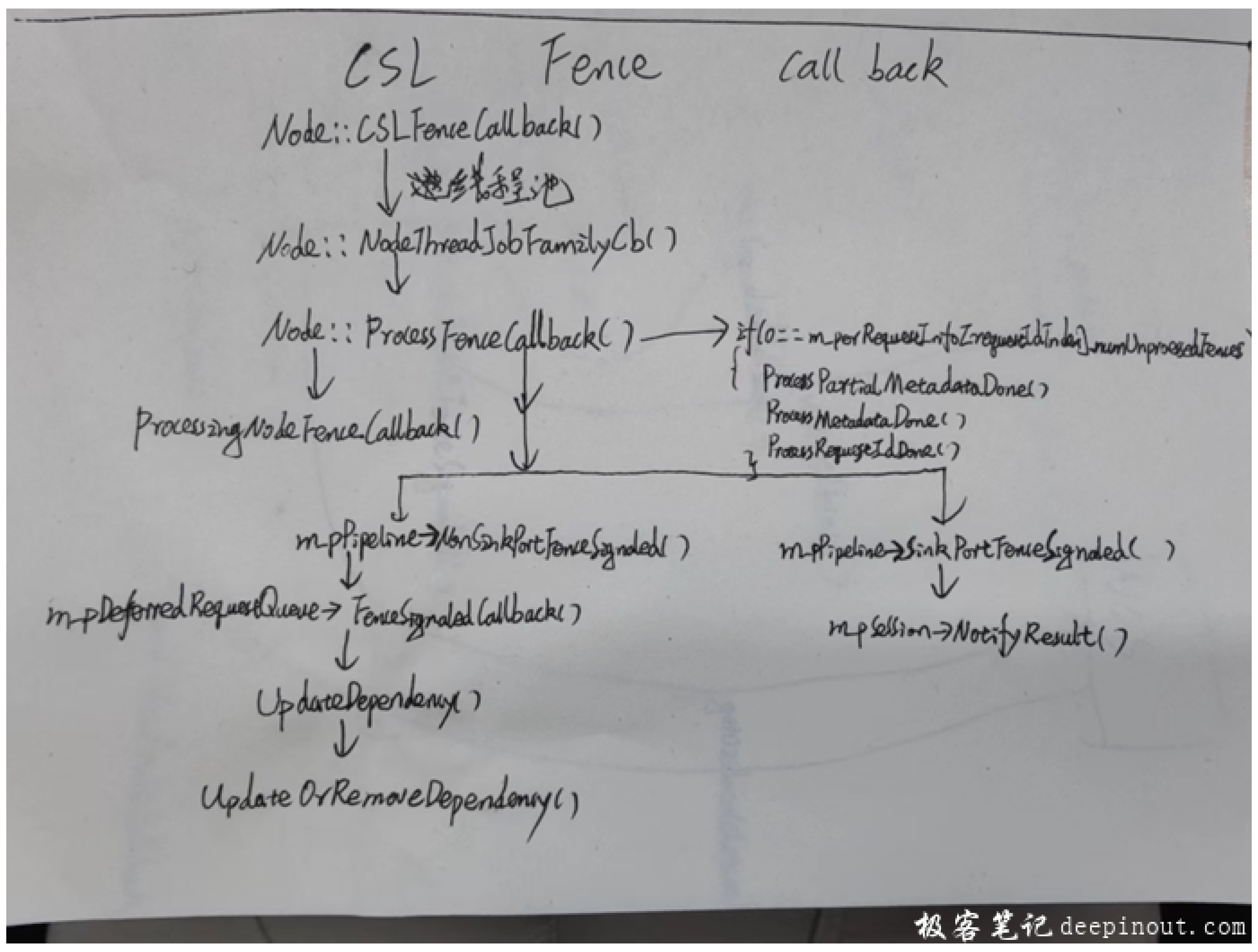
CSL Fence(buffer) async/callback详解

CSL Fence 注册async callback


csl fence register async callback流程
Pipeline
Node
camxcsl
camxcslhw
SyncManager
Pipeline: camxpipeline.cpp.cpp
Node: camxnode.cpp
camxcsl: camxcsl.cpp
camxcslhw: camxcslhw.cpp
SyncManager: camxsyncmanager.cpp
ProcessRequest()
loop [m_orderedNodeCount]
SetupRequest()
SetupRequestOutputPorts()
loop [outputPortNums]
alt [isSinkOuputStream]
ProcessSinkPortNewRequest()
SetupRequestOutputPortFence()
CSLCreatePrivateFence(pName, phFenceOut)
CSLCreatePrivateFenceHW(pName, phFenceOut)
CreateSync(pName, phFenceOut)
CSLFenceAsyncWait(hNewFenceNode, CSLFenceCallback, userdata)
CSLFenceAsyncWaitHW(hFence, handler, pUserData)
RegisterCallback(hFence, handler, pUserData)
[NonSinkOuput]
ProcessNonSinkPortNewRequest()
SetupRequestOutputPortFence()
CSLCreatePrivateFence(pName, phFenceOut)
CSLCreatePrivateFenceHW(pName, phFenceOut)
CreateSync(pName, phFenceOut)
CSLFenceAsyncWait(hNewFenceNode, CSLFenceCallback, userdata)
CSLFenceAsyncWaitHW(hFence, handler, pUserData)
RegisterCallback(hFence, handler, pUserData)
Pipeline
Node
camxcsl
camx
SyncManager

