Slice buffer design

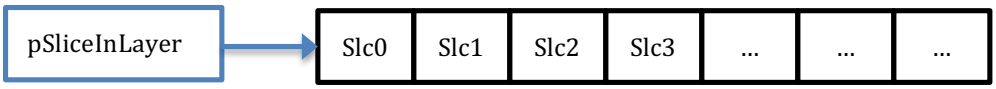
Table of Contents

1. Overview for new design

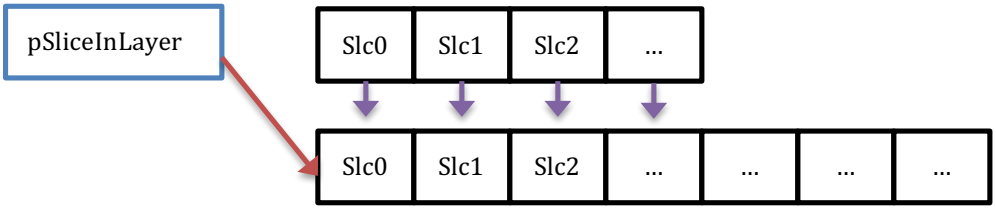
1.1 Origin design

SSlice* pSliceInLayer // slice buffer for all slices in layer

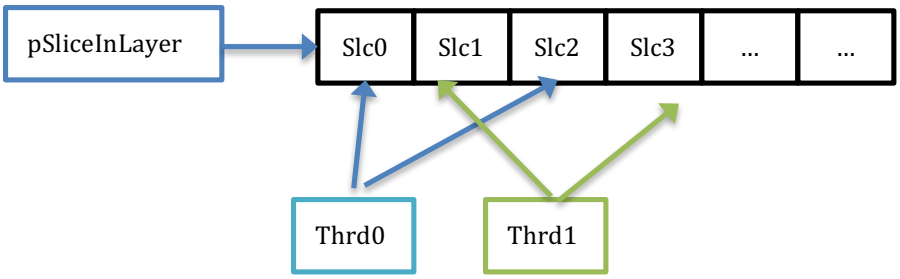
1.1.1. Single thread



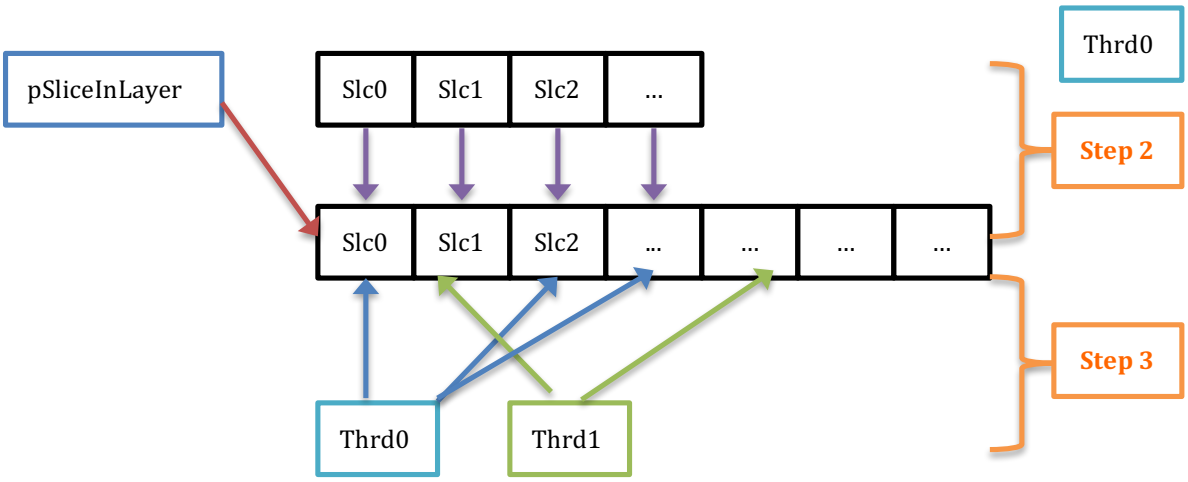
reallocate when current slice index larger than max slice num



1.1.2. Multi thread



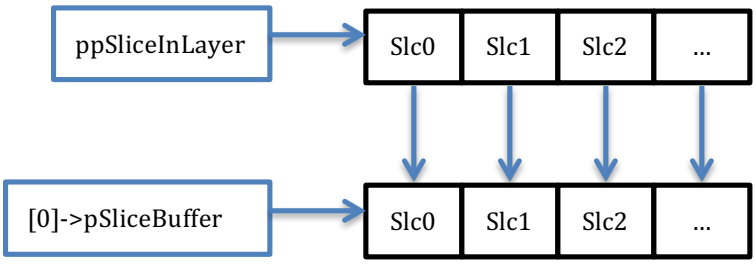
reallocate when current slice index larger than max slice num ,
--step 0: thread[0] detect that current slice index larger than max slice num;
--step 1: thread[0] need to wait thread[1] completed current slice encoding task
--step 2: thread[1] stop slice encoding and thread[0] reallocate slice buffer,
--step 3: thread[0]/thread[1] start to encode new slice



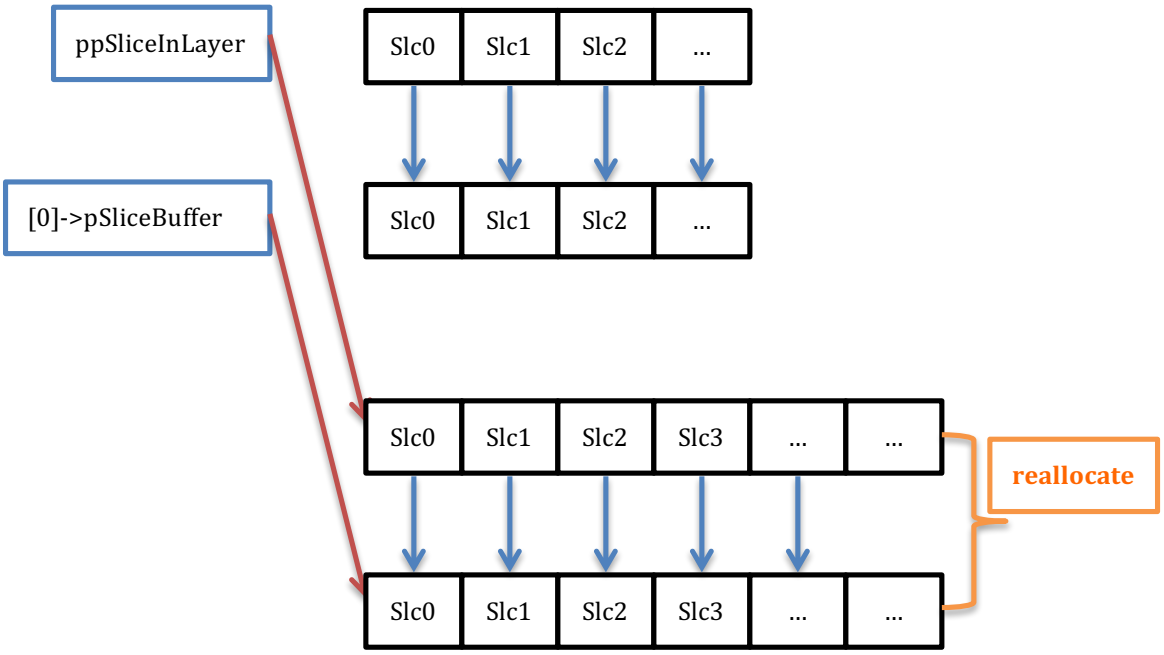
1.2 New design in review

```
SSlice** ppSliceInLayer;          // point to actual slice buffer

typedef struct TagSliceBufferInfo {
       SSlice*              pSliceBuffer;  // slice buffer for multi thread,
       int32_t              iMaxSliceNum;
       int32_t              iCodedSliceNum;
}SSliceBufferInfo;
```

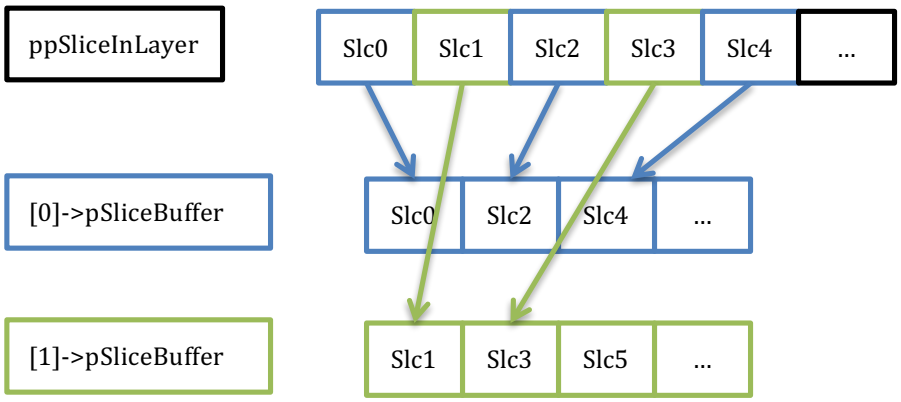
1.2.1 Single thread



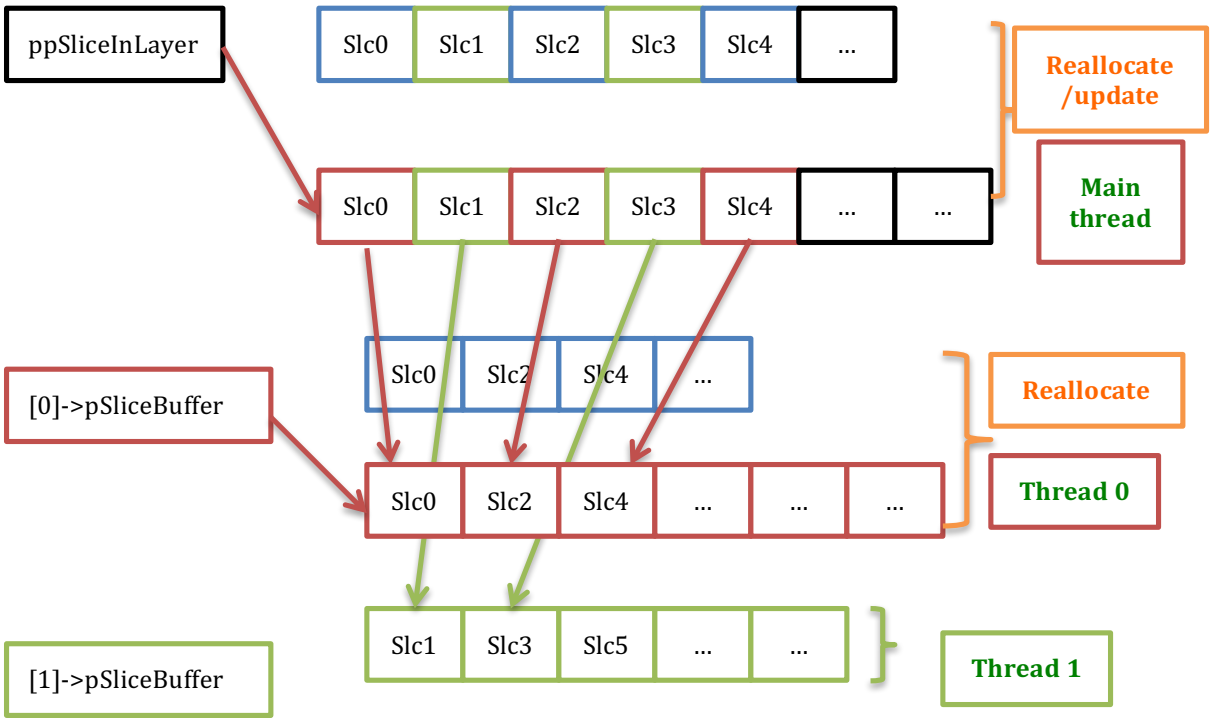
reallocate when current slice index larger than max slice num



