SISCI API INTERFACE

Roy Nordstrøm
Pci-support@dolphinics.com
Dolphin Interconnect Solutions

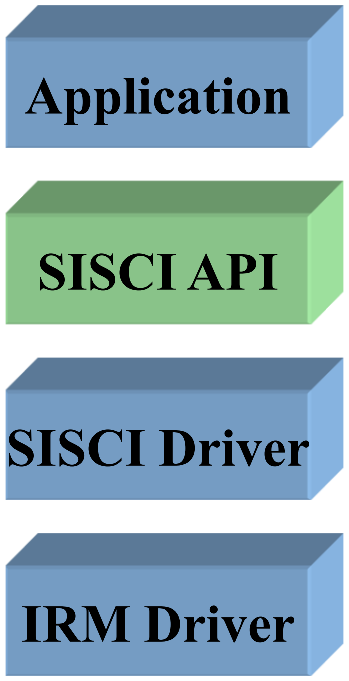
Version 2, IX

Porting to SISCI

- The SISCI API provides a powerful interface to migrate embedded applications to a Dolphin Express network.
- SISCI is fully supported on Dolphin IX, DX and SCI networks unless explicitly noted.
- Dolphin engineers have long time experience in porting applications to the SISCI API. It is strongly recommended to perform a design review with a Dolphin Engineer before starting initial coding.

SISCI API - Presentation overview

- SISCI API functions
- PIO Model
- DMA Model
- Remote Interrupt
- Lock Operations
- Privileged Operations
- Error Checking
- Callbacks



SISCI API

SISCI API

SISCI API

- SISCI –
  - **S**oftware **I**nfrastructure for **S**hared-Memory **C**luster **I**nterconnects
- Application Programming Interface (API)
- Developed in a European research project
- Shared Memory Programming Model
- User space access to basic NTB and Adapter properties
  - High Bandwidth
  - Low Latency
  - Memory Mapped Remote Access
  - DMA Transfer
  - Interrupts
  - Callbacks

SISCI API Features

- **Access to High Performance HW**
- **Highly Portable**
- **Cross Platform / Cross Operating system interoperable**
  - Big endian and little endian machines can be mixed
    - User data not converted
- **Simplified Cluster Programming**
- **Flexible**
- **Reliable Data transfers**
- **Host bridge / Adapter Optimization in libraries**

SISCI API - Handles

SISCI API
HANDLES

SISCI API – Handles – Basic SISCI Types

- Working with remote shared memory, DMA transfer and remote interrupt, requires the use of logical entities like devices, memory segments and DMA queues.
- Each of these entities is characterized by a set of properties that should be managed as an unique object in order to avoid inconsistencies.
- To hide the details of the internal representation and management of such properties to an API user, a number of handles / descriptors have been defined and made opaque

SISCI API - Handles

- **sci_desc_t**
  - It represents an SISCI virtual device, that is a communication channel with the driver. It is initialized by calling the function SCIOpen.
- **sci_local_segment**
  - It represent a local memory segment and it is initialized when the segment is allocated by calling the function SCICreateSegment()
- **sci_remote_segment**
  - It represent a segment residing on a remote node. It is initialized by calling either the function SCIConnectSegment() or SCIConnectSCISpace()

SISCI API - Handles

- **sci_map_t**
  - ➢ It represents a memory segment mapped in the process address space. It is initialized by calling either the function SCIMapRemoteSegment or the function SCIMapLocalSegment.
- **sci_sequence_t**
  - ➢ It represents a sequence of operations involving error checking with remote nodes. It is used to check if errors have occurred during a data transfer. The handle is initialized by calling the function SCICreateMapSequence

SISCI API - Handles

- **sci_dma_queue**
  - It represent a chain of specifications of data transfers to be performed using DMA. It is initialized by calling the function SCICreateDMAQueue.
- **sci_local_interrupt**
  - It represents an instance of an interrupt that an application has made available to remote nodes. It is initialized when the interrupt is created by calling the function SCICreateInterrupt.
- **sci_remote_interrupt**
  - It represents an interrupt that can be trigged on a remote nodes. It is initialized when the interrupt is created by calling the function SCIConnectInterrupt.

SISCI API

ERROR CODES

ERROR CODES

- Most of the SISCI API functions returns an error code as an output parameter to indicate if the execution succeeded or failed
- SCI_ERR_OK is returned when no errors occurred during the function call.
- The error codes are collected in an enumeration type called *sci_error_t*
- The error codes are specified in the DIS/src/SISCI/src/sisci_error.h file

SISCI API

FLAG OPTIONS

FLAG OPTIONS

- **Most SISCI API function have a flag option parameter**
  - SCI_FLAG_ ...
  - The flags options are is specified in the DIS/src/SISCI/api/sisci_api.h header file
- **The default option for the flag parameter is 0**
  - NO_FLAGS
    - The flag is commonly used, but not defined in the SISCI API
    - #define NO_FLAGS 0

SISCI API

EXAMPLE PROGRAMS

SISCI API – Example programs

- **Simple example applications are made for the usage of the SISCI API interface**
- **Located in the DIS/src/SISCI/cmd/examples/**
- **Test and benchmark application programs are located in the DIS/src/SISCI/cmd/test directory**
    - Testing of the system
    - Benchmarking

SISCI API

SISCI API FUNCTIONS

SISCI API - SCIInitialize()

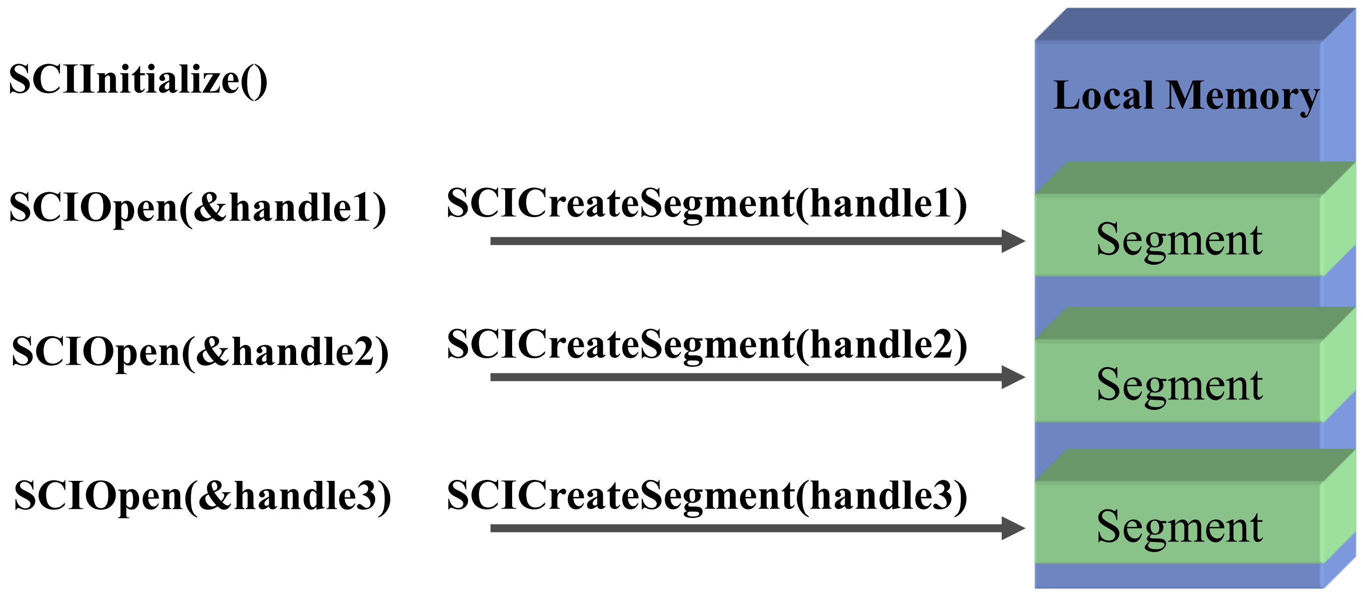
- **SCIInitialize()**
  - Initialize the SISCI Library
  - Checks the driver for CPU type, hostbridge, adapter type to select the optimized copy function for the system
  - Driver version checking
  - Allocates internal resources
  - The function must be called only once in the application program and before any other SISCI API function
  - If the SISCI library and the driver versions are not consistent, the function will return SCI_ERR_INCONSISTENT_VERSIONS

SISCI API - SCITerminate()

- **SCITerminate()**
  - Before an application is terminated, all allocated resources should be removed
  - Free the allocated resources that was created by the SCIInitialize()
  - Should be the last call in the application
  - Should be called only once in the application, independently of how many SISCI resources you use

SISCI API - SCIOpen()

- SCIOpen() creates a handle (virtual device)
- Each segment must be associated with a handle
- If the SCIInitialize() is not called before SCIOpen(), the function will return SCI_ERR_NOT_INITIALIZED



SISCI API - SCIClose()

- **SCIClose()**
  - Closes the virtual device
  - After this call, the virtual device becomes invalid and should not be used
  - If some resources is not deallocated, the SISCI driver will do the neccessary clean up at program exit

SISCI API – Initialization example

```
sci_error_t error;
sci_desc_t   vd;
SCIInitialize(NO_FLAGS,&error);
if (error != SCI_ERR_OK) {
    /* Initialization error */
    return error;
}
SCIOpen(&vd,NO_FLAGS,&error);
if (error != SCI_ERR_OK) {
    /* Error */
    return error;
}
/* Use the SISCI API */


SCIClose(vd,NO_FLAGS,&error);
SCITerminate();
```

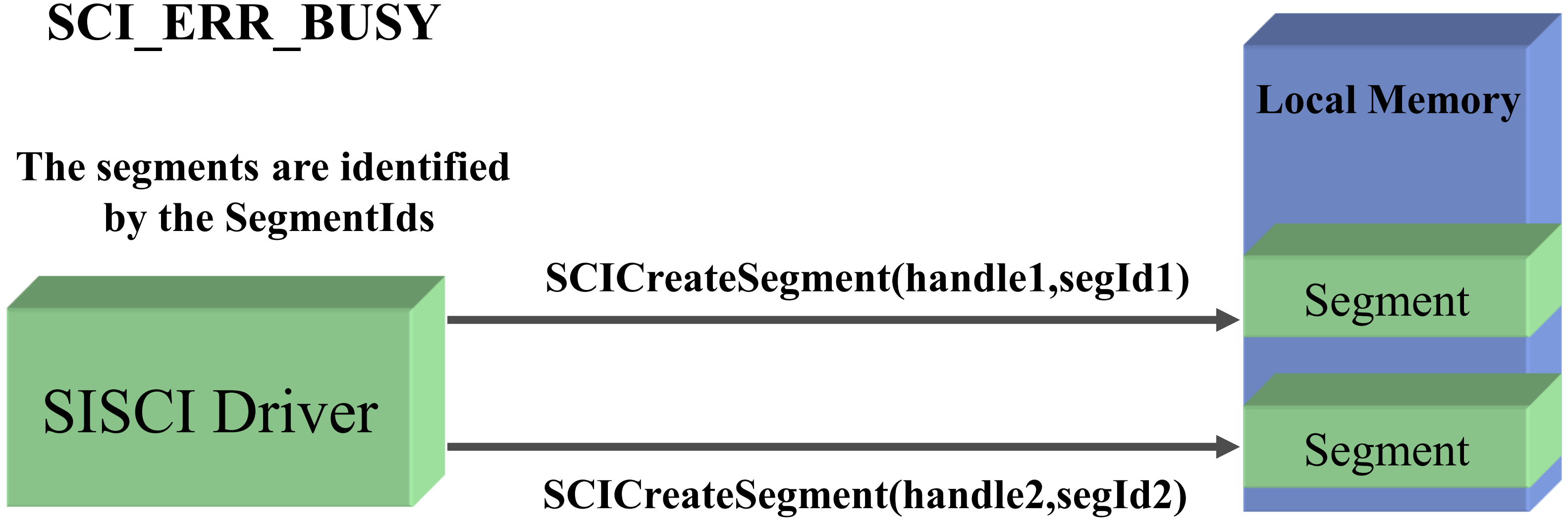
PIO MODEL

SISCI API - PIO Model

- **What is PIO?**
  - The possibility to have access to physically memory on another machine is the characteristic and the advantage of the Dolphin Express technology.
  - If the piece of memory is also mapped in the addressable space of a local process, a data transfer is as simple as a memcpy()
  - In such a case, it is the CPU that actively reads from or writes to remote memory using load/store operations
  - Once the mapping is created, the driver is not involved in the data transfer
  - This approach is known as Programmed I/O (PIO)