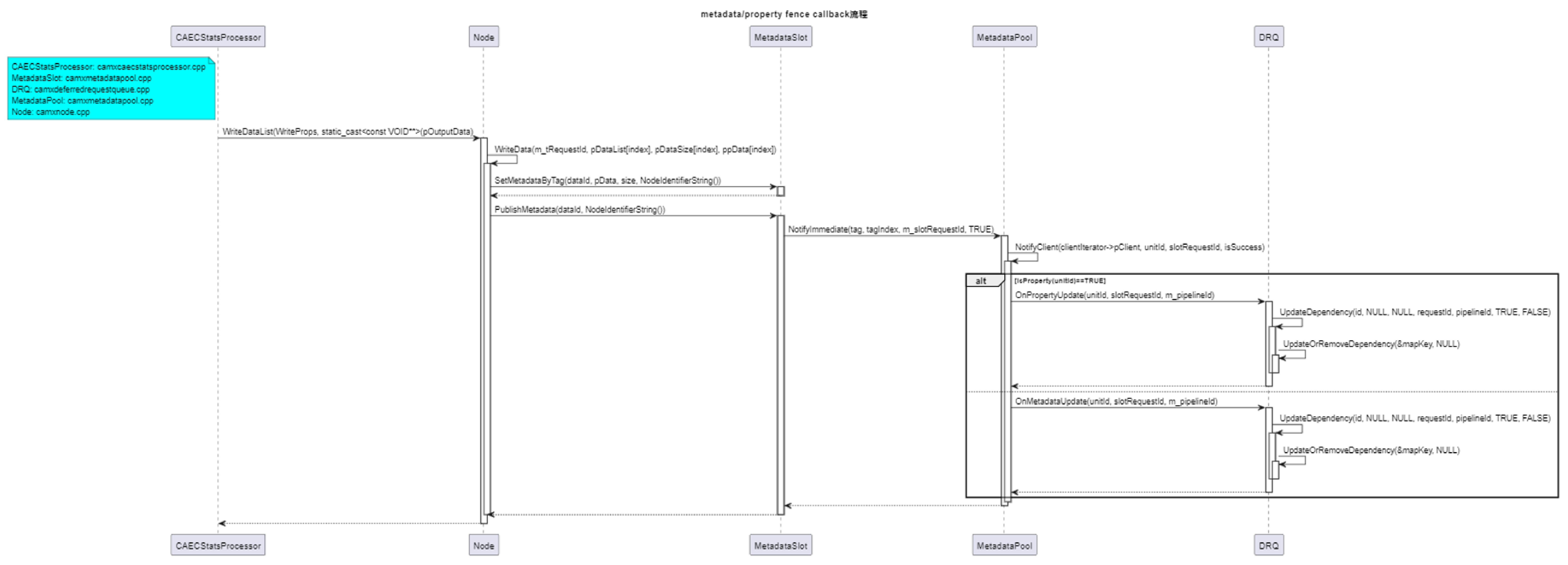

如何配置 csl fence async callback
pipeline::Process Request()
SetupRequest()
SetupRequestOutputPorts()
ProcessNonSink PortNewRequest()
Process SinkPortNewRequest()
SetupRequestOutputPortFence()
①
CSLCreatePrivateFence()
②
CSLFenceAsyncWait(hNewFence, Node::CSLFenceCallback, ...)