1.2.2. Multi-thread



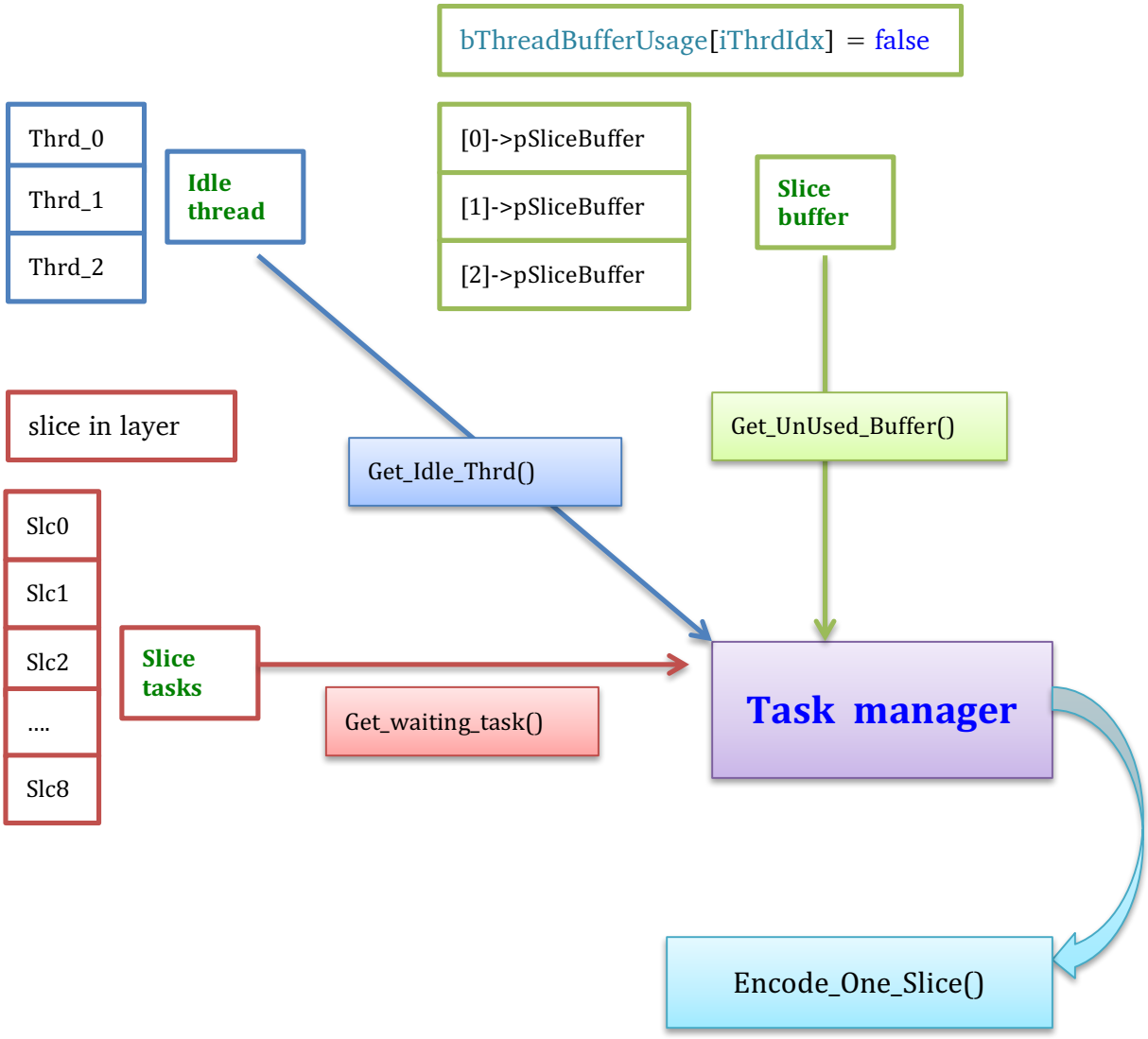
for reallocate, each thread will do it independently, and will update ppSliceInLayer by ***main thread*** when all slices in layer are encoded.



2. SLICE BUFFER AND THREAD

2.1 Before encoding one layer

the status of slice buffer and thread :
example:
thread: 3 threads
slices: 9 slices in layer



Query thread buffer function:

```
int32_t CWelsSliceEncodingTask::QueryEmptyThread (bool* pThreadBsBufferUsage) {
  for (int32_t k = 0; k < MAX_THREADS_NUM; k++) {
    if (pThreadBsBufferUsage[k] == false) {
      pThreadBsBufferUsage[k] = true;
      return k;
    }
  }
  return -1;
}
```

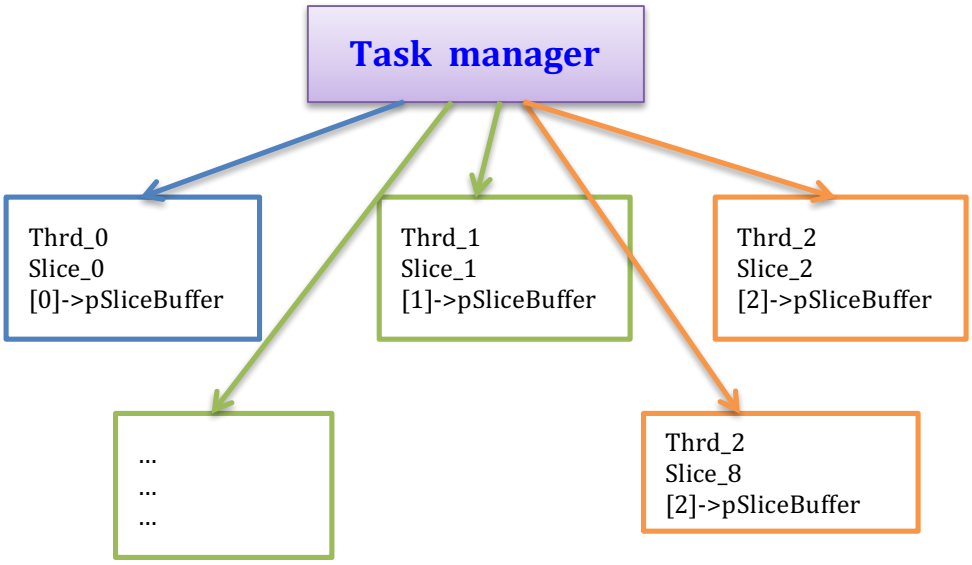
Example:

k=1, means that encoding slice task using [0]->pSliceBuffer as slice buffer

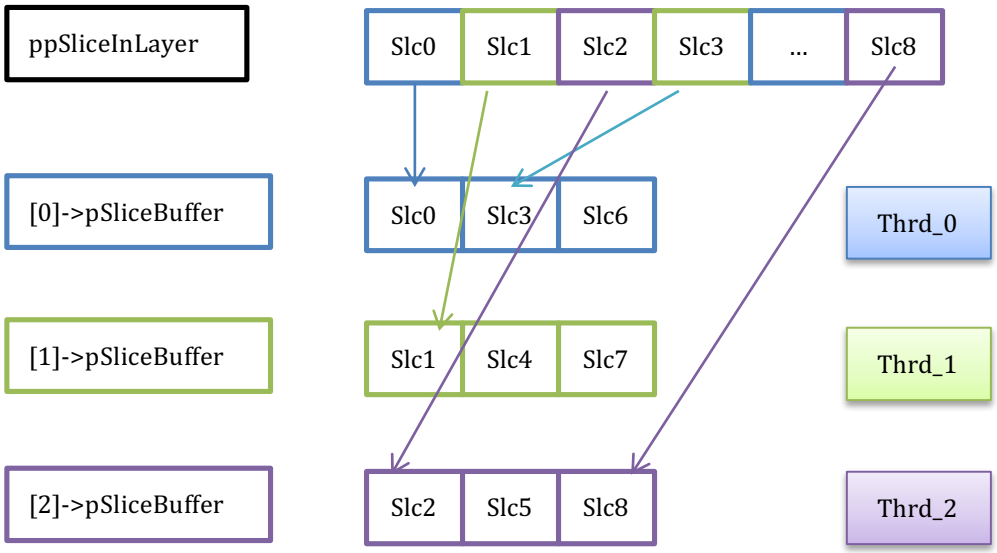
Query IDLE thread for slice task function:

```
CWelsTaskThread*   CWelsThreadPool::GetIdleThread() {
  CWelsAutoLock cLock (m_cLockIdleTasks);

  if (m_cIdleThreads->size() == 0) {
    return NULL;
  }

  CWelsTaskThread* pThread = m_cIdleThreads->begin();
  m_cIdleThreads->pop_front();
  return pThread;
}
```

2.2. Normal case for thread index and slice buffer index



Final map for thread index and slice buffer index, slice index

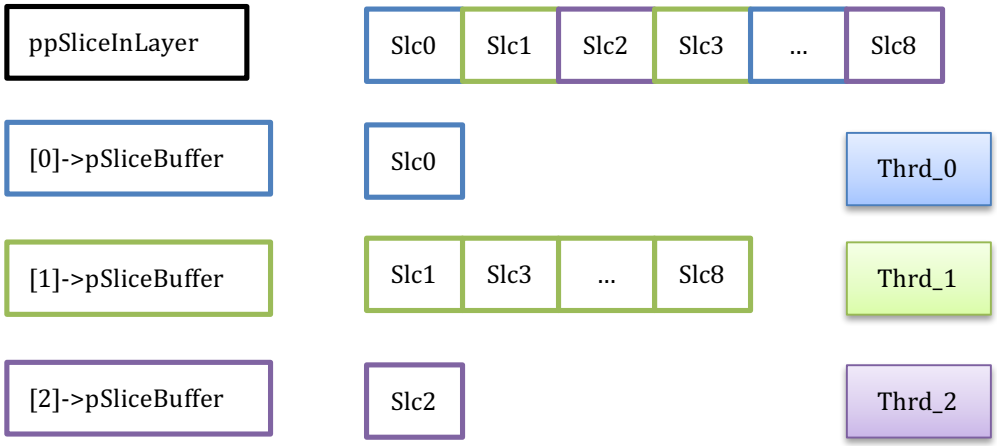


2.3. Encoded slice num is not the same among threads

case 1:

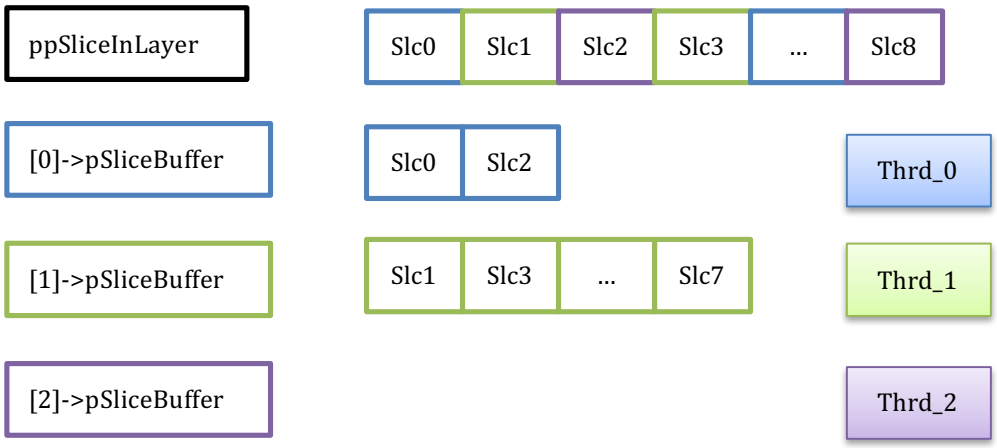
case 2:

encode time for Slc0 and Slc_2 are longer than Slc1, Slc3~Slc8



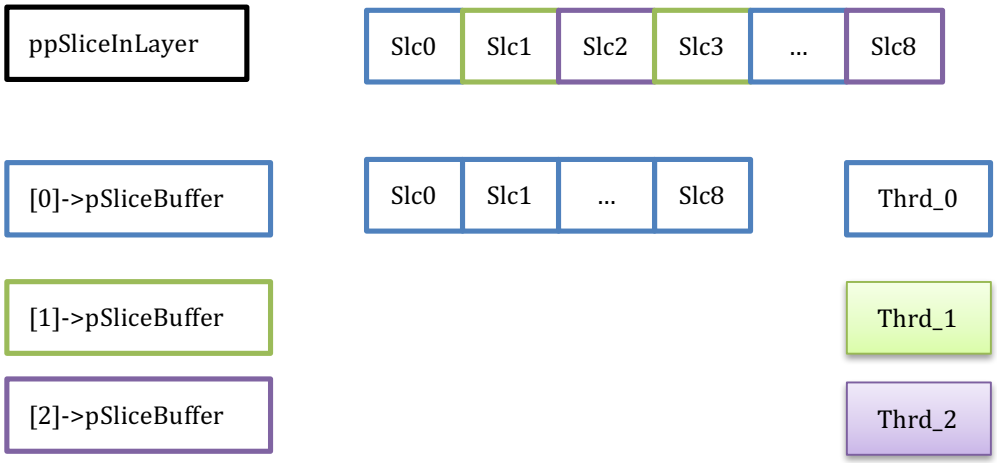
Corner case 1, no encoded slice for one thread:

Thread_0 and Thread_1 are fast enough and always get CPU resource,
Thread_2 is under waiting status/IDLE status.