SISCI API - SCICreateSegment()

- The allocation of a segment on the local host is done with the function SCICreateSegment()
- Allocates contiguous memory
- The segment is identified by the segmentId
- The SISCI driver creates a list of local segments
- The segmentId for each segment must be unique on the local machine
- If segmentId already exist, the SCICreateSegment() will return SCI_ERR_BUSY


The segments are identified by the SegmentIds
SCICreateSegment(handle1,segId1)
SCICreateSegment(handle2,segId2)
SISCI Driver
Local Memory
Segment
Segment

SISCI API - SCIRemoveSegment()

- **SCIRemoveSegment()**
  - This function will free the resources used by a local segment
  - The physical memory is deallocated if the segment was created with SCICreateSegment()

SISCI API - Creating SegmentIds

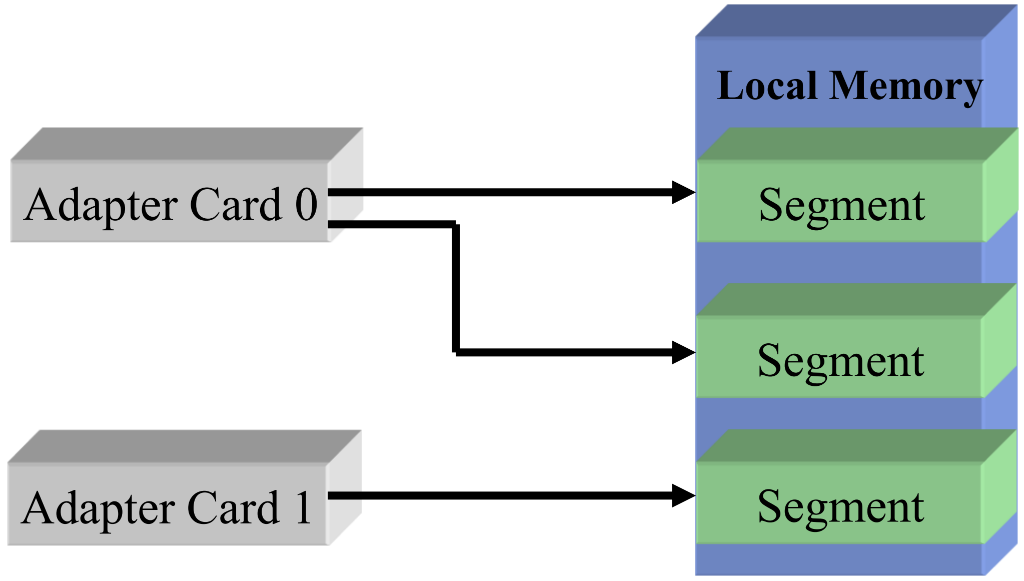
- A segmentId for a segment must be unique on the local machine (32 bit)
- A segment is identified by segmentId and nodeId
- Local and remote nodeId can be used to create a segmentId

- One possible way to create a segmentId

```
localSegmentId     = (localNodeId << 16)     | remoteNodeId  << 8 | KeyOffset;
remoteSegmentId = (remoteNodeId << 16) | localNodeId << 8      | KeyOffset;
```

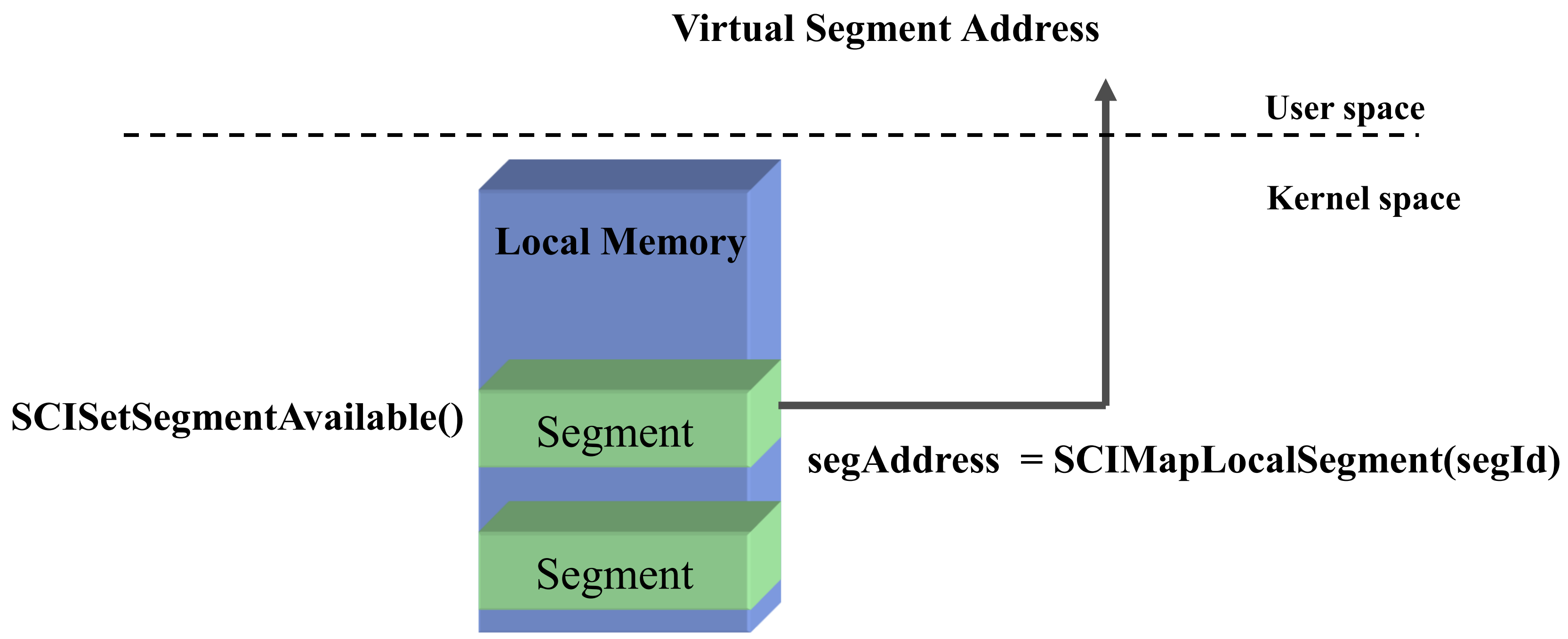
SISCI API - SCIPrepareSegment()

- **One machine can have several adapter cards.**
- **The function SCIPrepareSegment() prepare the segment to be accessable by the selected Dolphin adapter**



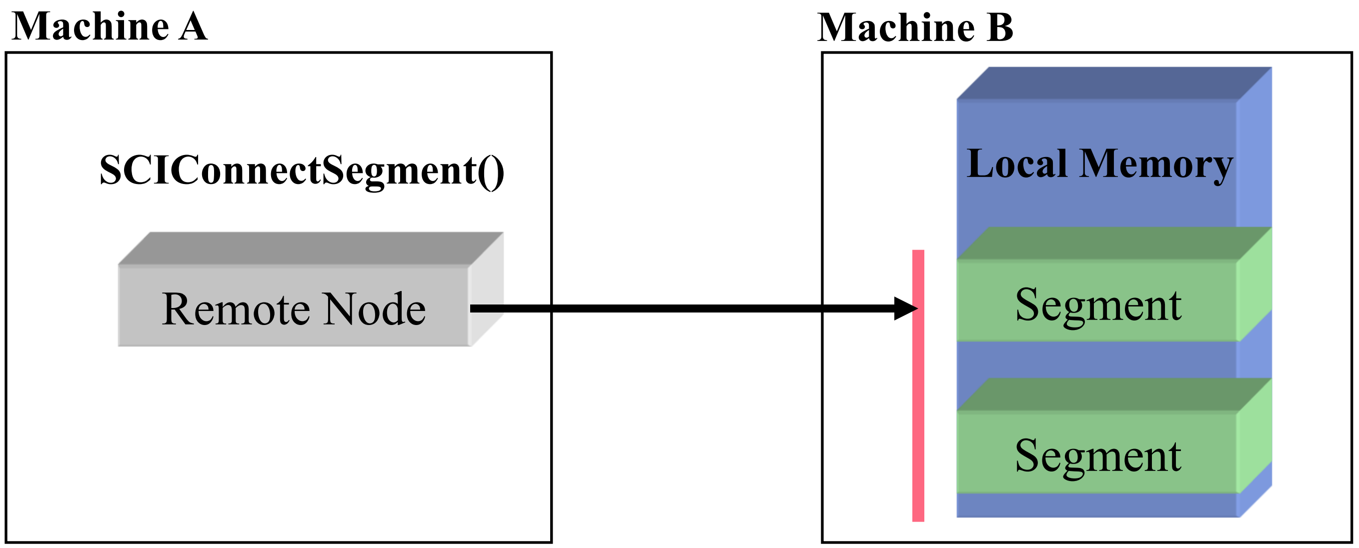
SISCI API - SCIMapLocalSegment()

- **SCIMapLocalSegment() maps the local segment into the applications virtual address space**



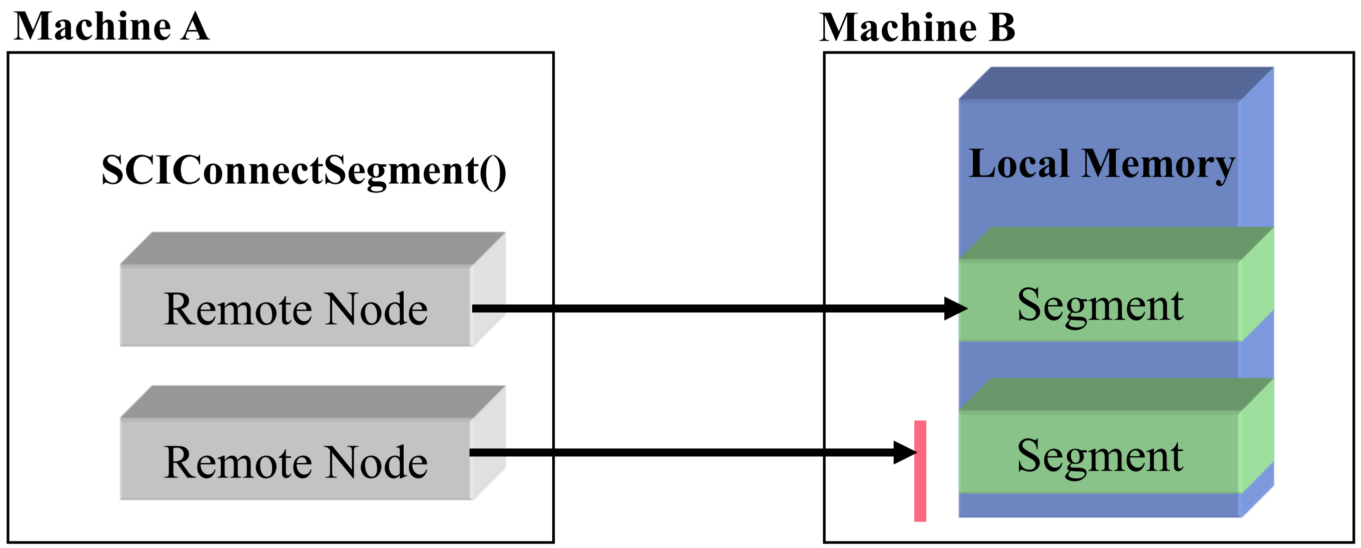
SISCI API - SCISetSegmentAvailable()

- **The function SCISetSegmentAvailable makes a local segment visible to the remote nodes**
- **The segment must set the local segment available to allow remote connection**



SISCI API - SCISetSegmentUnavailable()

- The function SCISetSegmentUnavailable hides an available segment from the remote nodes
- No new connection will be accepted on that segment
- The call to SCISetSegmentUnavailable doesn't affect existing remote connections, which are not even aware of the change.