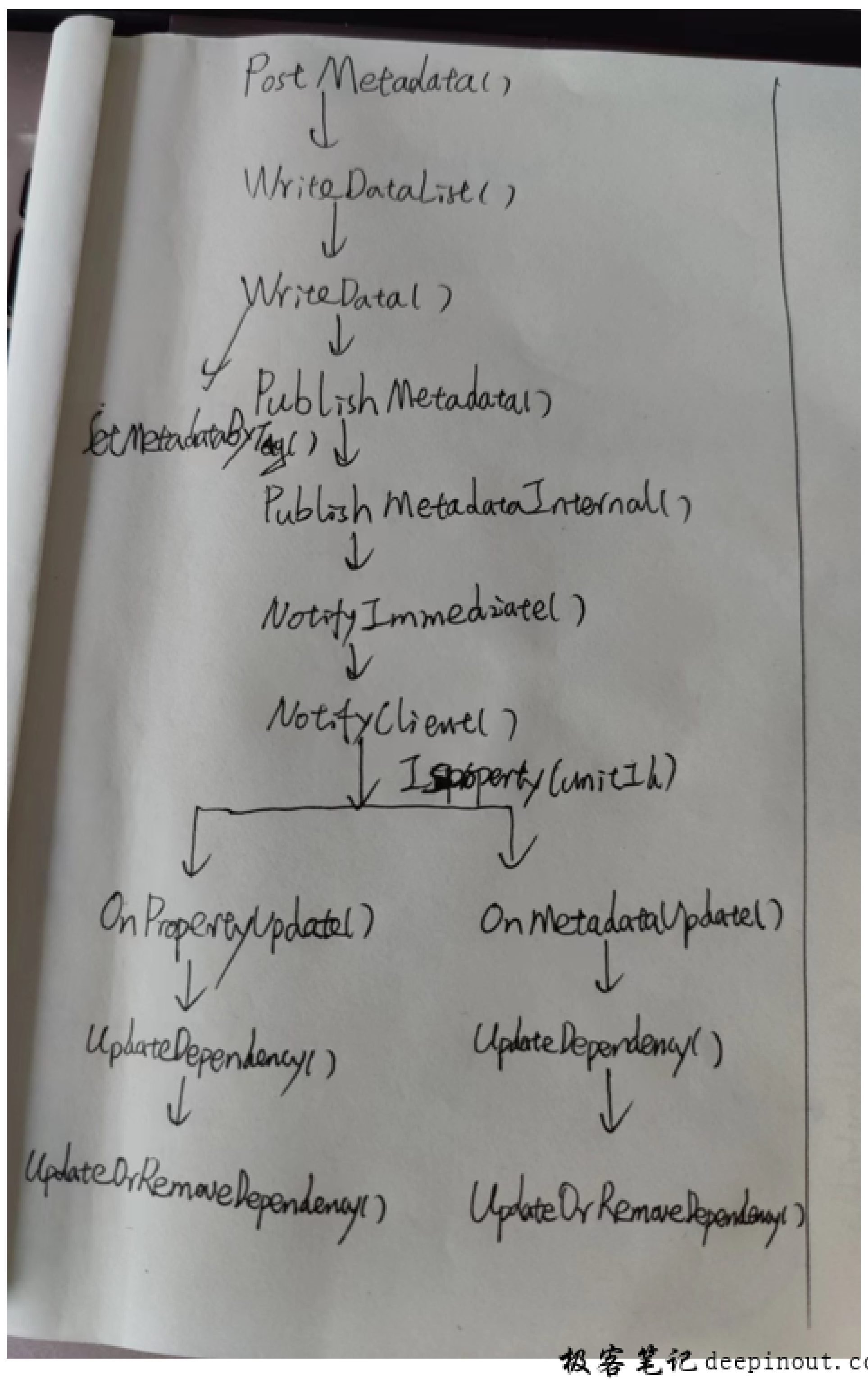



CSL Fence call back
Node::CSLFenceCallback()
递给线程池
Node::NodeThreadJobFamilyCb()
Node::ProcessFenceCallback()
if(0==m_perRequestInfo[requestIdIndex].numUnprocessedFences)
{ ProcessPartialMetadataDone()
ProcessMetadataDone()
ProcessRequestIdDone()
}
ProcessingNodeFenceCallback()
m_pPipeline->NonSinkPortFenceSignaled()
m_pDeferredRequestQueue-> FenceSignaledCallback()
UpdateDependency()
UpdateOrRemoveDependency()
m_pPipeline->SinkPortFenceSignaled()
m_pSession->NotifyResult()