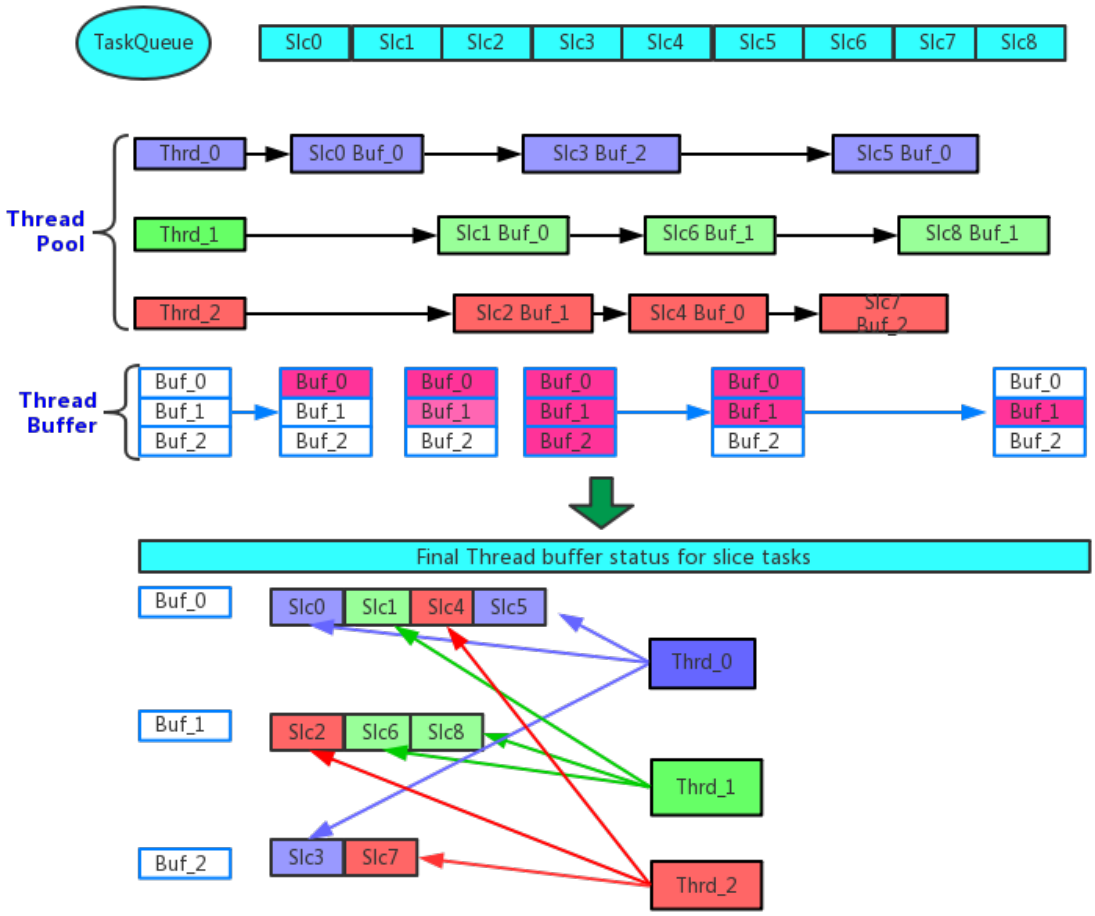
Corner case 2, all slices encoded by one thread :

Thread_0 is fast enough and always get CPU resource,
Thread_1 and Thread_2 are under waiting status/IDLE status.

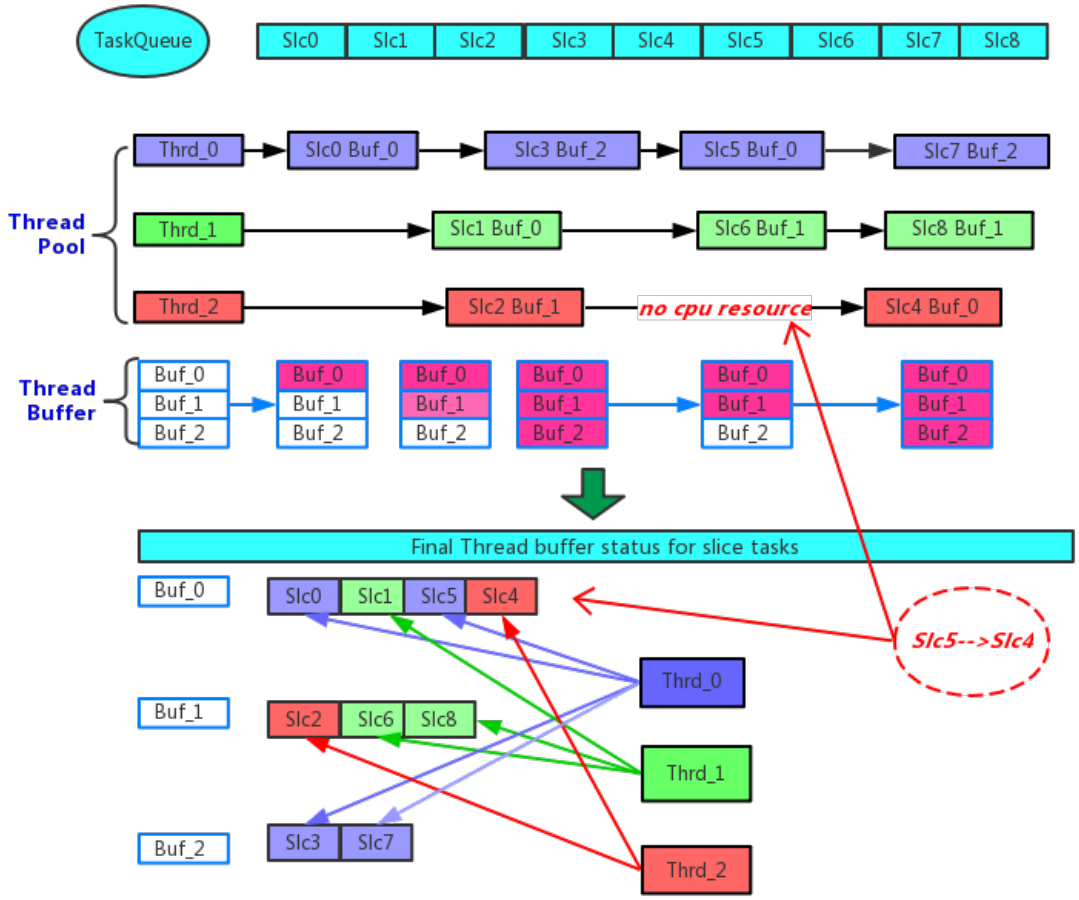


2.4. Different thread index in the same slice buffer

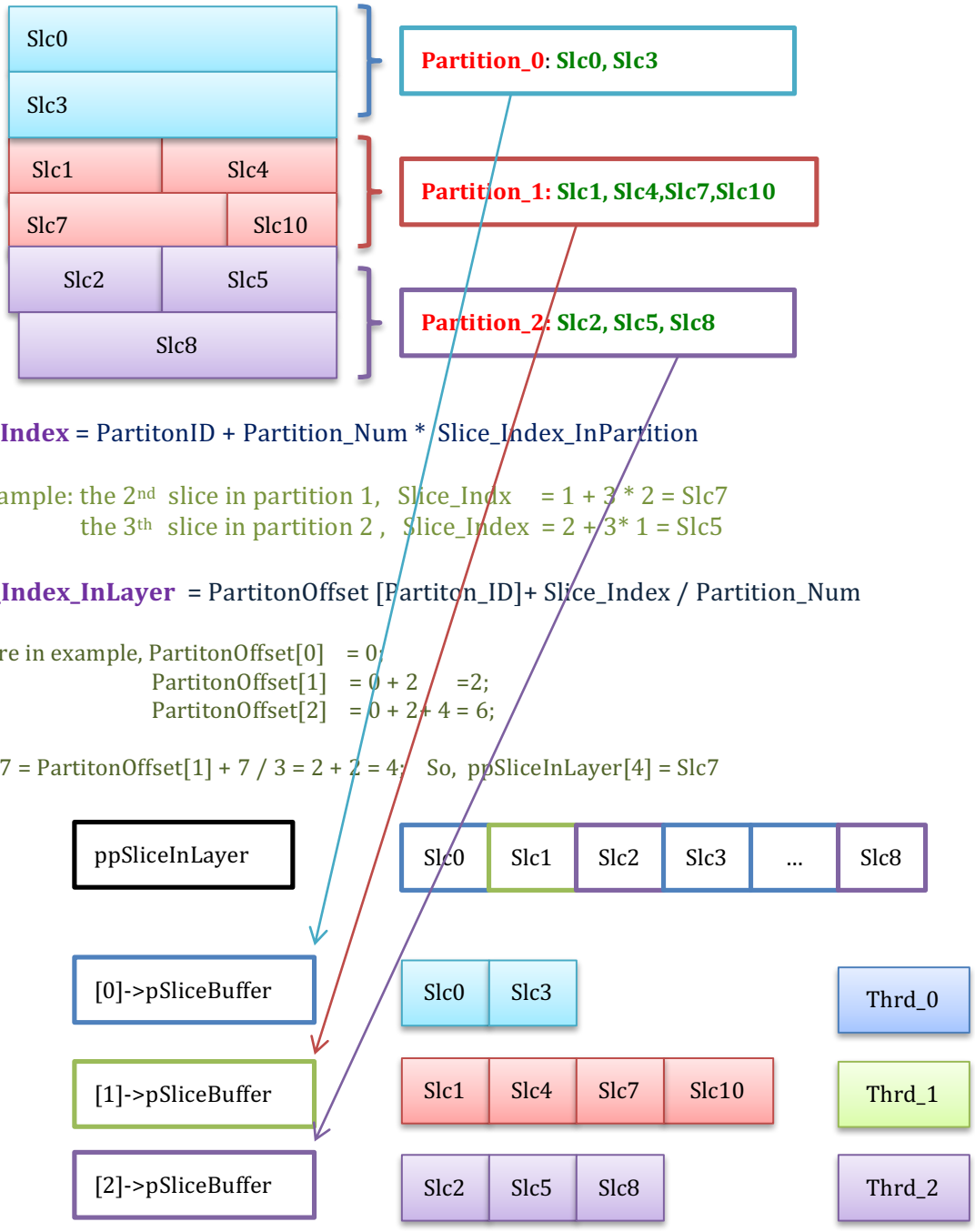
Timing diagram for Thread—SlcTask—ThreadBuffer