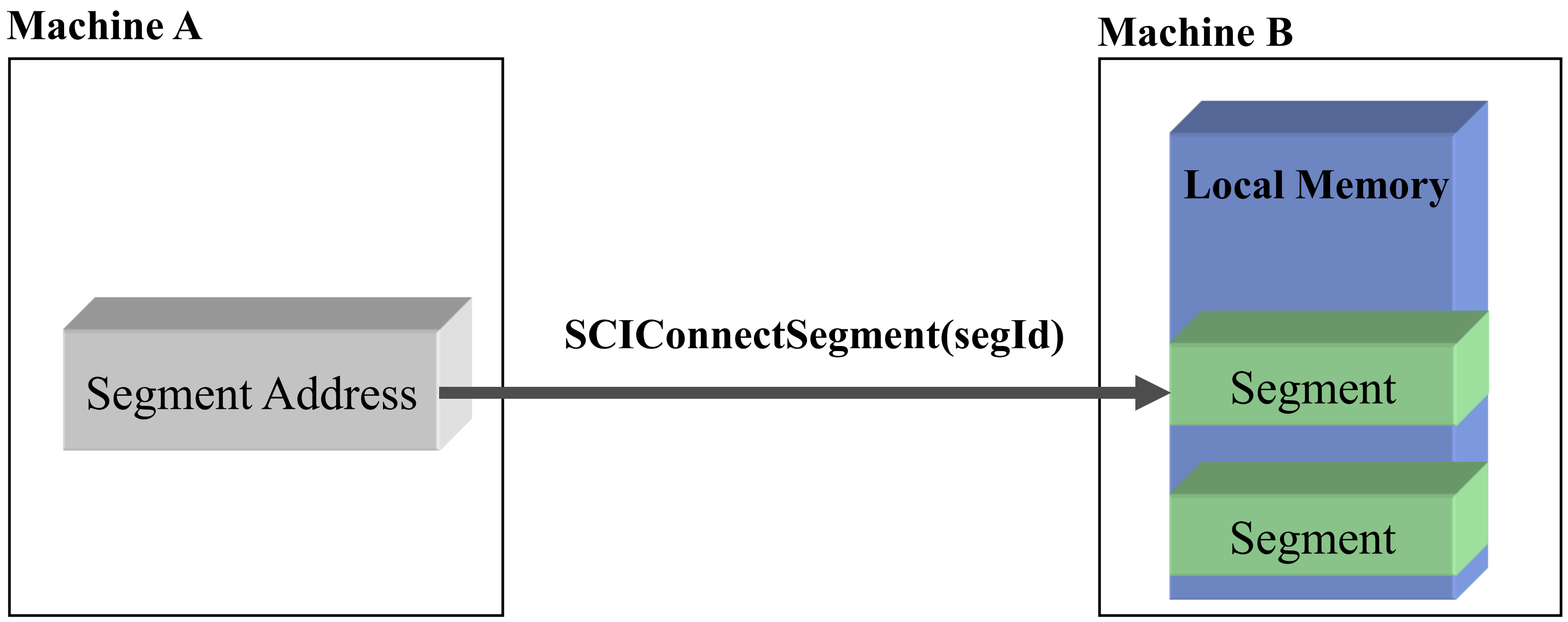


SCISetSegmentUnavailable() - Flag options

- **If SCI_FLAG_NOTIFY is specified, the operation is notified to the remote node connected to the local segment**
  - In this case, the remote node should disconnect
- **If the flag SCI_FLAG_FORCE_DISCONNECT is specified, the remote nodes are forced to disconnect.**

SISCI API - SCIConnectSegment()

- SCIConnectSegment() connects to a segment made available on a remote node
- Creates and initializes a handle for the connected segment



SISCI API - SCIConnectSegment()

- **The call SCIConnectSegment() must be called in a loop**
- **The status of the remote segment is not known**
  - The segment is not created
  - The remote node is still booting
  - The driver is not yet loaded

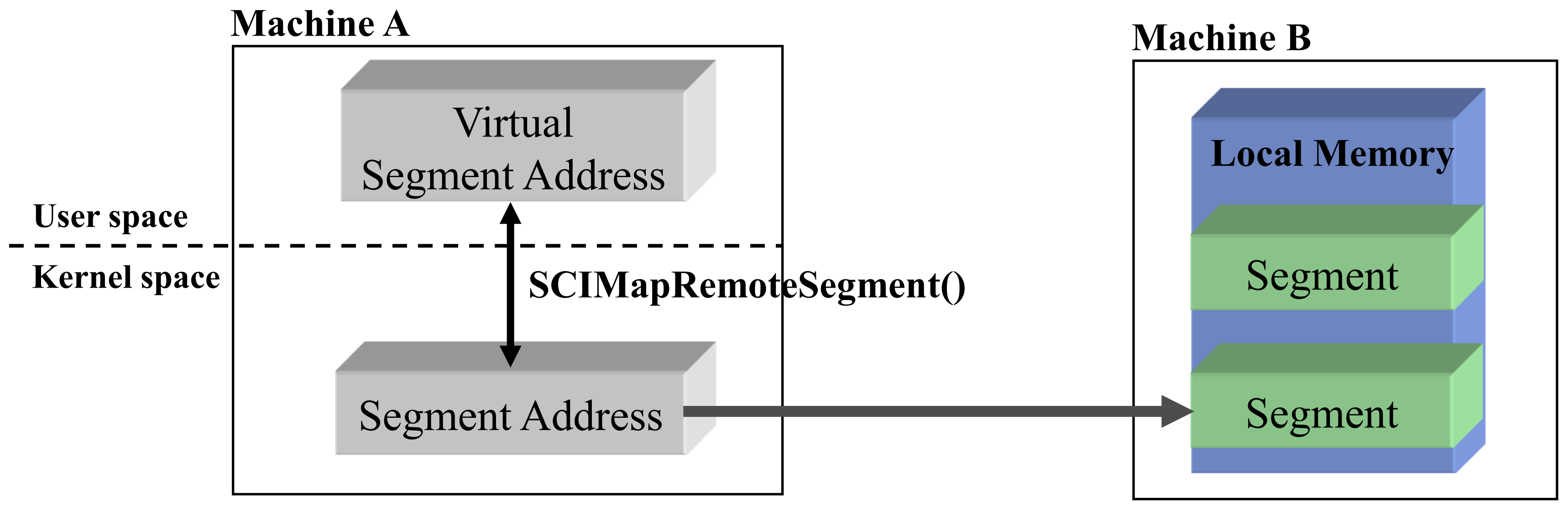
```
do {
        SCIConnectSegment(&error);
           /* Sleep before next connection attempt */
         if (error == SCI_ERR_ILLEGAL_PARAMETER) break;
         sleep(1);
} while (error != SCI_ERR_OK) ;
```

SISCI API - SCIDisconnectSegment()

- **SCIDisconnectSegment()**
  - The function disconnects from a remote segment connected by the calls SCIConnectSegment() or SCIConnectSCISpace()
  - If the segment was connected using SCIConnectSegment(), the execution of SCIDisconnectSegment() also generates an SCI_CB_DISCONNECT event directed to the application that created the segment.
  - If the Segment is still mapped, the function will return SCI_ERR_BUSY

SISCI API - SCIMapRemoteSegment()

- SCIMapRemoteSegment maps an area of a remote segment connected with either SCIConnectSegment() into the addressable space of the program (user space) and returns a pointer to the beginning of the mapped segment



SISCI API - SCIMapRemoteSegment()

- **It is possible to to map only a part of the segment by varying the the *size* and *offset* parameters, with the constraint that the sum of the size and offset does not go beyond the end of the segment**

- **Once a memory segment is available, i.e. you have a handle to either local or remote segment resources, you can access the segment in two ways:**
  - Map the segment into the address space of your process and then access it as normal memory operations - e.g. via pointer operations or SCIMemCpy()
  - Use the Dolphin adapter DMA engine to move data (RDMA)

SCIUnmapSegment()

- **SCIUnmapSegment()**
  - Unmaps the segment from the program's address space (user space) that was mapped either with SCIMapLocalSegment() or SCIMapRemoteSegment()
  - It destroy the corresponding handle
  - If the error return value is SCI_ERR_BUSY, the segment is in use

SISCI API – SCIGetRemoteSegmentSize()

- **SCiGetRemoteSegmentSize()**
  - Return the size of the remote segment after a connection has been established with SCIConnectSegment()

SISCI API - Data Transfer

DATA TRANSFER

SISCI API - Data Transfer

- **The virtual segment address can be used for data transfer**
  - Using the address directly doing CPU load/store operations
  - Using SCIMemCpy()

- **If the function succeeds the return value is a pointer to the beginning of the mappped segment**
- **The address can be used directly to transfer data**
  - *remoteAddress = data;
- **Note that the address pointer is declared as *volatile* to prevent the compiler from doing wrong optimization of the code**

```
volatile *unsigned int remoteMapAddr;
remoteMapAddr = (volatile *unsigned int)SCIMapRemoteSegment();

for (i=0; i < numberOfStores; i++) {
        remoteMapAddr[i] = i;
}
```

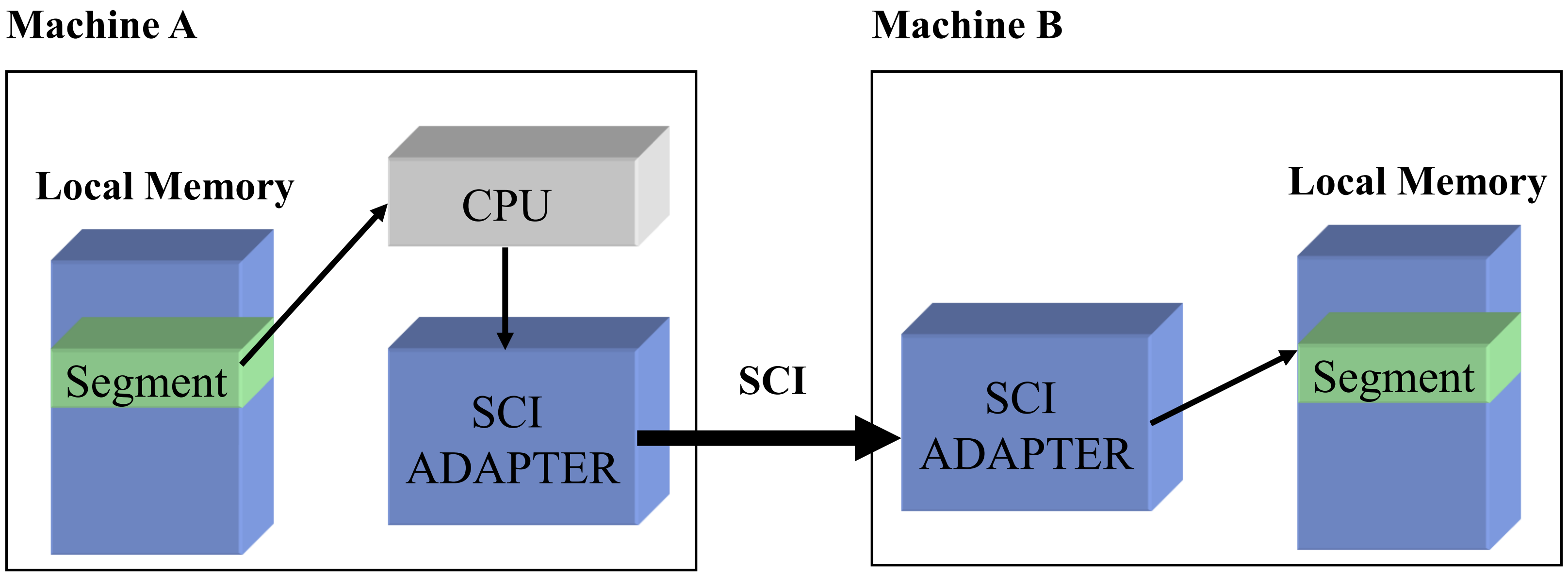
SISCI API - SCIMemCpy()

- **The SCIMemCpy() use PIO for data transfer**
- **The function is optimized for the CPU and various hostbridges**
- **Most library copy functions are written as assembler functions**
- **Transfer a specified block of data**
- **Flag option to enable error checking.**
- **Use MMX and SIMD registers to optimize the data transfer**

- **Note: Do note use SCIMemCopy()**
  - Old implementation and not very efficient
    - – Allocates error checking resources for each call