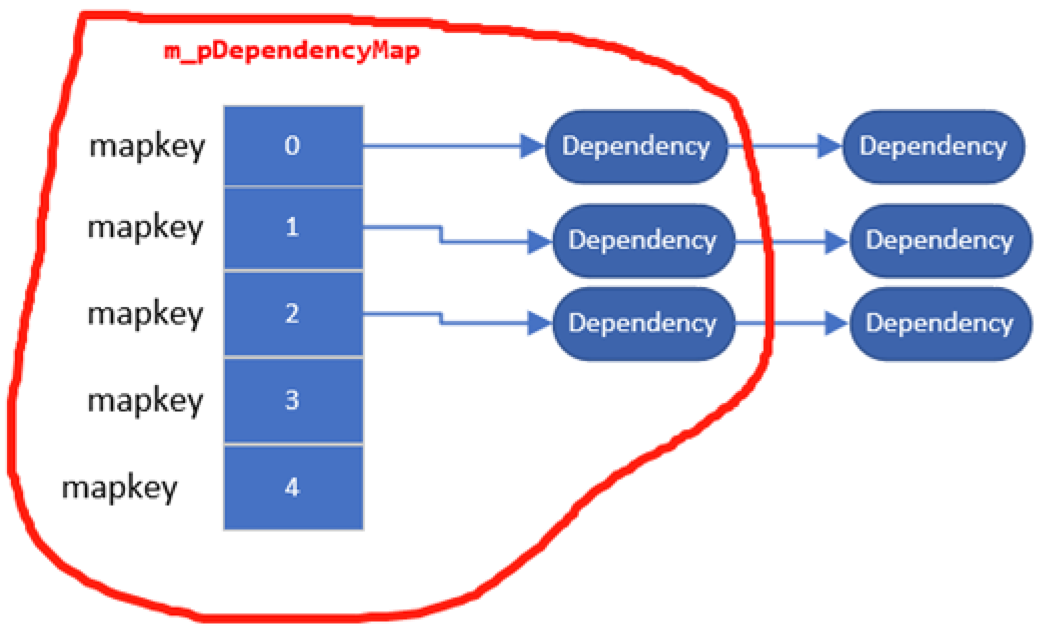
Metadata/property update 详解

Property/metadata callback flow

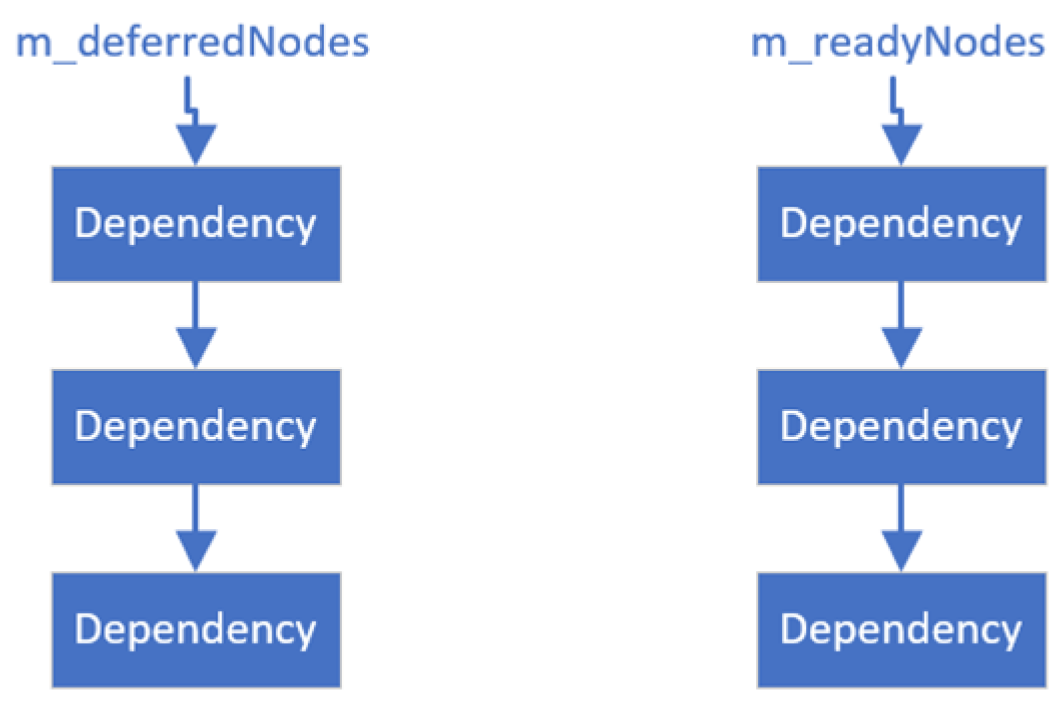

metadata/property fence callback流程
CAECStatsProcessor
Node
MetadataSlot
MetadataPool
DRQ
CAECStatsProcessor: camxcaecstatsprocessor.cpp
MetadataSlot: camxmetadatapool.cpp
DRQ: camxdeferredrequestqueue.cpp
MetadataPool: camxmetadatapool.cpp
Node: camxnode.cpp
WriteDataList(WriteProps, static_cast<const VOID**>(pOutputData)
WriteData(m_tRequestId, pDataList[index], pDataSize[index], ppData[index])
SetMetadataByTag(dataId, pData, size, NodeIdentifierString())
PublishMetadata(dataId, NodeIdentifierString())
NotifyImmediate(tag, tagIndex, m_slotRequestId, TRUE)
NotifyClient(clientIterator->pClient, unitId, slotRequestId, isSuccess)
alt
[IsProperty(unitId)==TRUE]
OnPropertyUpdate(unitId, slotRequestId, m_pipelineId)
UpdateDependency(id, NULL, NULL, requestId, pipelineId, TRUE, FALSE)
UpdateOrRemoveDependency(&mapKey, NULL)
OnMetadataUpdate(unitId, slotRequestId, m_pipelineId)
UpdateDependency(id, NULL, NULL, requestId, pipelineId, TRUE, FALSE)
UpdateOrRemoveDependency(&mapKey, NULL)
CAECStatsProcessor
Node
MetadataSlot
MetadataPool
DRQ



Dependency HashMap结构



m_deferredNodes与m_readyNodes结构


m_deferredNodes
m_readyNodes
Dependency
Dependency
Dependency
Dependency
Dependency
Dependency