2.5. Slice index out-of-order case

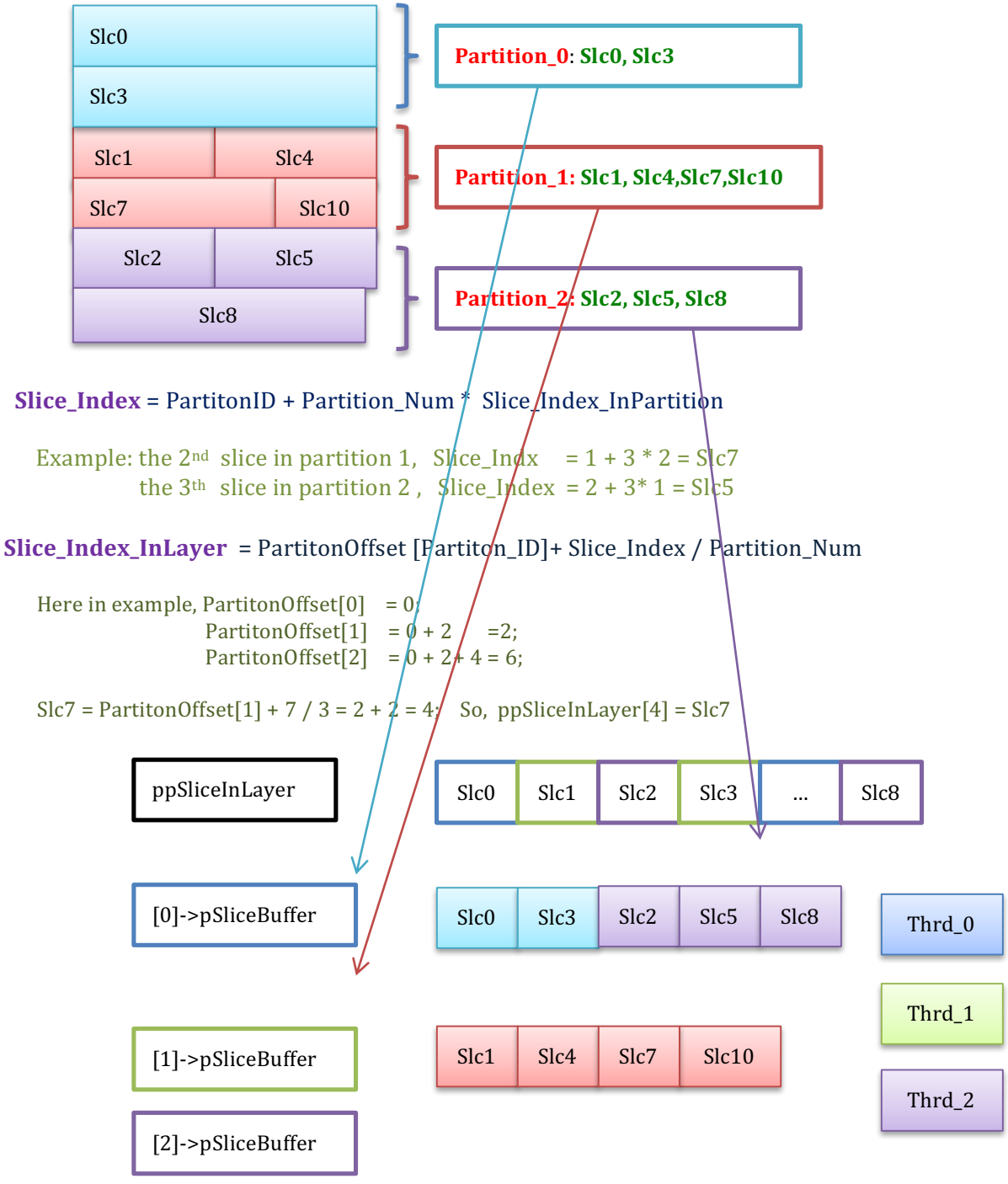


2.6. Dynamic slice mode case1



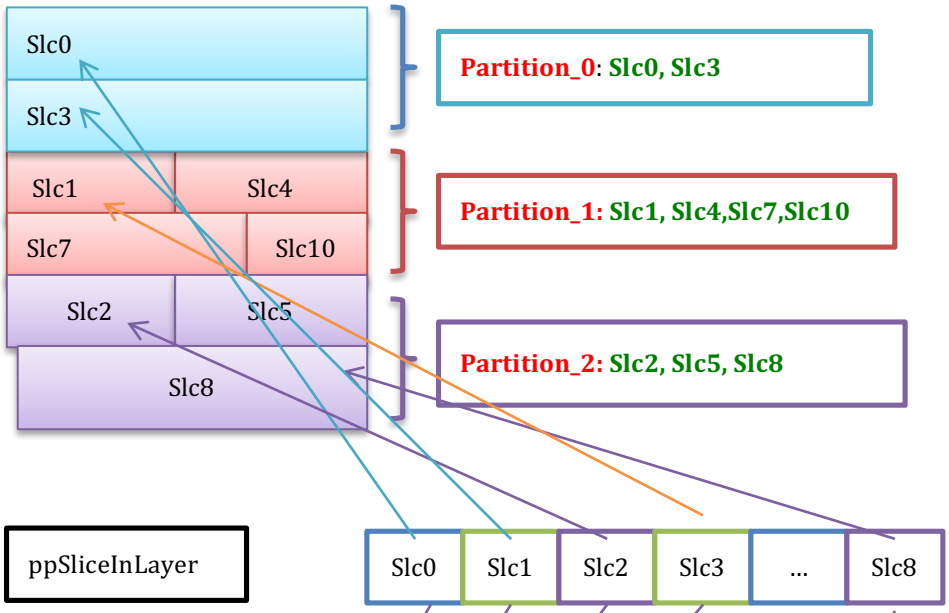
2.7. Dynamic slice mode case2

Thread_0 encoded two partitions while no partition for Thread_2

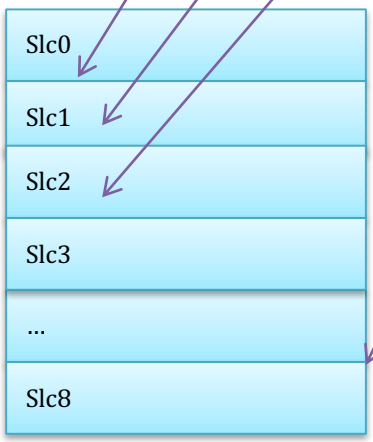


2.8 Slice index in different slice mode

Dynamic slice mode



Non-dynamic slice mode



3. Slice Buffer Init/Update/Reorder

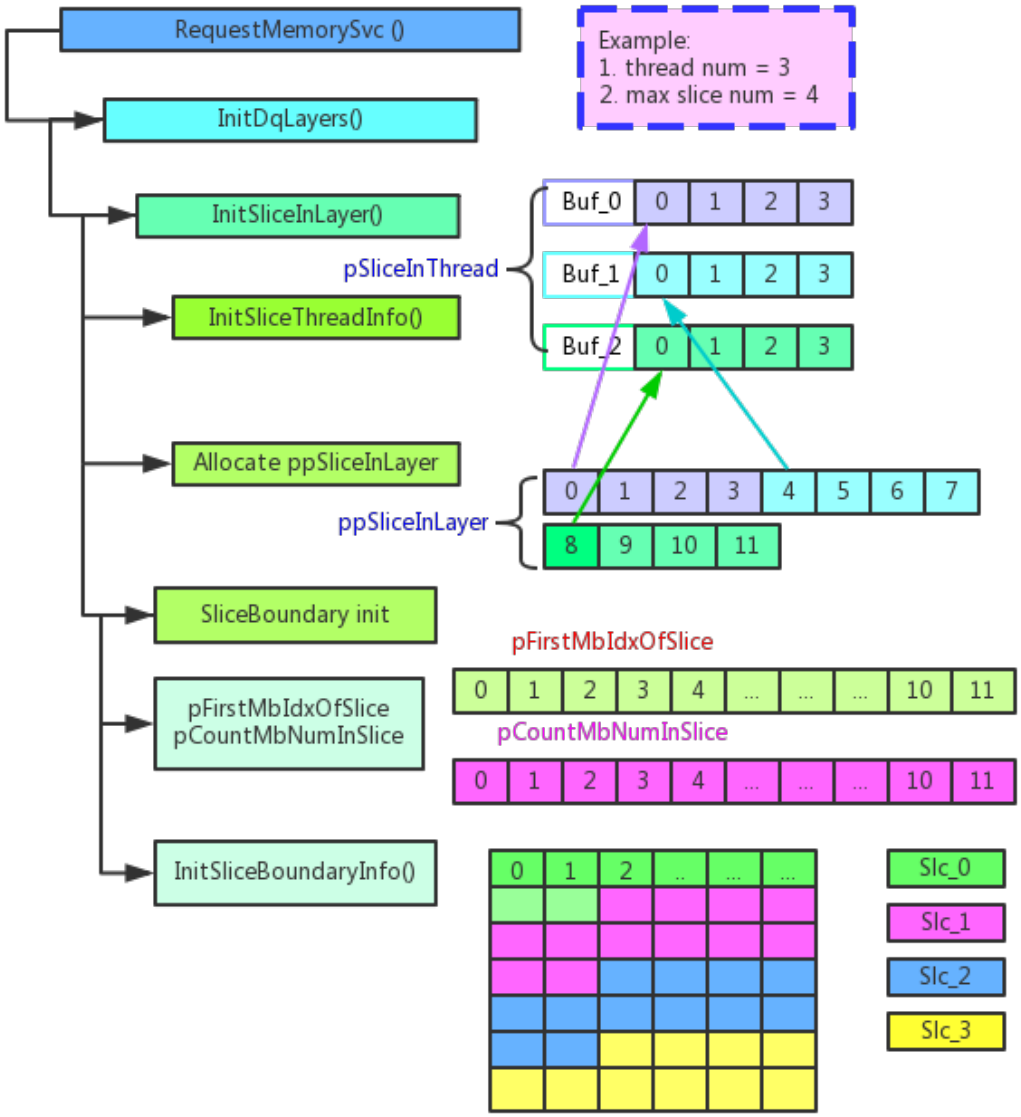
3.1. Allocate and initial before encoding first IDR