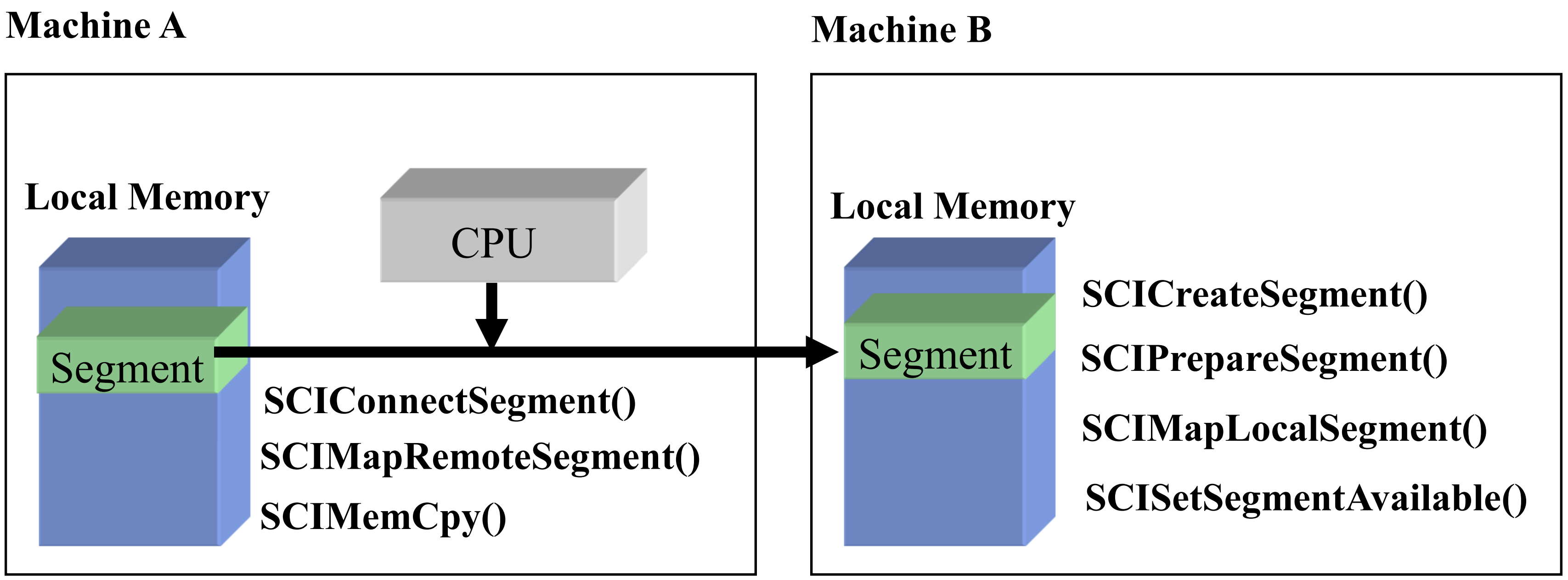
SISCI API - SCIMemCpy()

- **SCIMemCpy() use MMX and SIMD registers to optimize the data transfer**
  - This is auto detected in SCIInitialize()

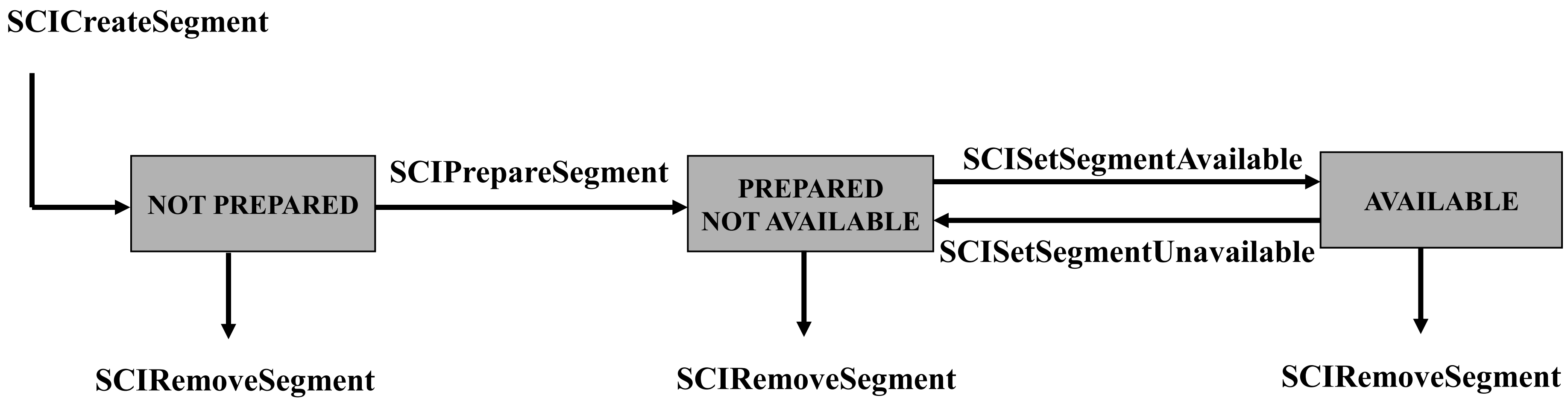


SISCI API - PIO Model

- PIO model call sequence on client and server node



State diagram for a local segment



SHARED LOCAL SEGMENT

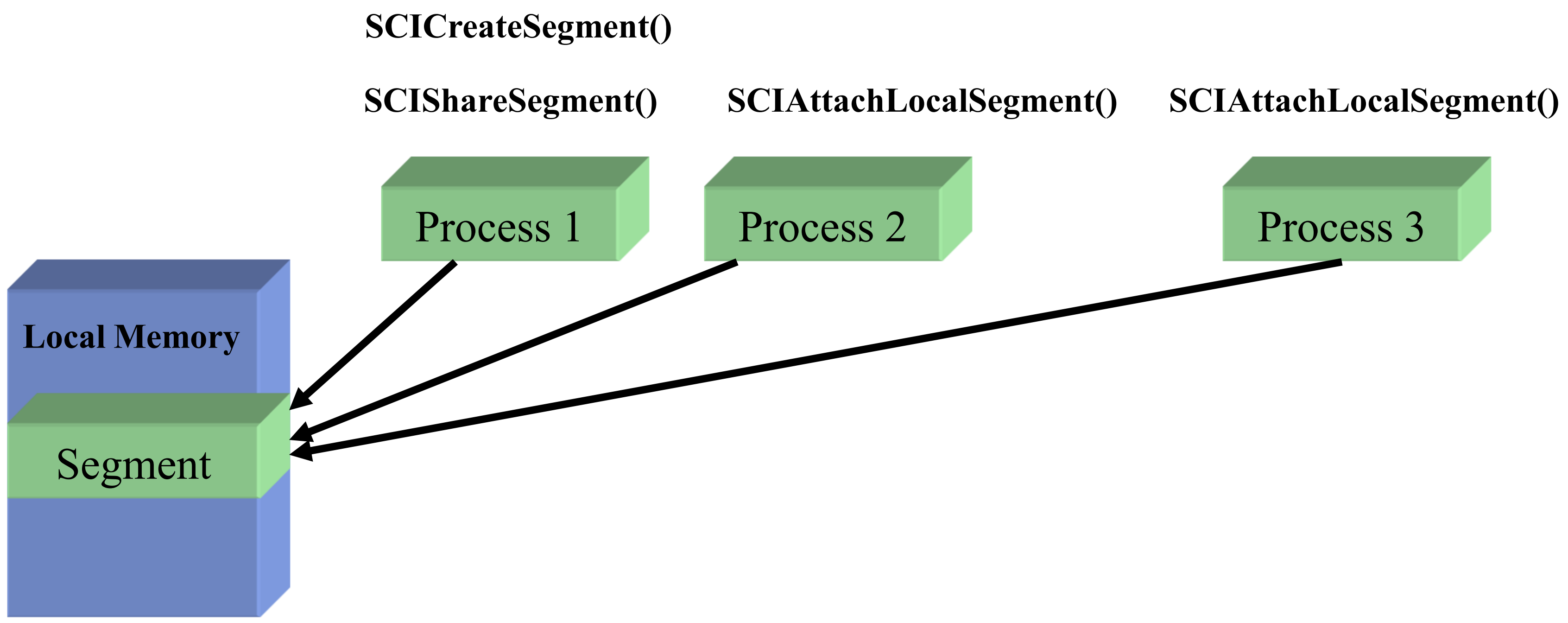
SISCI API - SCIShareSegment()

- **SCIShareSegment()**
  - permits other application to "attach" to an already existing local segment, implying that two or more application can share the same local segment

SISCI API - SCIAttachLocalSegment()

- **SCIAttachLocalSegment()**
  - SCIAttachLocalSegment() causes an application to "attach" to an already existing local segment, implying that two or more applications are sharing the same local segment
  - The application which originally created the segment ("owner") must have preformed a SCIShareSegment() in order to mark the segment "shareable".
  - The local segment is identified using the segmentId
  - If multiple local processes share the segment, all attached processes must perform a SCIRemoveSegment() before the segment is physically removed
  - The creator and all attached processes share ”ownerships” and have the same permissions

SISCI API - SCIShareSegment()



LOCK OPERATIONS

SISCI API - Lock Operations

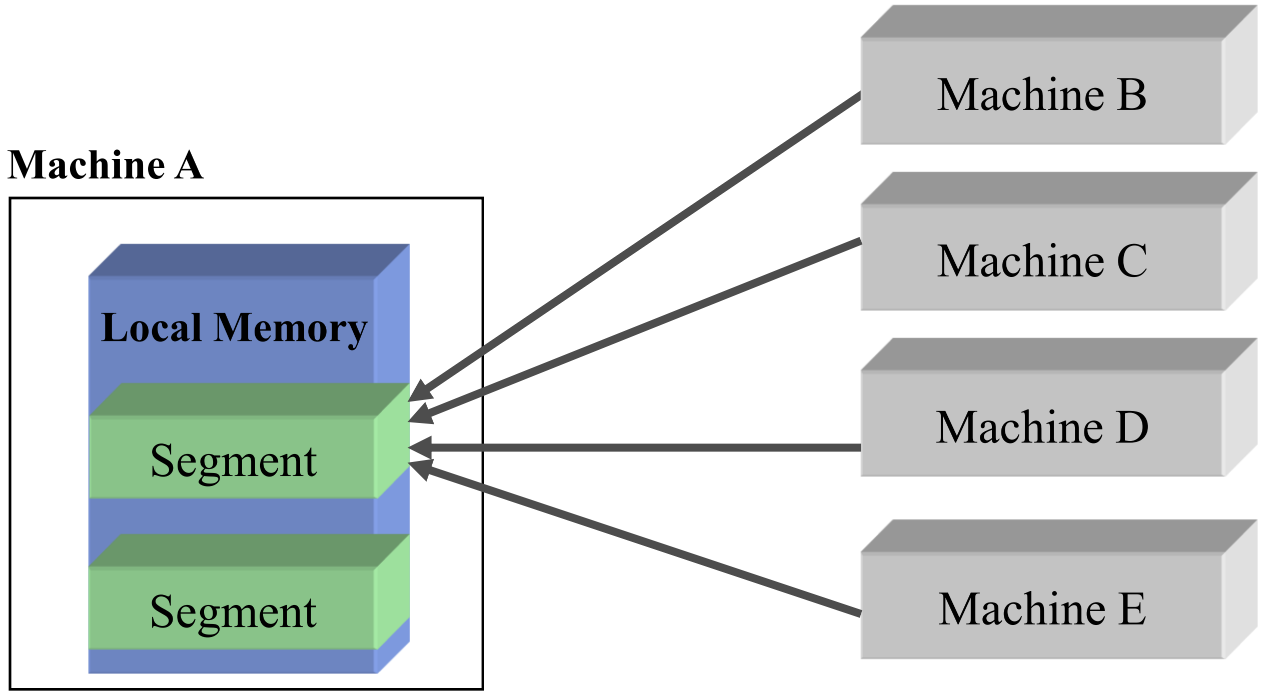
- A lock segment is created as a regular segment
- The flag option SCI_FLAG_LOCK_OPERATION to the function SCIMapRemoteSegment() maps the segment as a lock segment.