2. Allocate [0]->pSliceBuffer, [1] ->pSliceBuffer, etc.

here pSliceInThread has the same meaning with [0/1/2…]->pSliceBuffer

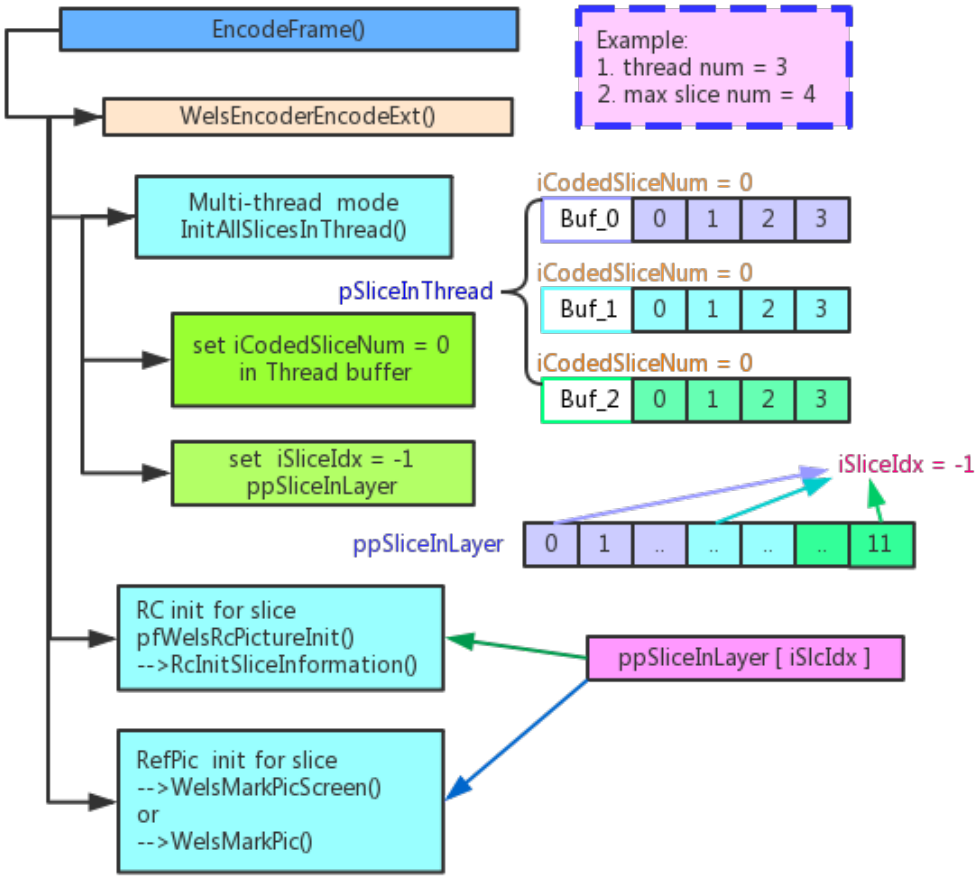
3. Allocate ppSliceInLayer buffer
4. Allocate pFirstMbIdxOfSlice and pCountMbNumInSlice buffer

Init slice boundary info



3.2 Init slice buffer info before encoder one layer

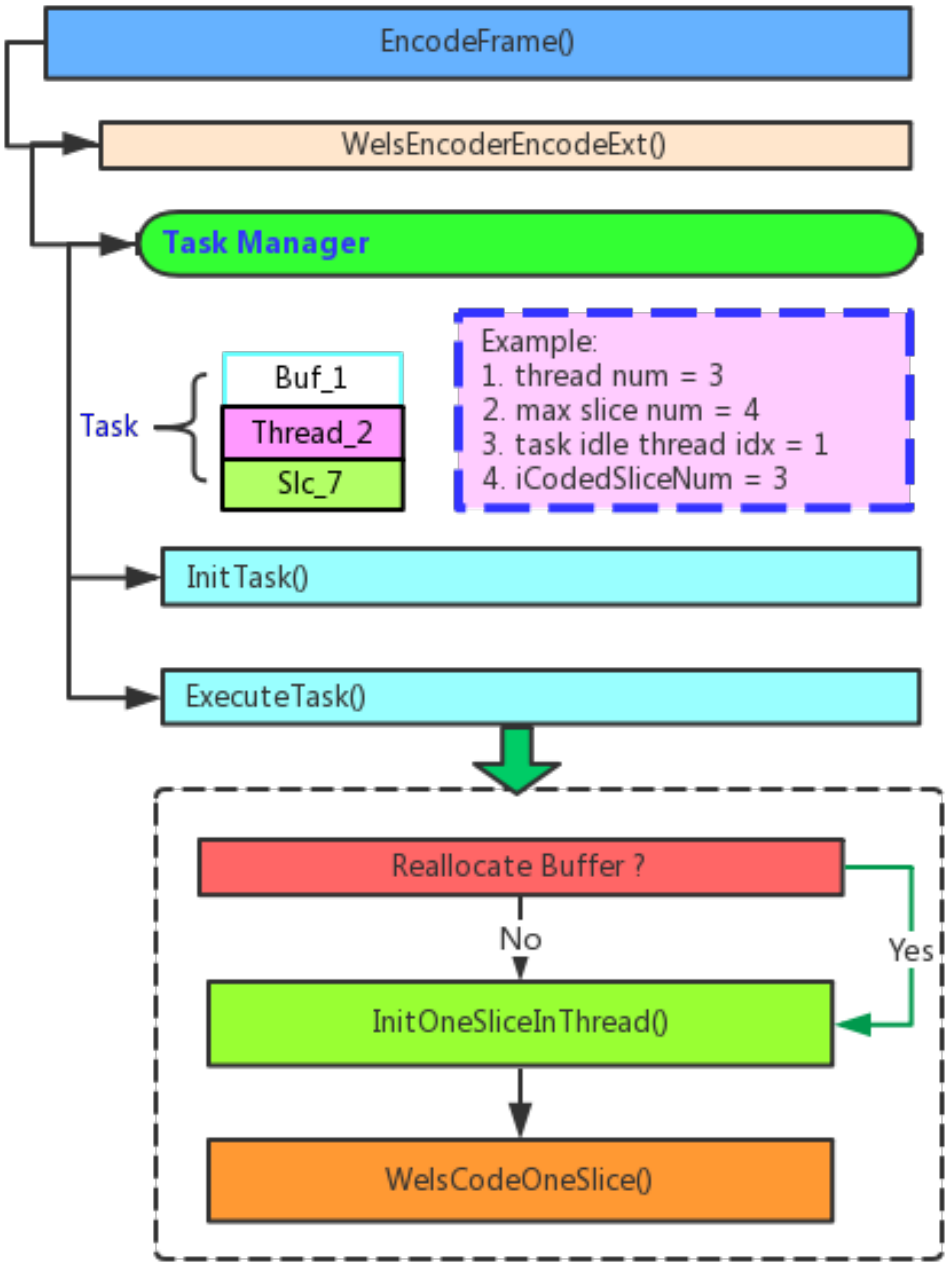
- Set iCodedSliceNum = 0 for all thread buffer
- Set iSliceIdx = -1 for all slice buffer (will set to actual slice idx during encoding one slice)
- Init RC info for all slice level parameters using ppSliceInLayer[iSlcIdx]
- Init reference pic info for all slices using ppSliceInLayer[iSlcIdx]



3.3 Slice buffer init/update/reallocate during encoding one slice

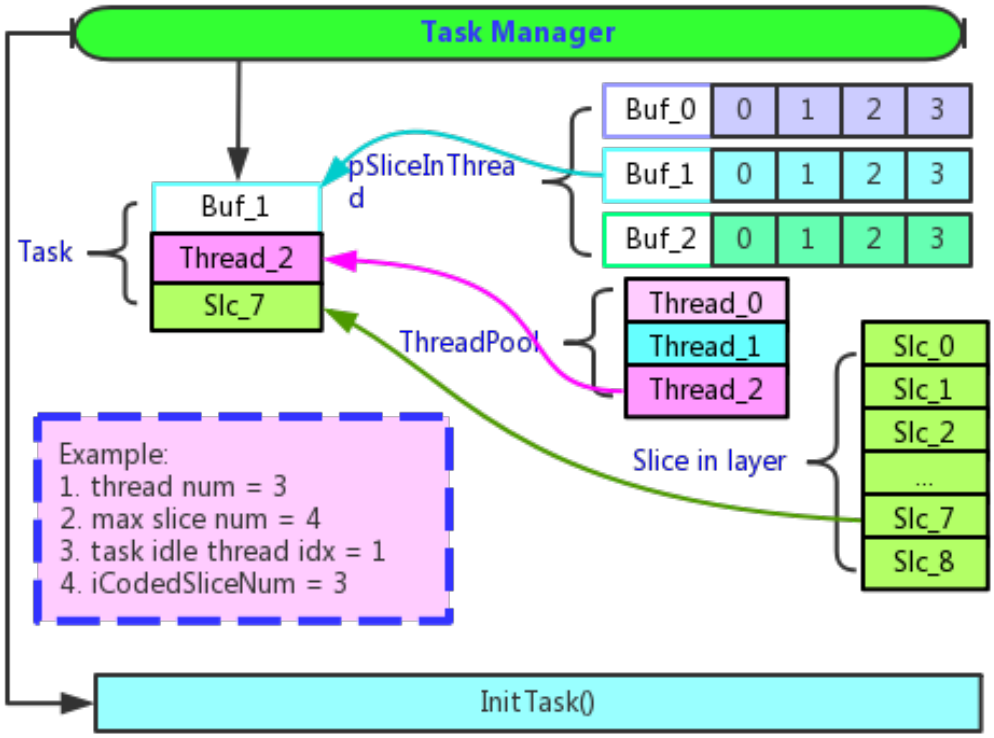
3.3.1 basic flow chart for encoding one slice task

- Task manager: Get idle thread, unused thread buffer, and encoding slice
- Init task: Init slice boundary info etc.
- Excute task: Encode one slice etc



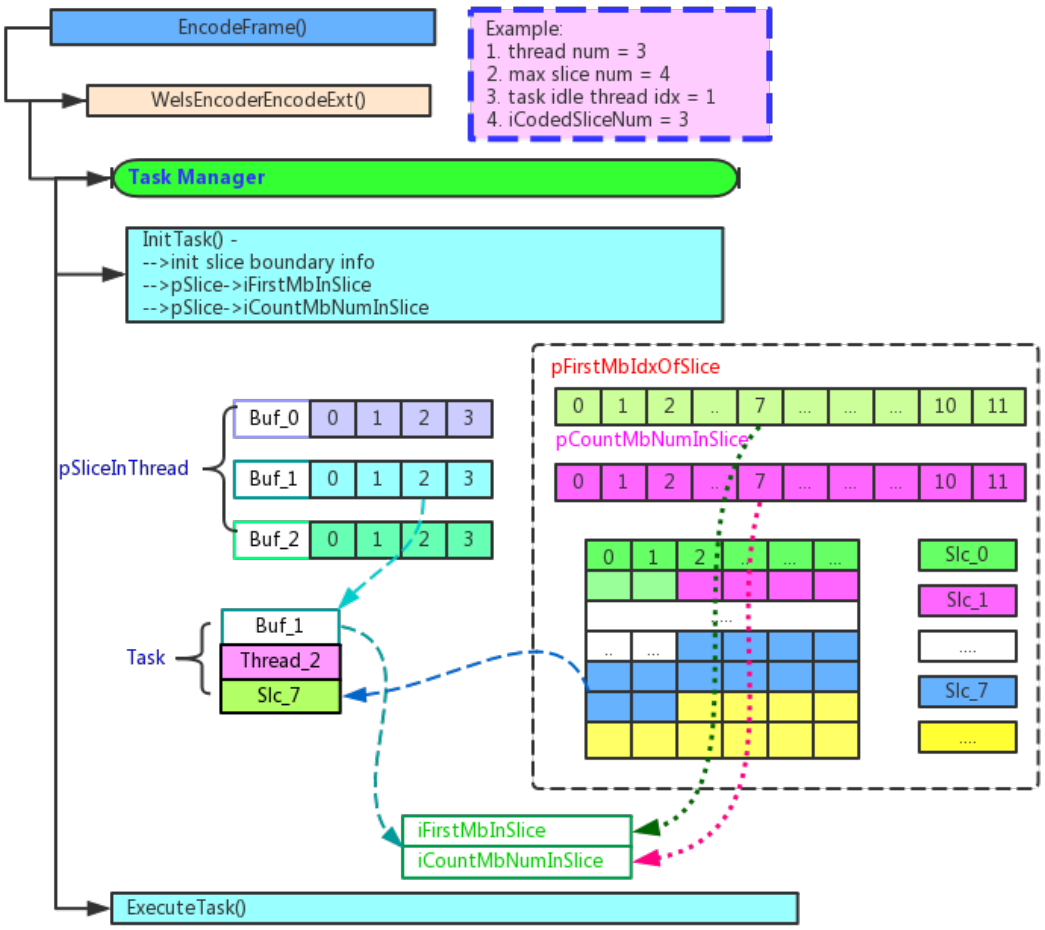
3.3.2 Task manager

- Get unused thread buffer, in this example, choose Buf_1 as slice buffer
- Get Idle thread, thread_2
- Get encode slice index in layer which will be encoded in this task, Slc_7



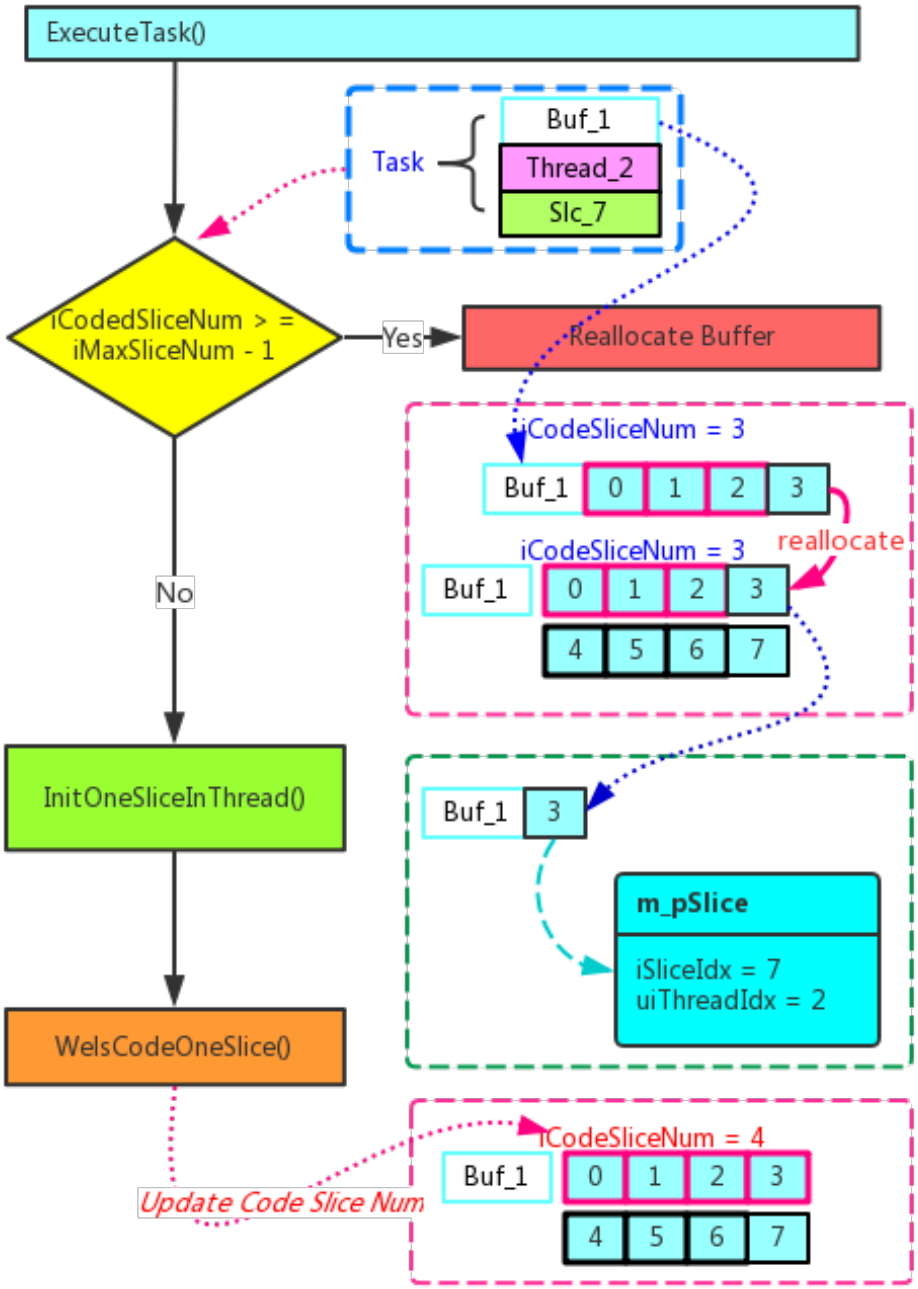
3.3.3. InitTask before encode one slice

- init rc with boundary info
  GomRCInitForOneSlice()
- update next slice boundary(start MB index etc.)



3.3.4 Slice buffer update/reallocate during execute task

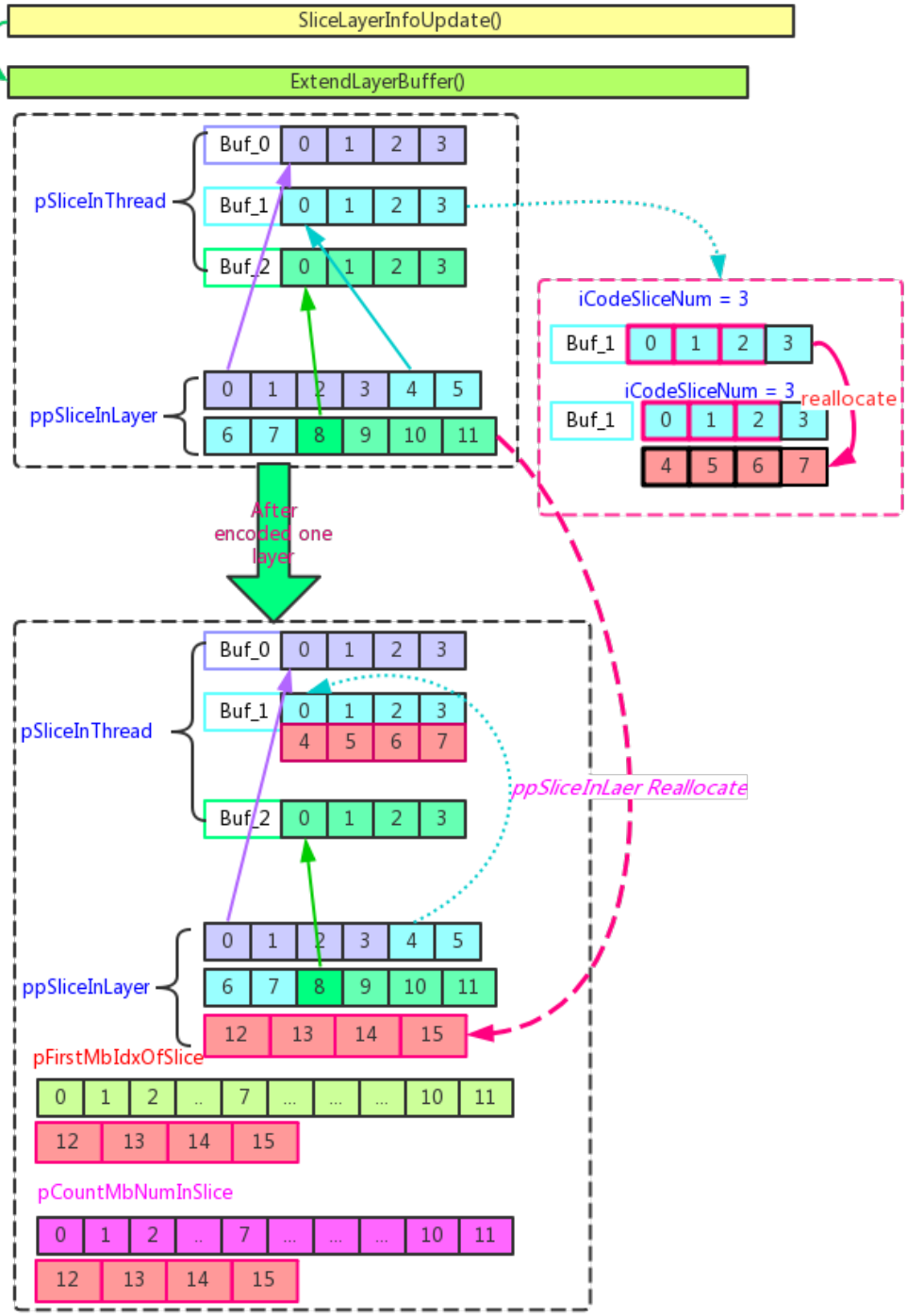
- check whether need to reallocate
- init m_pSlice buffer
- update thread buffer info, iCodeSliceNum ++



3.4. Slice buffer update after encoder one layer

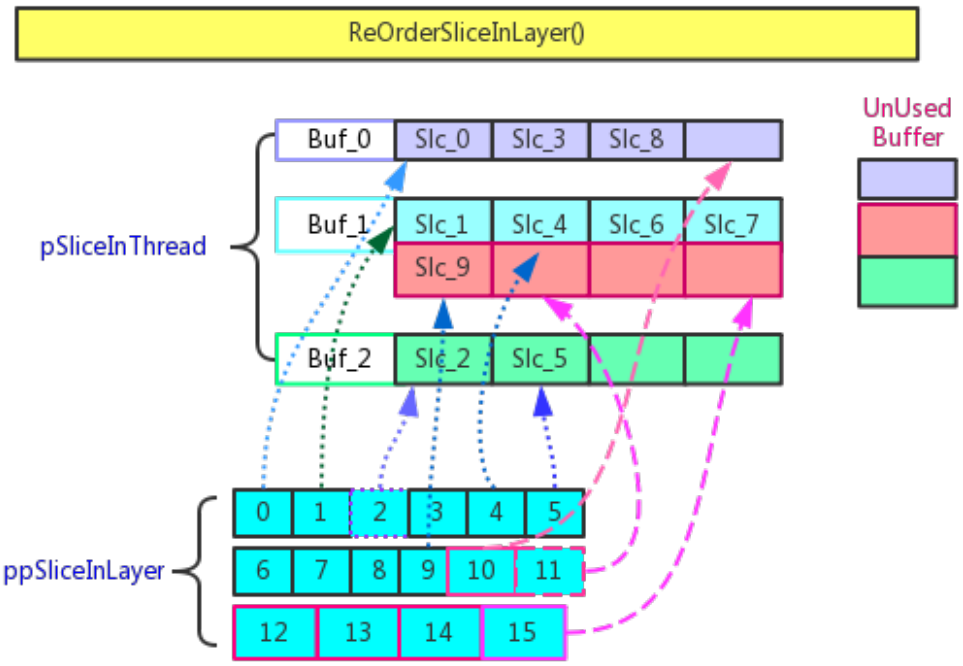
3.4.1. ppSliceInLayer update

extend and reallocate ppSliceInLayer buffer,
ppSliceInLayer buffer num == All thread slice buffer num