  SCIMapRemoteSegment(SCI_FLAG_LOCK_OPERATION)

- Lock operations are ONLY available with the SCI/D Dolphin Interconnect

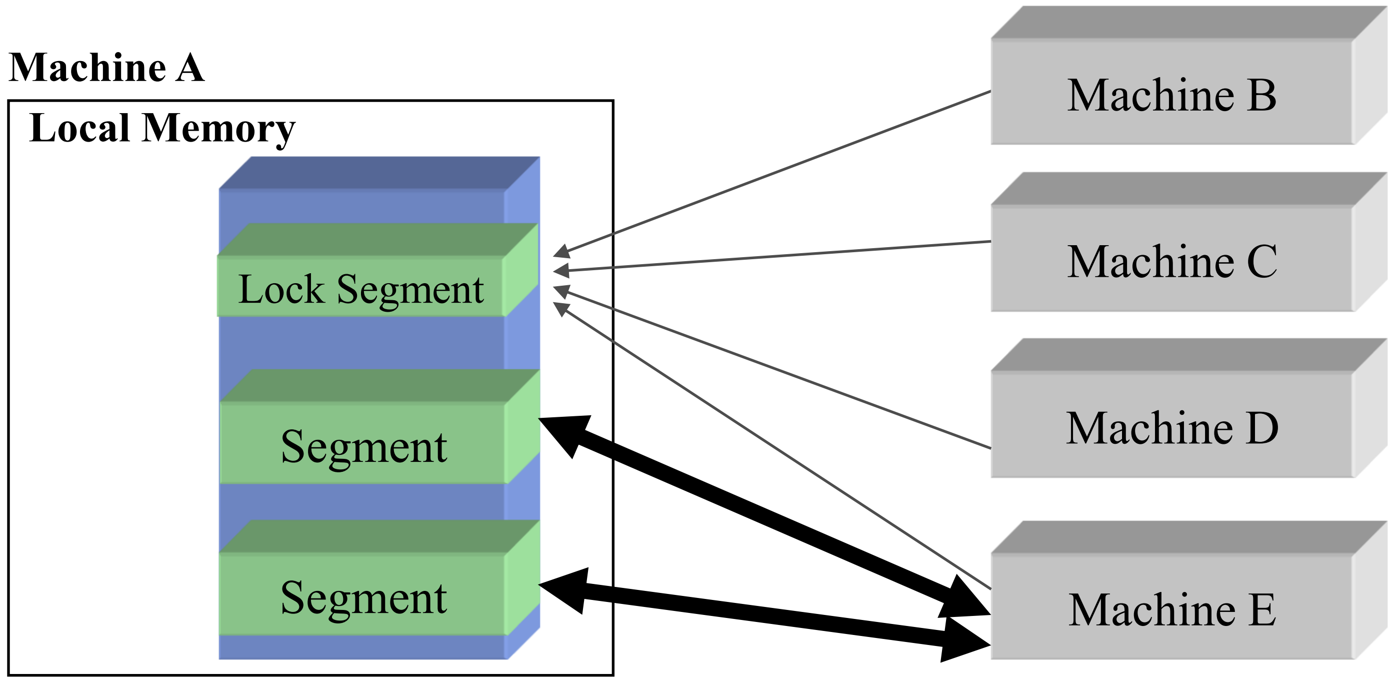
SISCI API - Multiple connections

- **A segment can have several connections**
- **The protocol must be implemented to ensure that only one machine access the memory (lock operation)**



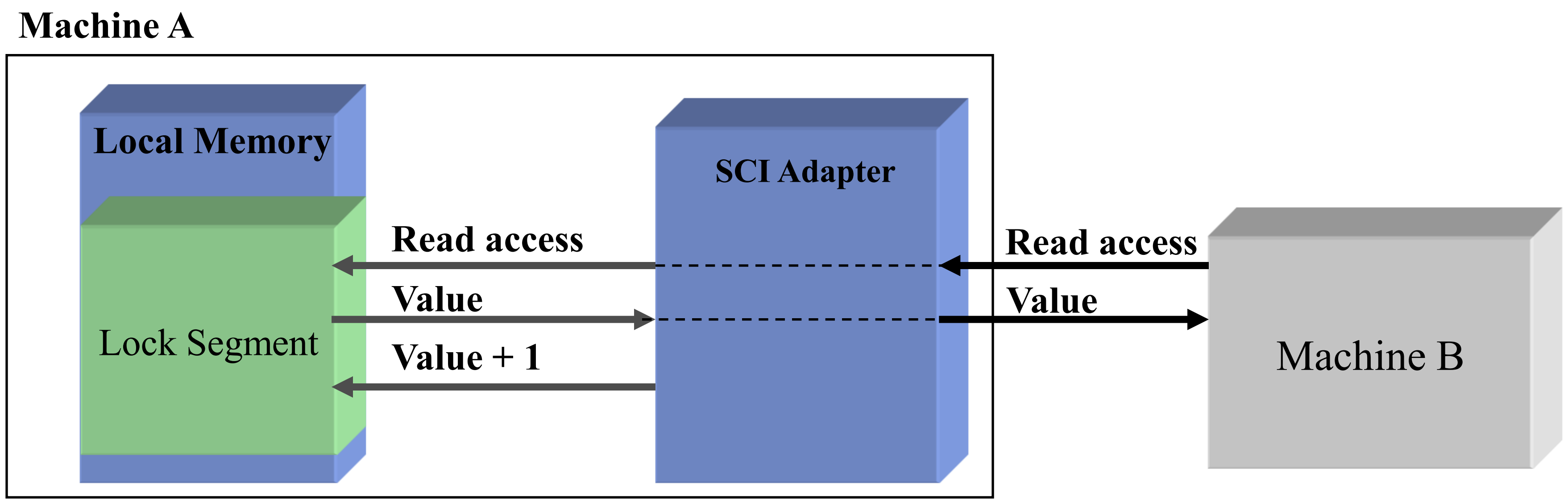
SISCI API - Multiple Connections and Lock Operations

- **The remote machines are trying to get the lock**
- **Machine E gets the Lock and have access to the remote segment**



SISCI API - Lock Operations

- The remote machine B reads the lock segment
- The value is returned to machine B
- The adapter increments the value and write it to the lock segment
- Atomic operation



SISCI API – Lock Operations

```
/* Try to get the lock */
readLockValue = remoteLockSeg[0];

if  (readLockValue == LOCK_INIT_VALUE) {
     /* Got the lock and ’owns’ the remote data segment */
     < Do the operations on the data segment >

    /* Release the lock */
    remoteLockSeg[0] = LOCK_INIT_VALUE;
} else {
     /* Another node got the lock */
}
```

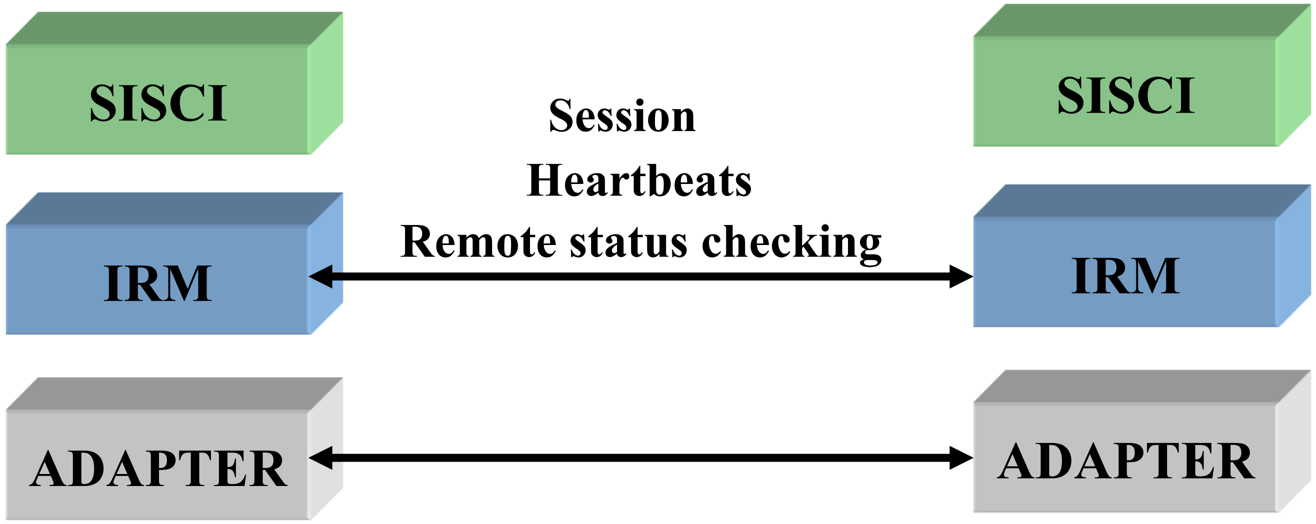
ERROR CHECKING

SISCI API – ERROR CHECKING

- The hardware protocols for IX, DX and SCI garantees that the data are delivered successfully when error checking mechanism is used and the sequence check returns SCI_ERR_OK
- Garanteed correct data delivery from the function call SCIStartSequence() and SCICheckSequence()
- The SCICheckSequence() flushes the write buffers from the CPU and wait for the outstanding requests
- The SCICheckSequence function returns when all outstanding packets have returned (Store Barrier)
- The error checking rate depends of the application and system
- Some overhead is added to the data transfer

SISCI API – Error Checking

- **A session is established between the nodes that communicates**
  - A heartbeat mechanism make sure that the remote node is alive
- **Periodically checking of the status of the remote node**
- **The error checking mechanism check the session status, the interrupt status register and the cable status.**



SISCI API – SCICreateMapSequence

- **SCICreateMapSequence()**
  - The function creates and initializes a new sequence descriptor that can be used to check for transmission errors
- **SCICreateMapSequence(remoteMap,&sequence, ...)**
  - Creates a sequence assosiated with a remoteMap
- **SCIStartSequence()**
  - Check the adapter status, SCI connection and the valid remote map

```
do {
        /* Start the data error checking */
        sequenceStatus = SCIStartSequence(sequence,....);
 } while (sequenceStatus != SCI_SEQ_OK) ;
```

SISCI API – SCIStartSequence

- **SCIStartSequence()**
  - The function performs the preliminary check of the error flags on the SCI adapter before starting a sequence of read and write operations on the mapped segment
  - Subsequent checks are done calling SCICheckSequence()
  - If the return value is SCI_SEQ_PENDING there is a pending error and the program is required to call SCIStartSequence until it succeeds before doing other transfer operations on the segment

SISCI API – SCICheckSequence

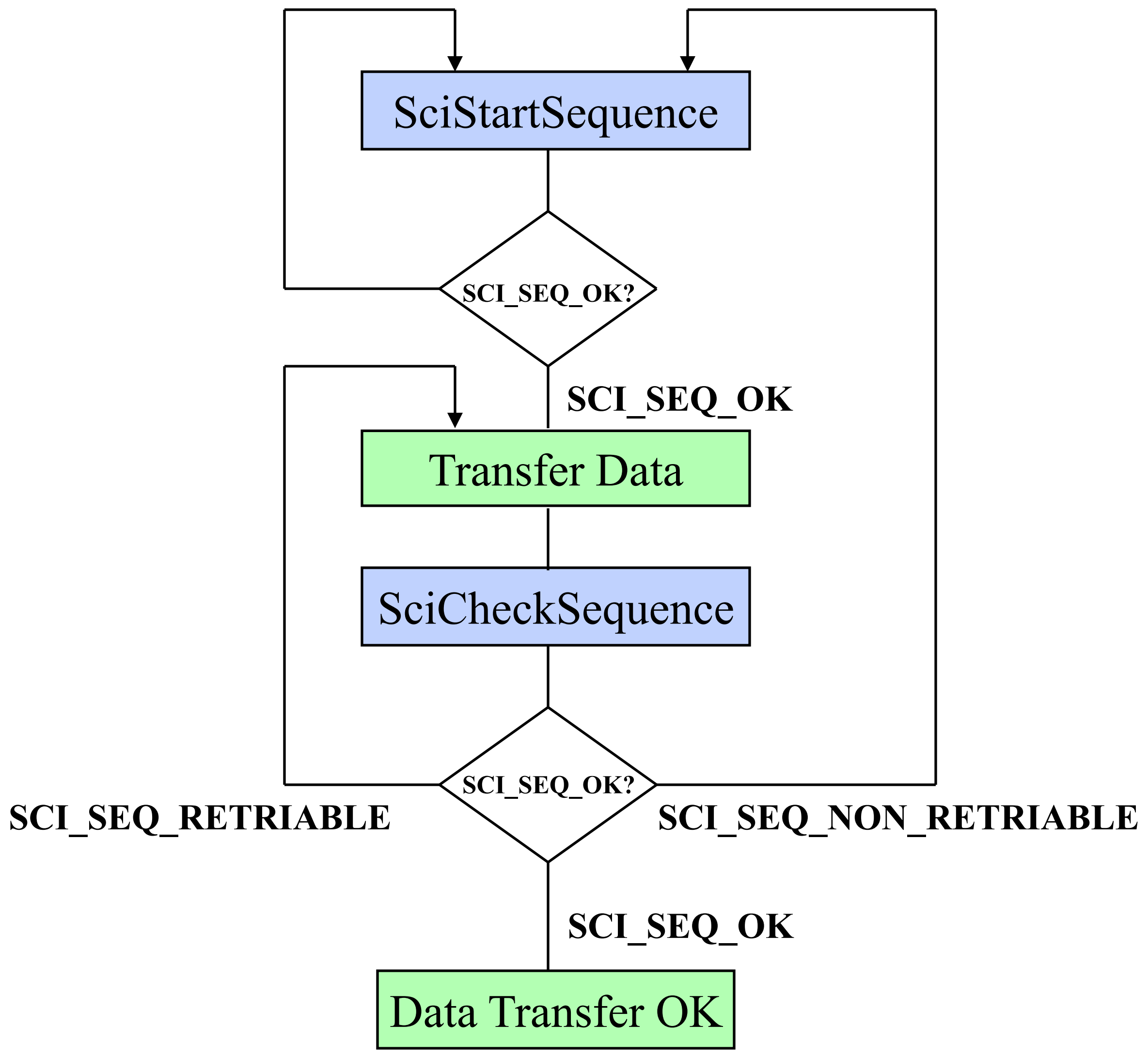
- **SCICheckSequence()**
  - The function checks if any error has occurred in the data transfer controlled by a sequence since the last check
  - The previous check can have been done by either calling either SCIStartSequence or the SCICheckSequence itself
  - The function can be invoked several times in a row without calling SCIStartSequence
  - By default SCICheckSequence also flushes the CPU write buffers and wait for all outstanding SCI transactions to complete. In other words it internally performs an action similar to the SCIStoreBarrier()
    - SCI_FLAG_NO_FLUSH
    - SCI_FLAG_NO_STORE_BARRIER

  - SCICheckSequence(SCI_FLAG_NO_FLUSH |
    SCI_FLAG_NO_STORE_BARRIER)

SISCI API – SCIStartSequence() / CheckSequence()

- **The status from the sequence functions can return four possible values:**
- **SCI_SEQ_OK**
  - The transfer was successful
- **SCI_SEQ_RETRIABLE**
  - The transfer failed due to non-fatal error but can be immediately retried (e.g. The system is busy because of heavy traffic)
- **SCI_SEQ_NON_RETRIABLE**
  - The transfer failed due to a fatal error (e.g. cable unplugged) and can be retried only after a successful call to SCIStartSequence()
- **SCI_SEQ_PENDING**
  - The transfer failed, but the driver hasn't been able to determine the severity of the error (fatal or non-fatal). SCIStartSequence() must be called until it succeeds

SISCI API – SCIStartSequence() / CheckSequence()



SISCI API – SCIStartSequence() / CheckSequence()

```
{/* Start the data error checking */
do {
        do
             sequenceStatus = SCIStartSequence(sequence,....);
         } while (sequenceStatus != SCI_SEQ_OK);

        /* The SCI connection is OK */
        <Do the data transfer>
        do {
               sequenceStatus = SCICheckSequence(sequence,....);
        while (sequenceStatus == SCI_SEQ_PENDING) ;

        if (sequenceStatus == SCI_SEQ_NON_RETRIABLE) {
                     <error handling>
                     break;
        }
} while (sequenceStatus != SCI_SEQ_OK);
/* Successful data transfer */
```

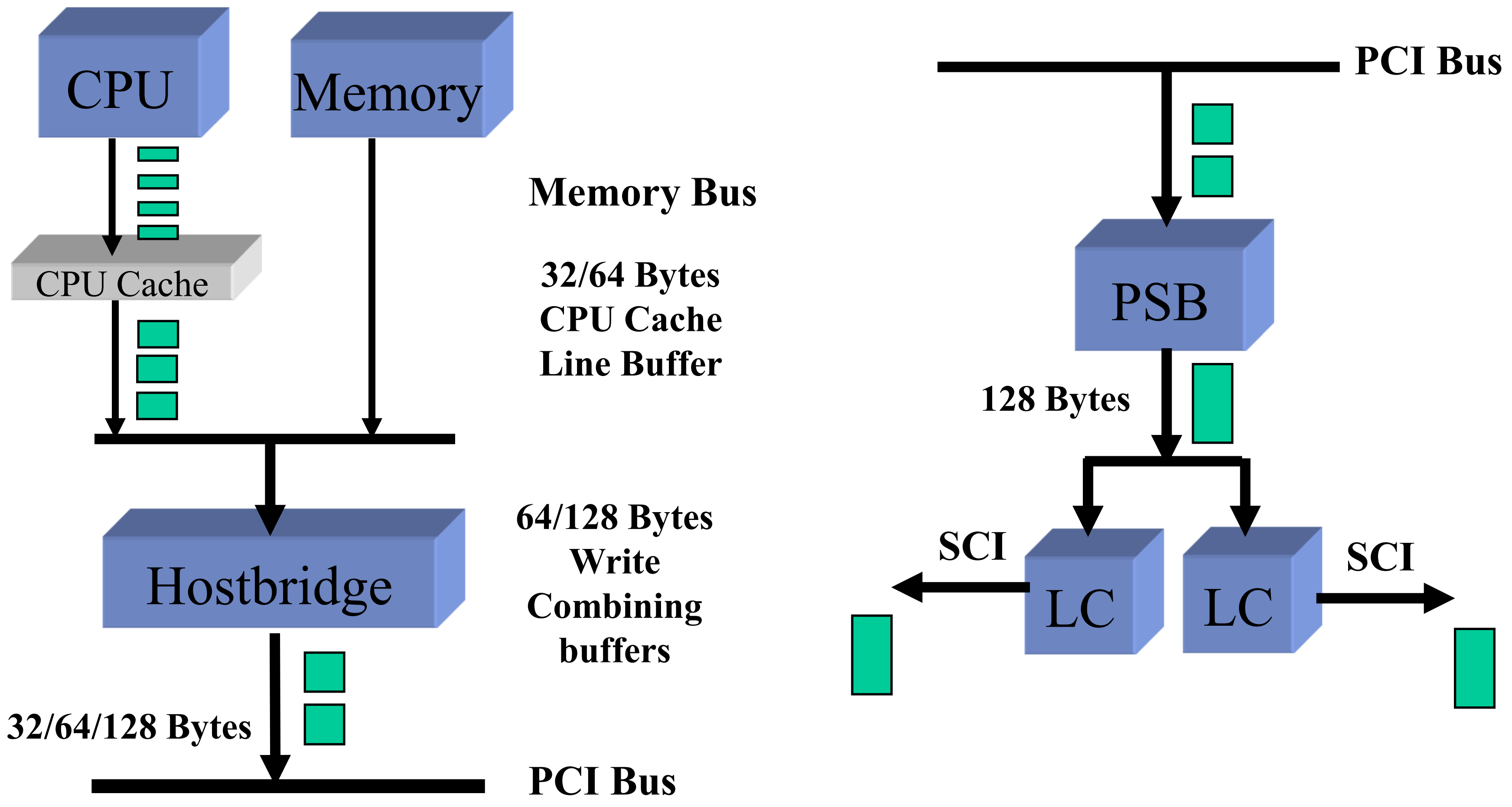
SISCI API – SISCI API

SCI FLUSH

SISCI API – SCIFlush()

- **SCIFlush()**
  - Flushes the data from the CPU buffer, cache, IO-system and from the adaper card
    - CPU flush
    - Store barrier
  - If flag option SCI_FLAG_FLUSH_CPU_BUFFERS_ONLY is specified, only the the CPU buffer/cache is flushed

SCIFlush() - Write Combining - SCI



SISCI API – SCIStoreBarrier()

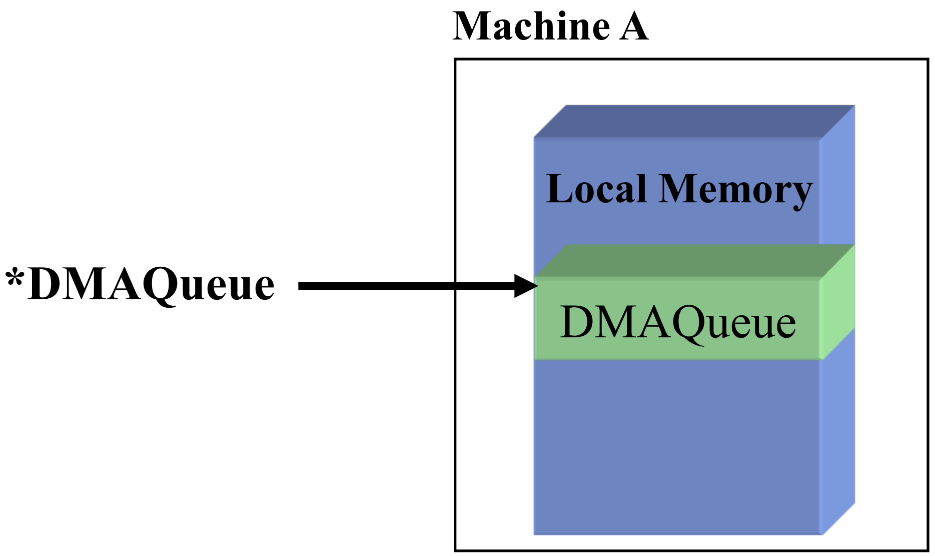
- **SCIStoreBarrier()**
  - Synchronize all the access to the mapped segment
  - The function flushes all outstanding transactions
  - The function does not return until all outstanding transactions has been confirmed

SISCI API

DMA MODEL

SISCI API - SCICreateDMAQueue()

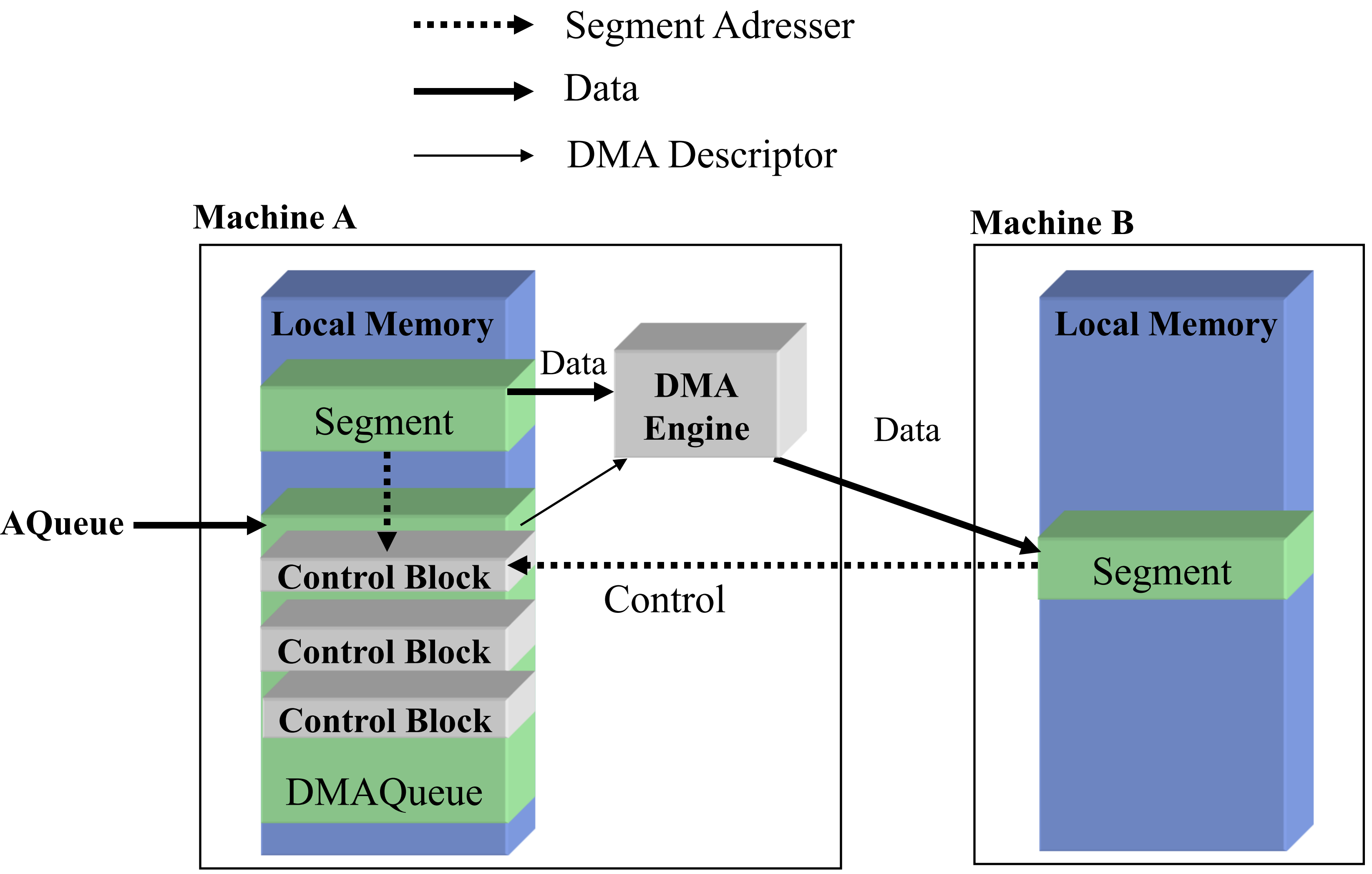
- **SCICreateDMAQueue()**
  - Allocates resources for a queue of DMA transfers
  - Creates, initializes and returns a handle for the new DMA queue