3.4.2. Slice buffer reorder

example 1

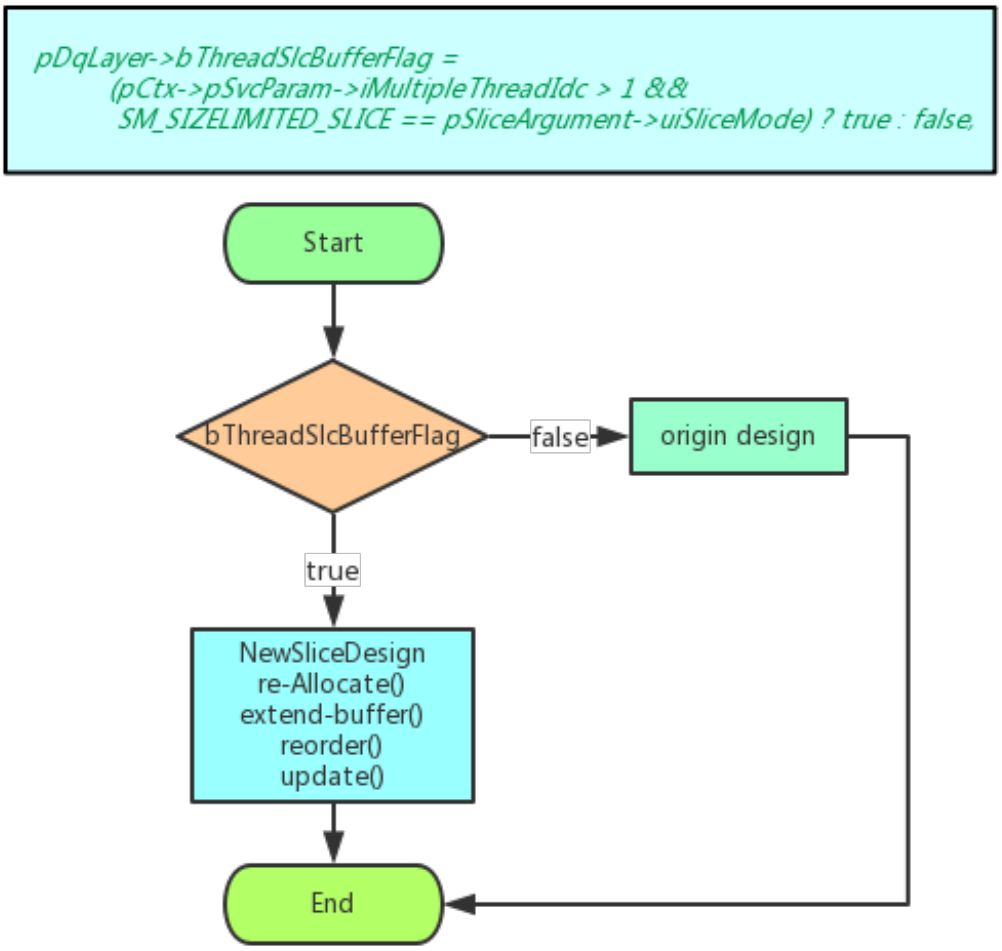


4. Final design

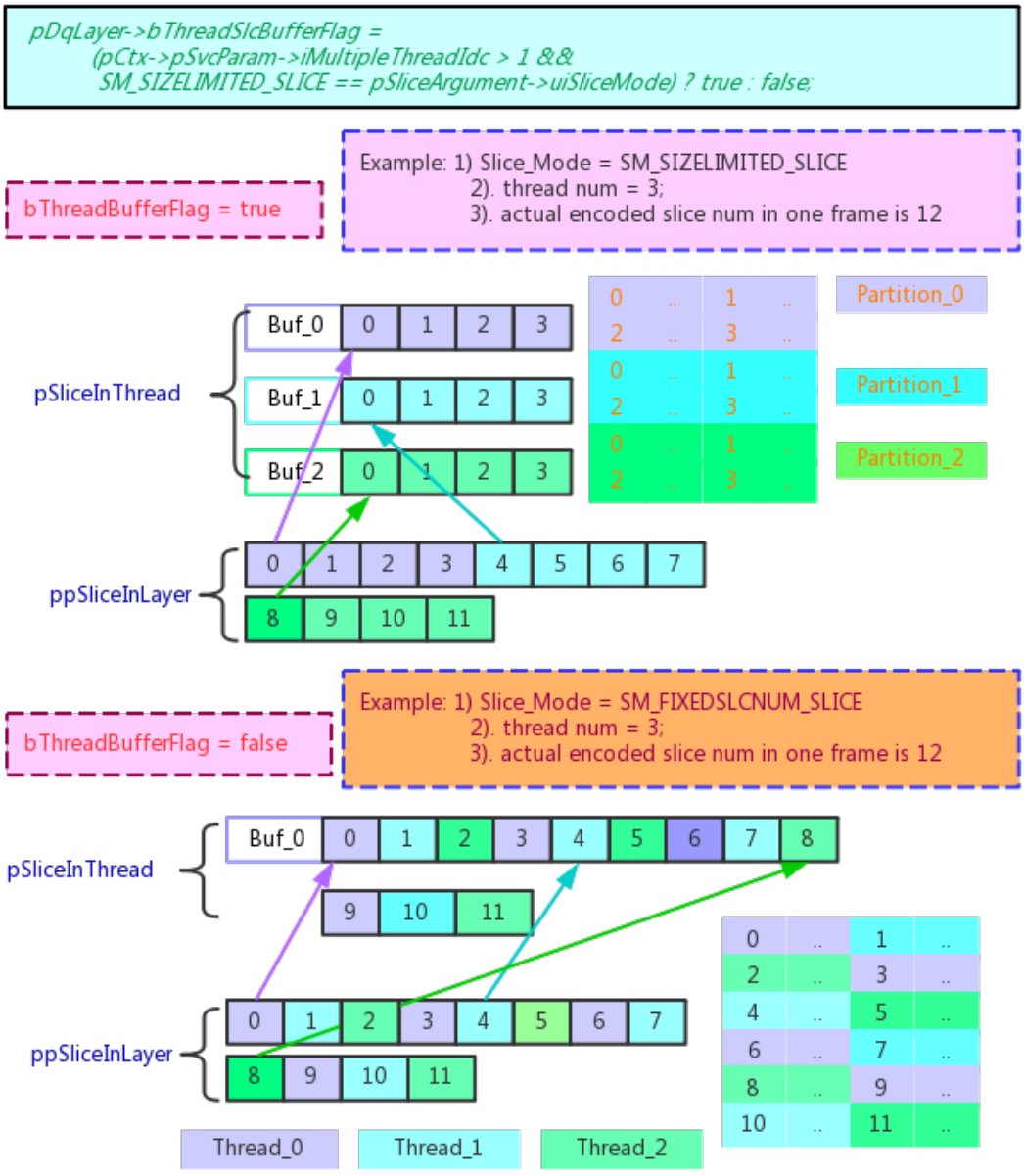
4.1 overall flow chart

```
pDqLayer->bThreadSlcBufferFlag =
        (pCtx->pSvcParam->iMultipleThreadIdc > 1 &&
        SM_SIZELIMITED_SLICE == pSliceArgument->uiSliceMode) ? true : false;
```

- bThreadSlcBufferFlag = fase, keep origin design
- bThreadSlcBufferFlag = true, using new buffer design



Example


pDqLayer->bThreadSlcBufferFlag =
(pCtx->pSvcParam->iMultipleThreadIdc > 1 &&
SM_SIZELIMITED_SLICE == pSliceArgument->uiSliceMode) ? true : false;
bThreadBufferFlag = true
Example: 1) Slice_Mode = SM_SIZELIMITED_SLICE
2). thread num = 3;
3). actual encoded slice num in one frame is 12
Buf_0
Buf_1
Buf_2
pSliceInThread
ppSliceInLayer
Partition_0
Partition_1
Partition_2
bThreadBufferFlag = false
Example: 1) Slice_Mode = SM_FIXEDSLCNUM_SLICE
2). thread num = 3;
3). actual encoded slice num in one frame is 12
Thread_0
Thread_1
Thread_2

4.2. functions for new design

4.2.1 Reallocate module

```
bNeedReallocate = (pCurDq->sSliceBufferInfo[m_iThreadIdx].iCodedSliceNum  >=
                   pCurDq->sSliceBufferInfo[m_iThreadIdx].iMaxSliceNum -1)
                   ? true : false;
if (bNeedReallocate) {
   WelsMutexLock (&m_pCtx->pSliceThreading->mutexThreadSlcBuffReallocate);
   //for memory statistic variable
   iReturn = ReallocateSliceInThread(m_pCtx, pCurDq, m_pCtx->uiDependencyId,
             m_iThreadIdx);
   WelsMutexUnlock (&m_pCtx->pSliceThreading->mutexThreadSlcBuffReallocate);
   if (ENC_RETURN_SUCCESS != iReturn) {
     return iReturn;
   }
 }
```

4.2.2 Extend layer buffer module

```
  //reallocate ppSliceInLayer if total encoded slice num exceed max slice num
  if (iMaxSliceNum > pCtx->pCurDqLayer->iMaxSliceNum) {
    iRet = ExtendLayerBuffer(pCtx, pCtx->pCurDqLayer->iMaxSliceNum, iMaxSliceNum);
    if (ENC_RETURN_SUCCESS != iRet) {
      return iRet;
    }
    pCtx->pCurDqLayer->iMaxSliceNum = iMaxSliceNum;
  }
```

4.2.3. Reorder slice buffer module

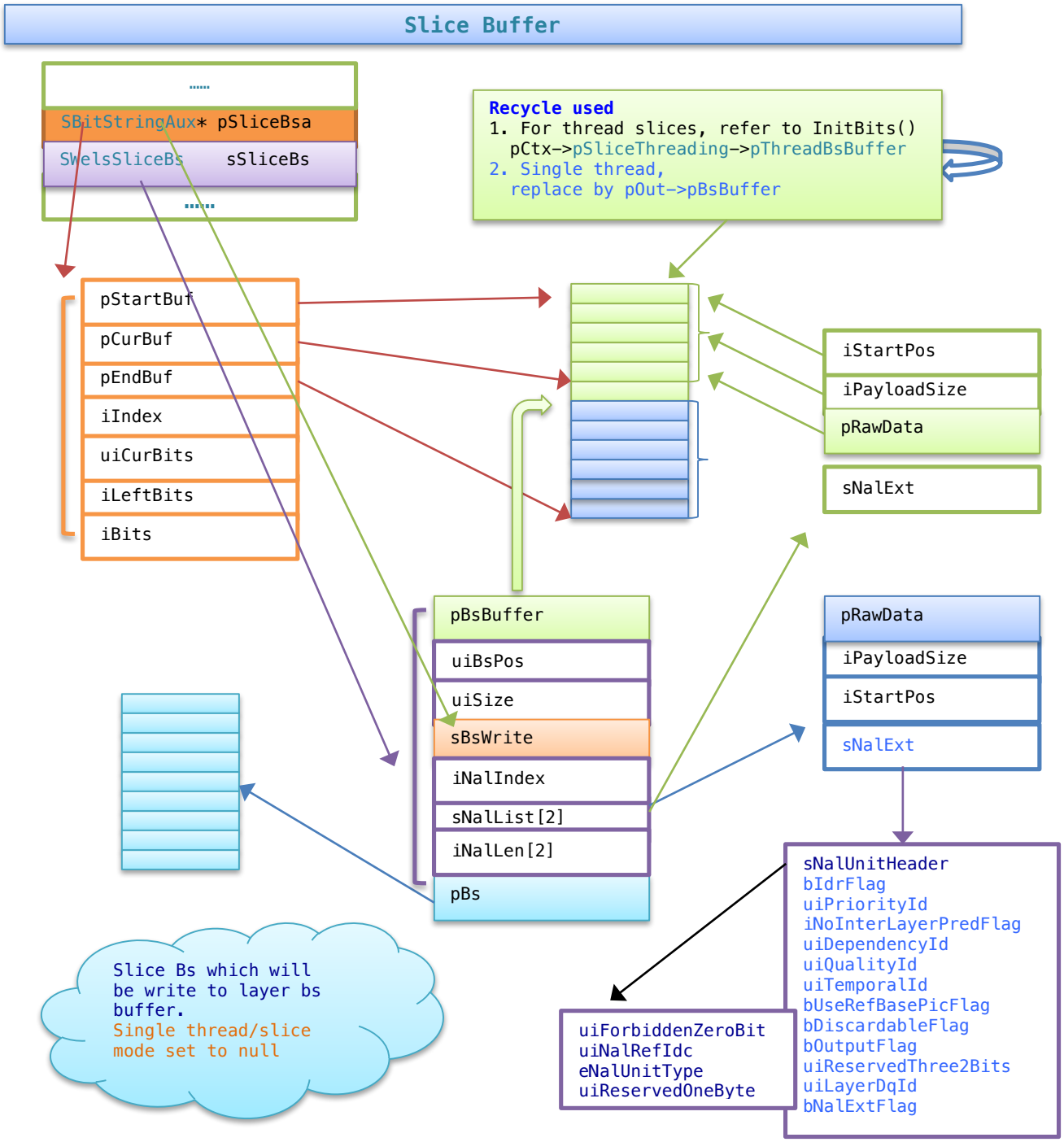
```
  //update ppSliceInLayer based on pSliceBuffer, reordering based on slice index
  iRet = ReOrderSliceInLayer (pCtx, kuiSliceMode, pCtx->iActiveThreadsNum);
  if (ENC_RETURN_SUCCESS != iRet) {
    WelsLog (& (pCtx->sLogCtx), WELS_LOG_ERROR,
             "CWelsH264SVCEncoder::SliceLayerInfoUpdate: ReOrderSliceInLayer
failed");
    return iRet;
  }
```