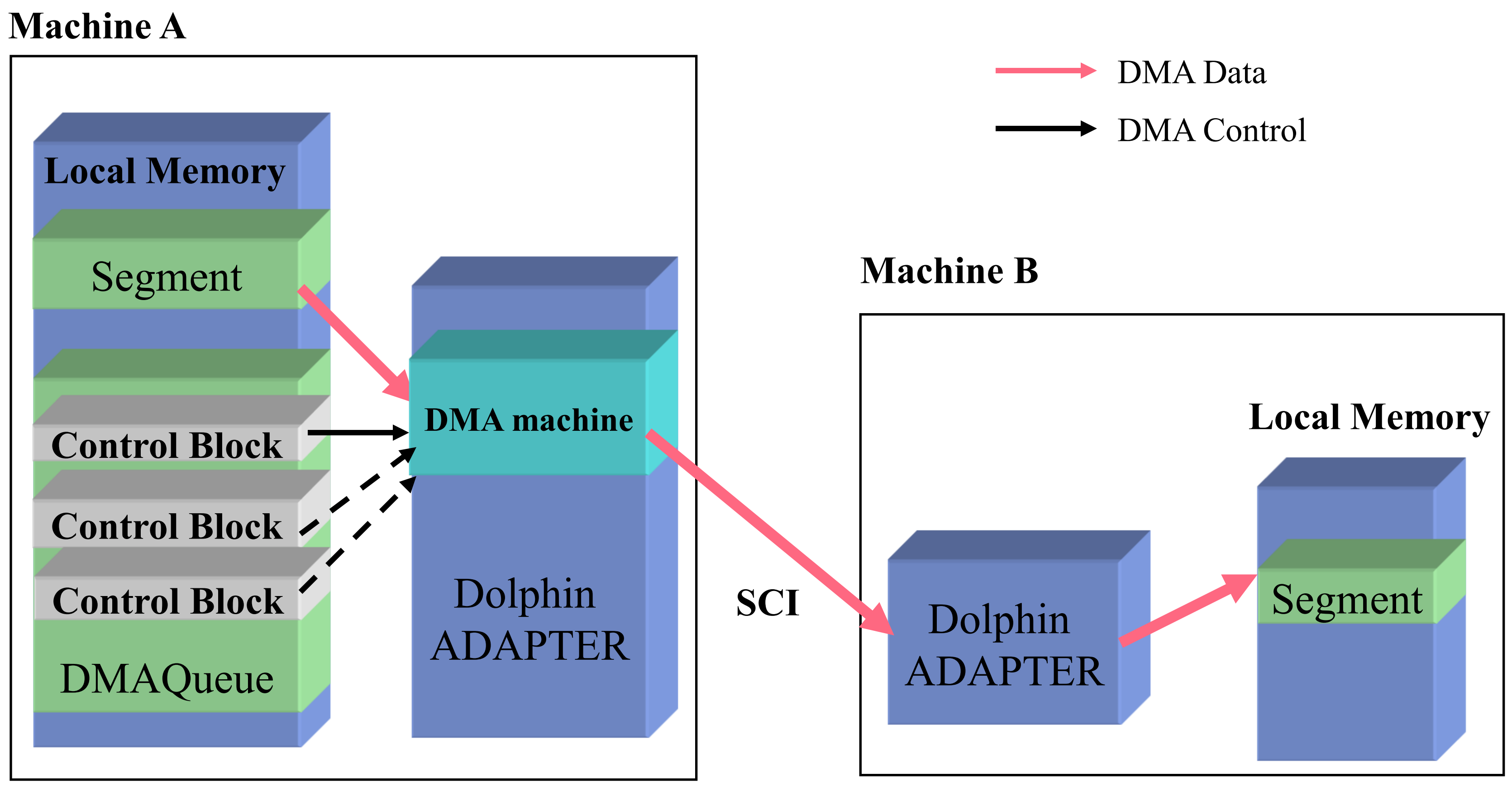
SISCI API - SCIEnqueueDMATransfer()

- **SCIEnqueueDMATransfer**
  - Adds a specification of the new future transfer to a DMA queue
  - Creates a control block in the memory for each DMA transfer
    - The control block contains the source and destination addresses and transfer size
    - The control blocks can be chained
  - Either the source or the destination of the transfer must be a local segment
  - By default the transfer operation is PUSH, i.e. from the local segment to the remote segment
  - DMA size alignment requirements is 8 bytes.

SISCI API - SCIEnqueueDMATransfer()



SISCI API - SCIPostDMAQueue()



SISCI API - SCIPostDMAQueue()

- **SCIPostDMAQueue()**
  - Starts the DMA machine on the Dolphin Adapter
  - Fetch the first control block from the memory
  - Starts the data transfer
  - Fetch the next control block from the memory

SISCI API - SCIWaitForDMAQueue()

- **SCIWaitForDMAQueue()**
  - The function blocks a program until a DMA queue has finished
    - Completion of the DMA transfer
    - An error occurred
  - The function returns the current state of the queue

SISCI API - SCIResetDMAQueue()

- **SCIResetDMAQueue()**
  - Reset or empties a DMA queue (without removing the queue)
  - The queue can be reused for another chain of transfers
  - This function not can be called when the queue state is POSTED.

SISCI API - DMA Model

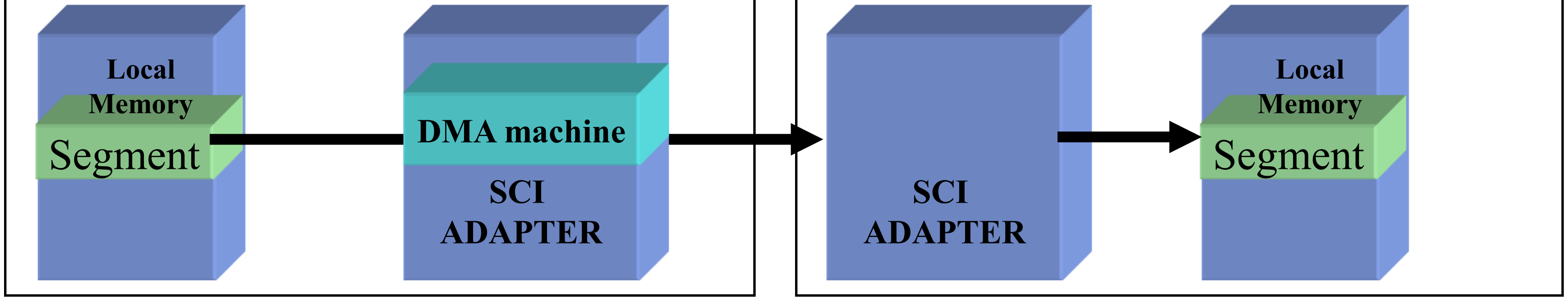
- DMA call sequence on client and server node

Machine A

SCIConnectSegment()
SCIMapRemoteSegment()
SCICreateDMAQueue()
SCIEnqueueDMATransfer()
SCIPostDMAQueue()
SCIWaitForDMAQueue()

Machine B

SCICreateSegment()
SCIPrepareSegment()
SCIMapLocalSegment()
SCISetSegmentAvailable()



SISCI API - DMA Optimization

- For each system function call, some overhead is added
- An optimized version can be used by combining several SISCI function calls into one function call

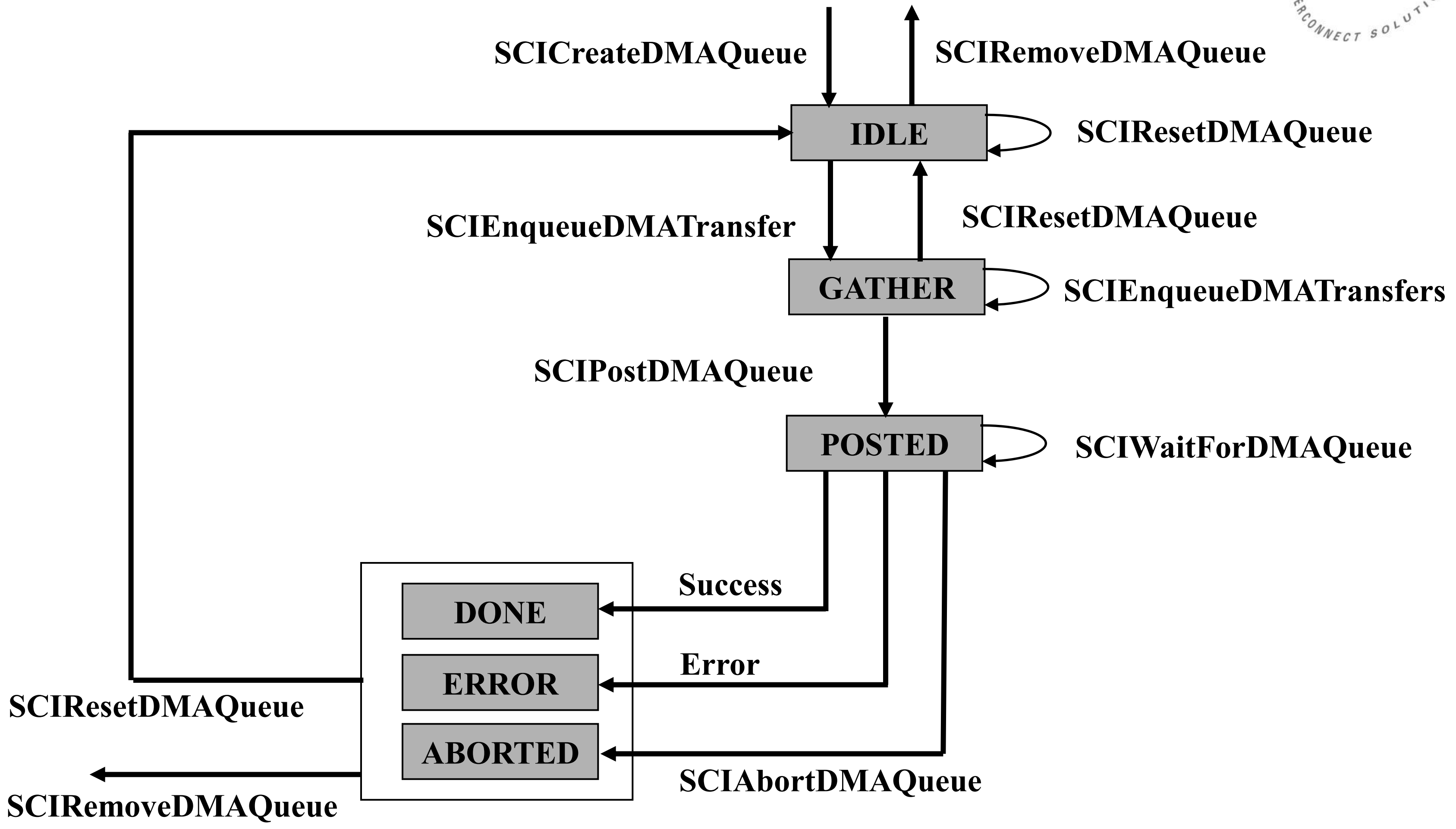
Alternative 1	Alternative 2 (optimized version)

SCICreateDMAQueue()
SCIEnqueueDMATransfer()
SCIPostDMAQueue()
SCIWaitForDMAQueue()
SCIResetDMAQueue()
	SCICreateDMAQueue()
	SCIEnqueueDMATransfer(SCI_FLAG_DMA_POST,
	SCI_FLAG_DMA_WAIT,
	SCI_FLAG_DMA_RESET)

SISCI API - SCIEnqueueDMATransfer()

```
SCIEnqueueDMATransfer(dmaQueue,
                      localSegment,
                      remoteSegment,
                      localOffset,
                      remoteOffset,
                      transSize,
                      SCI_FLAG_DMA_POST |
                      SCI_FLAG_DMA_WAIT |
                      SCI_FLAG_DMA_RESET,
                      &error);
```