4.2.4 other functions for new design

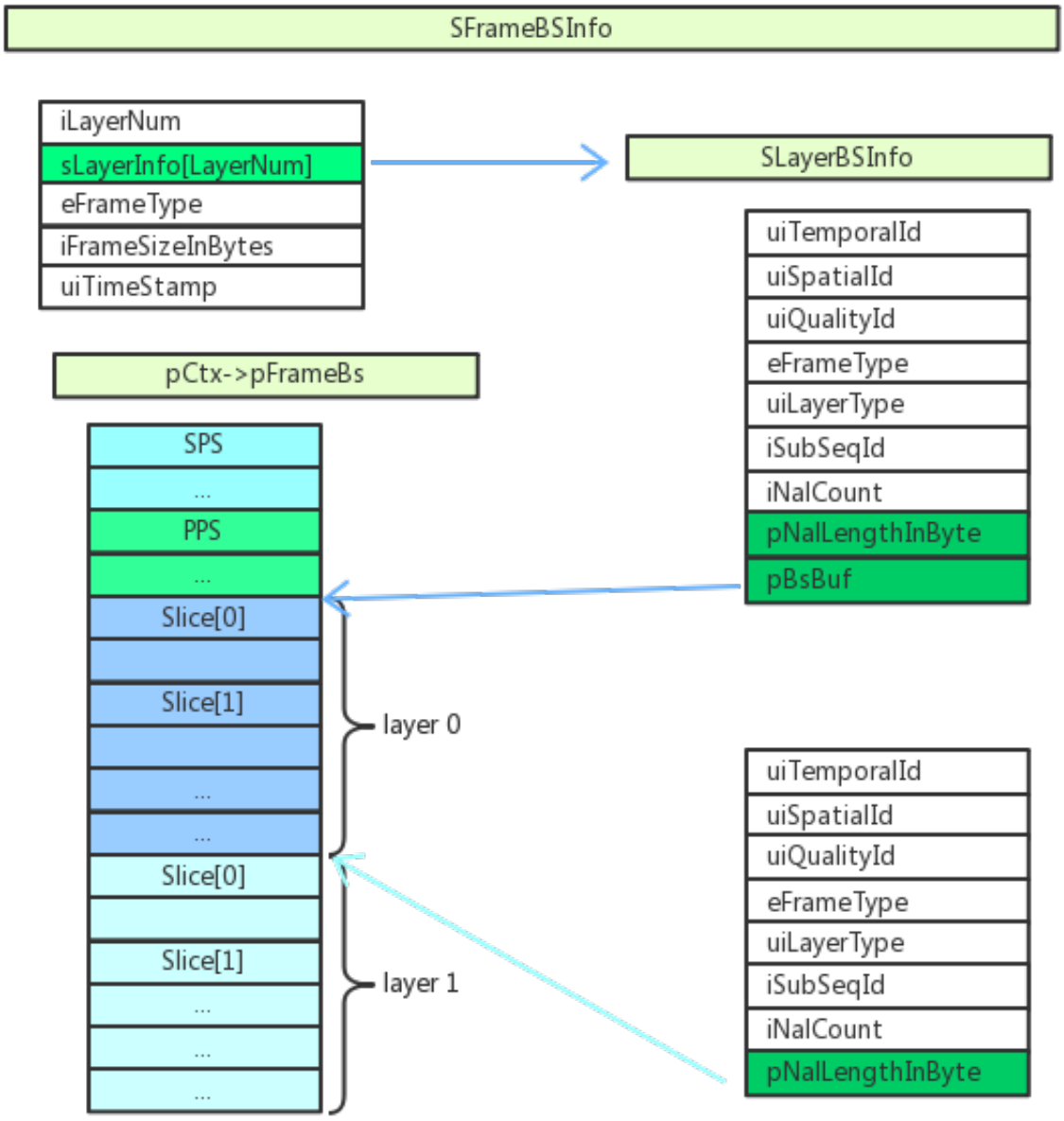
```
int32_t InitSliceList (SSlice*& pSliceList,
                       SBitStringAux* pBsWrite,
                       const int32_t kiMaxSliceNum,
                       const int32_t kiMaxSliceBufferSize,
                       const bool bIndependenceBsBuffer,
                       CMemoryAlign* pMa);

int32_t InitAllSlicesInThread (sWelsEncCtx* pCtx);

int32_t InitOneSliceInThread (sWelsEncCtx* pCtx,
                              SSlice*& pSlice,
                              const int32_t kiSlcBuffIdx,
                              const int32_t kiDlayerIdx,
                              const int32_t kiSliceIdx);

int32_t InitSliceInLayer (sWelsEncCtx* pCtx,
                          SDqLayer* pDqLayer,
                          const int32_t kiDlayerIndex,
                          CMemoryAlign* pMa);

int32_t ReallocateSliceList (sWelsEncCtx* pCtx,
                             SSliceArgument* pSliceArgument,
                             SSlice*& pSliceList,
                             const int32_t kiMaxSliceNumOld,
                             const int32_t kiMaxSliceNumNew);

int32_t ReallocateSliceInThread (sWelsEncCtx* pCtx,
                                 SDqLayer* pDqLayer,
                                 const int32_t kiDlayerIdx,
                                 const int32_t KiSlcBuffIdx);

int32_t ReallocSliceBuffer (sWelsEncCtx* pCtx);

int32_t GetCurLayerNalCount(const SDqLayer* pCurDq, const int32_t kiCodedSliceNum);
int32_t GetTotalCodedNalCount(SFrameBSInfo* pFbi);

int32_t FrameBsRealloc (sWelsEncCtx* pCtx,
                        SFrameBSInfo* pFrameBsInfo,
                        SLayerBSInfo* pLayerBsInfo,
                        const int32_t kiMaxSliceNumOld);

int32_t ReOrderSliceInLayer(sWelsEncCtx* pCtx,
                            const SliceModeEnum kuiSliceMode,
                            const int32_t kiThreadNum);

int32_t SliceLayerInfoUpdate (sWelsEncCtx* pCtx,
                              SFrameBSInfo* pFrameBsInfo,
                              SLayerBSInfo* pLayerBsInfo,
                              const SliceModeEnum kuiSliceMode);
```

5. Buffer Structure

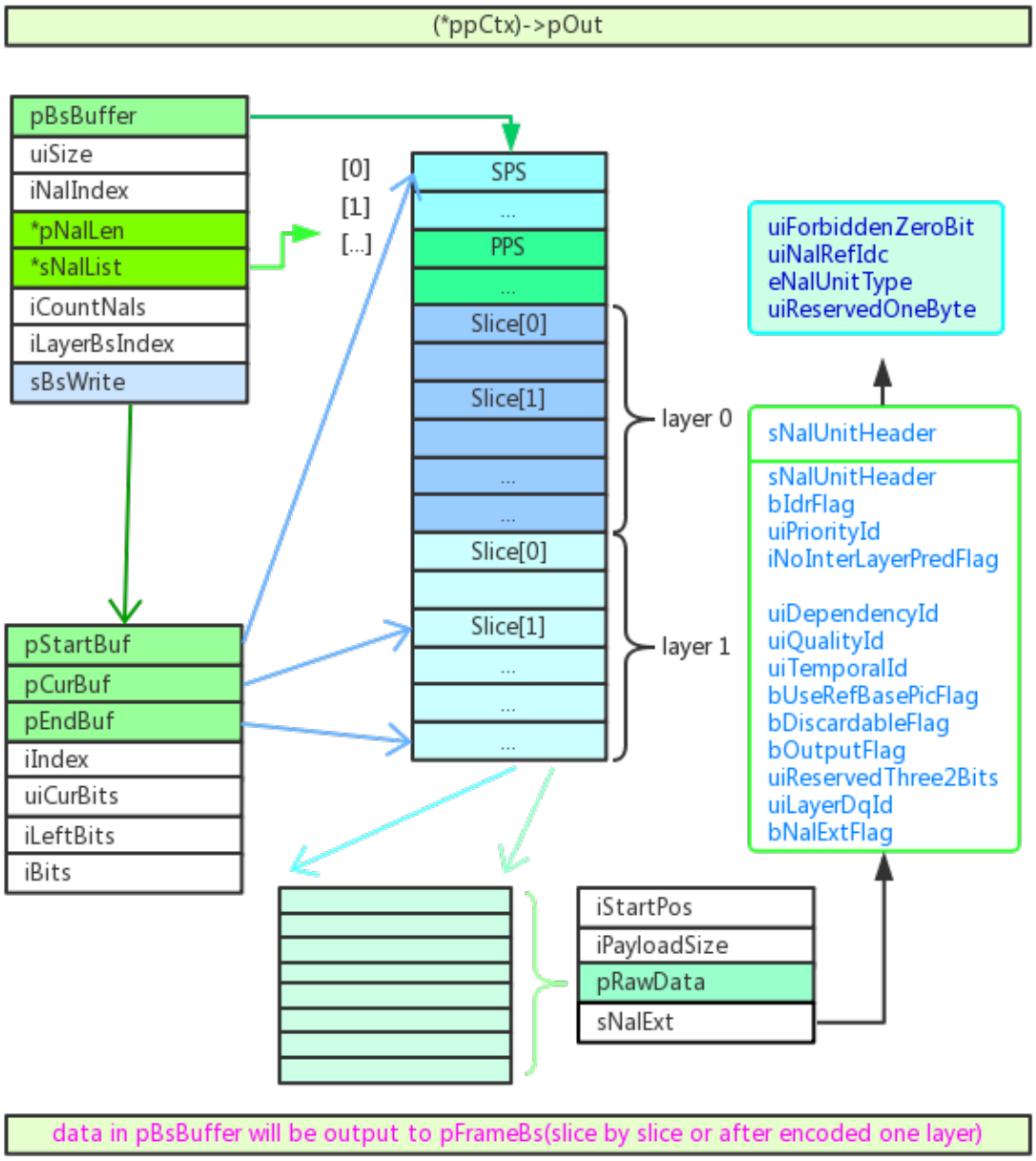
4.1. Slice Bs buffer



4.2. Frame bs buffer



4.3 pCtx-pOut


(*ppCtx)->pOut
pBsBuffer
uiSize
iNalIndex
*pNalLen
*sNalList
iCountNals
iLayerBsIndex
sBsWrite
[0]
[1]
[...]
SPS
...
PPS
...
Slice[0]
Slice[1]
...
...
Slice[0]
Slice[1]
...
...
...
layer 0
layer 1
uiForbiddenZeroBit
uiNalRefIdc
eNalUnitType
uiReservedOneByte
sNalUnitHeader
sNalUnitHeader
bIdrFlag
uiPriorityId
iNoInterLayerPredFlag
uiDependencyId
uiQualityId
uiTemporalId
bUseRefBasePicFlag
bDiscardableFlag
bOutputFlag
uiReservedThree2Bits
uiLayerDqId
bNalExtFlag
pStartBuf
pCurBuf
pEndBuf
iIndex
uiCurBits
iLeftBits
iBits
iStartPos
iPayloadSize
pRawData
sNalExt
data in pBsBuffer will be output to pFrameBs(slice by slice or after encoded one layer)

6. Other design

6.1. dynamic slice, slice boundary strategy