SISCI API - State Diagram for DMA operations



SISCI API - SCIDMAQueueState()

- **SCIDMAQueueState()**
  - Returns the current state of the DMA queue
- **The user must poll the status to determine when the DMA has completed**
- **Can be used as an option to the SCIWaitForDMAQueue()**

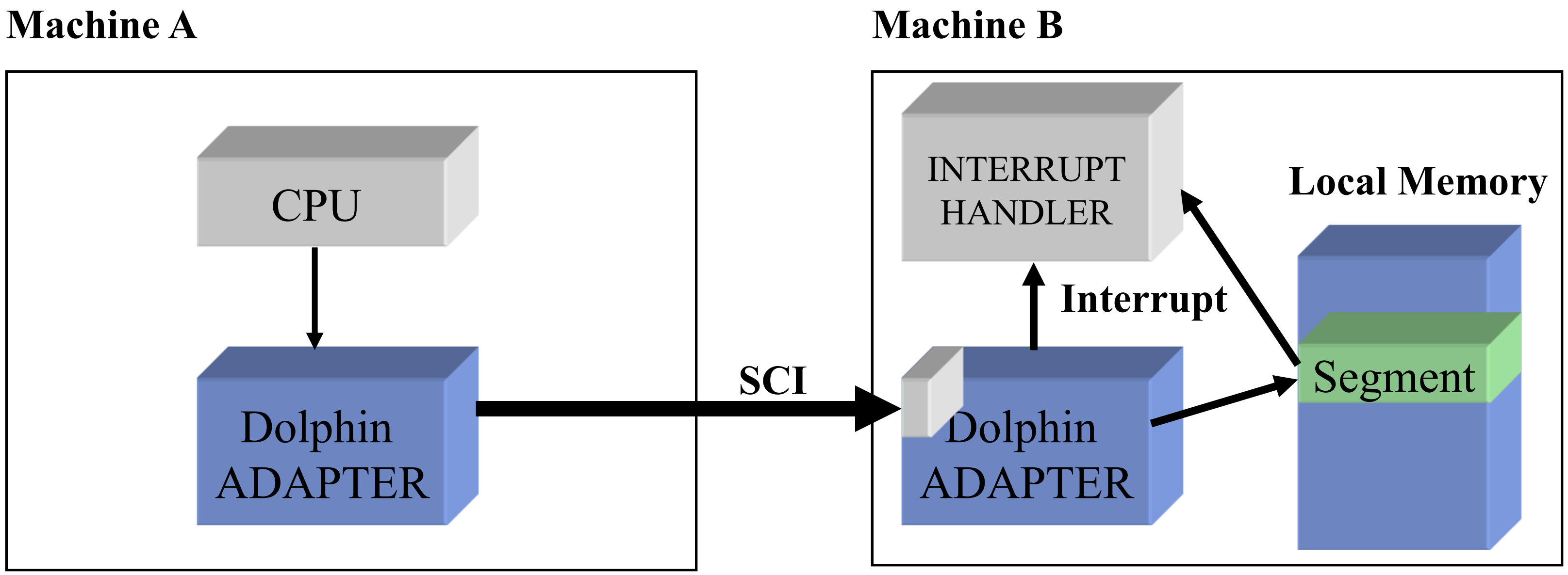
- **DMA queue states**
  - SCI_DMAQUEUE_IDLE
  - SCI_DMAQUEUE_GATHER
  - SCI_DMAQUEUE_POSTED
  - SCI_DMAQUEUE_DONE
  - SCI_DMAQUEUE_ABORTED
  - SCI_DMAQUEUE_ERROR

SISCI API

REMOTE INTERRUPTS

SISCI API – Interrupt Model

- **A write access to a remote register triggers an interrupt on the remote node**
- **The driver handles the interrupt and forwards the interrupt to the correct process**



SISCI API – SCICreateInterrupt()

- **SCICreateInterrupt()**
  - Creates an interrupt resource and makes it available for remote nodes
  - Initialize a handle for the interrupt
  - An interrupt is associated by the driver with a unique number (interruptId)
  - If the flag SCI_FLAG_FIXED_INTO is specified, the function use the number passed by the caller

SISCI API – SCIConnectInterrupt()

- **SCIConnectInterrupt()**
  - Connects the caller to an interrupt resource available on a remote node
  - The function creates and initializes a descriptor for the connected interrupt
  - Since the status of the remote interrupt is not known (i.e, not created) the SCIConnectInterrupt() must be called in a loop

```
do {
        SCIConnectInterrupt(...., &error);
        sleep(1);
while (error != SCI_ERR_OK) ;
```

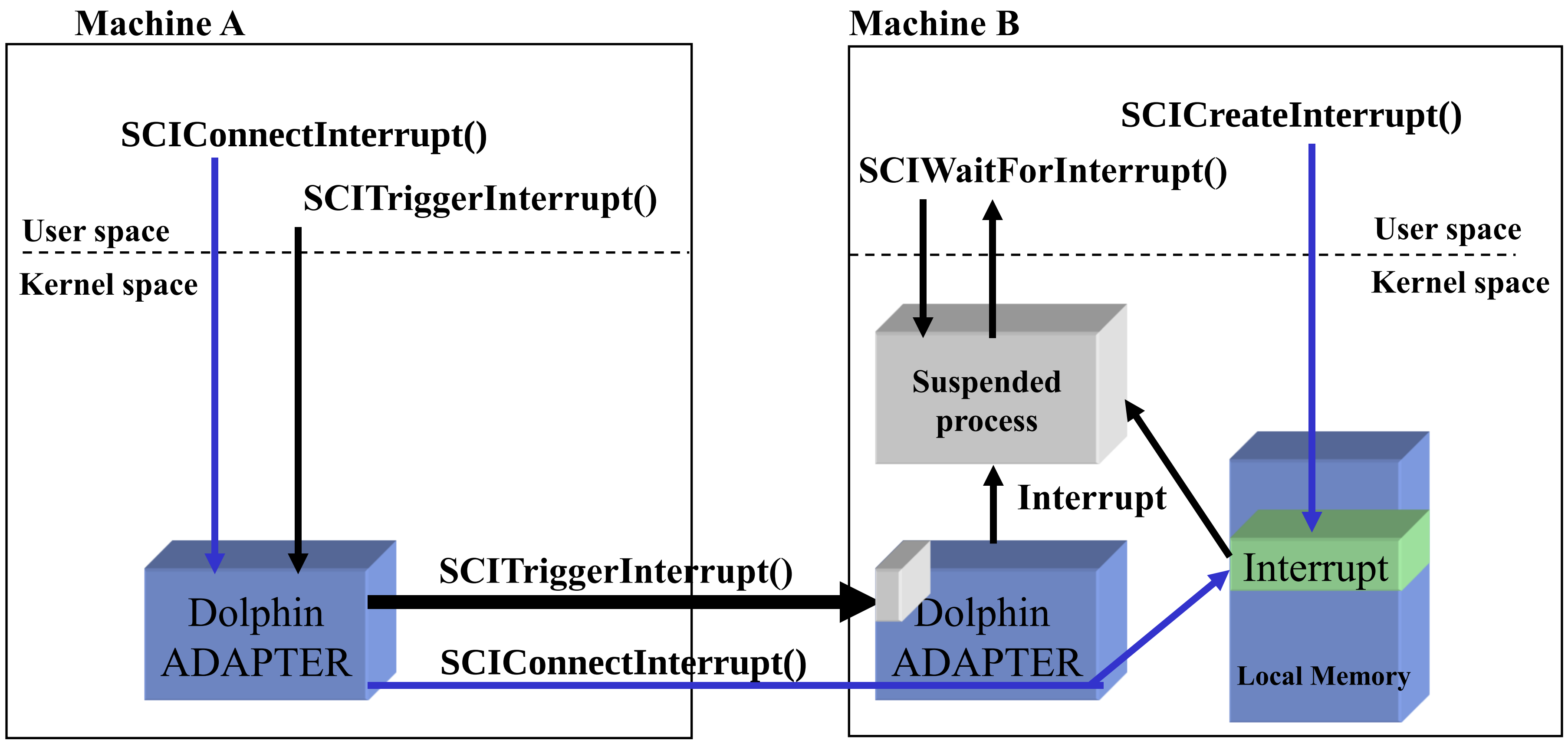
SISCI API – SCITriggerInterrupt()

- **SCITriggerInterrupt()**
  - The function triggers an interrupt on a remote node
  - The remote node gets notified

SISCI API – SCIWaitForInterrupt()

- **SCIWaitForInterrupt()**
  - This function blocks a program util an interrupt is received
  - If the flag option SCI_INIFINITE_TIMEOUT is specified, the function wait until the interrupt has completed
  - If a timeout value is specified, the function will gives up when the timeout expires.

SISCI API – INTERRUPT MODEL



SISCI API – SISCI API

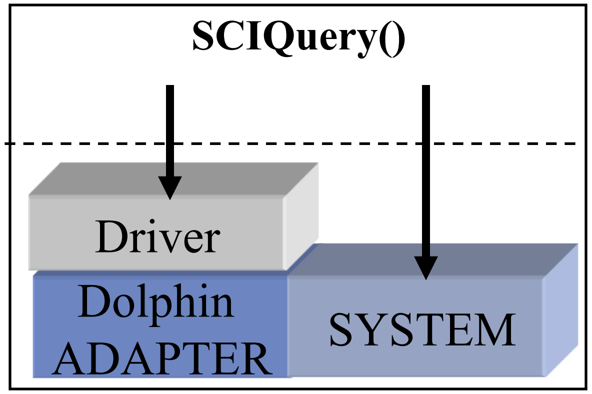
SCI QUERY

SISCI API – SCIProbeNode()

- **SCIProbeNode()**
  - The function check if the a remote node on the SCI network is reachable
  - The function is useful to check if all nodes on the cluster is initialized and reachable
  - Possible error codes
    - SCI_ERR_NO_LINK_ACCESS
    - SCI_ERR_NO_REMOTE_LINK_ACCESS

SISCI API – SCIQuery()

- **SCIQuery**
  - Provides information about the underlying Dolphin Express system
  - Each main group of requests defines its own data structure to be used as input and output to SCIQuery
  - The queries consist of the main COMMAND and a subcommand
    - SCI_Q_ADAPTER
      - SCI_Q_ADAPTER_SERIAL_NUMBER
      - SCI_Q_ADAPTER_NODEID
    - SCI_Q_SYSTEM
      - SCI_Q_SYSTEM_HOSTBRIDGE
  - Definitions of the queries are defined in DIS/src/api/sisci_api.h file



SISCI API - SCIQuery() Example

```
sci_error_t GetLocalNodeId(unsigned int localAdapterNo,
                           unsigned int *localNodeId)
{
   sci_query_adapter_t queryAdapter;
   sci_error_t  error;
   unsigned int nodeId;

   queryAdapter.subcommand = SCI_Q_ADAPTER_NODEID;
   queryAdapter.localAdapterNo = localAdapterNo;
   queryAdapter.data = &nodeId;

   SCIQuery(SCI_Q_ADAPTER,&queryAdapter,NO_FLAGS,&error);

   *localNodeId = nodeId;
   return error;
}
```

SISCI API

PRIVILEGED OPERATION

SISCI API – Privileged operation

- **SCIGetCSRRegister()**
  - This function reads a value from the specified CSR register on the Dolphin adapter
- **SCISetCSRRegister()**
  - This function writes a value to the specified CSR register on the Dolphin adapter
- **SCIConnectSCISpace()**
  - This function connects directly to any valid cluster address without any restriction
  - The responsibility of the segment handling is left to the programmer and the correctness of the used addresses

SISCI API - SCIConnectSCISpace

- **Allows the user to create a remote segment mapping any memory in the cluster**
- **Must be used with care as the same way as all the physical SISCI functions**
- **Not available with DX**
- **The user must specify the 64 bit network address**
  - Physical remote nodeId (16 bit)
  - OffsetHi (16 bit)
  - OffsetLo (32 bit)
    - IO address on remote node (offset)

SISCI API

CALLBACKS

SISCI API – Callbacks

- **Callbacks is used as an alternative methode for the SCIWaitFor...() functions**
- **Callback wont block the application**
- **The SISCI API support segment, DMA and interrupt callbacks**
- **A callback function and a callback parameter must be specified**
- **The callback functions require additional compilation setting in the application**
  - -D_REENTRANT
- **SCI_FLAG_USE_CALLBACK must be specified in the appropriate function calls**

SISCI API – Callback implementation

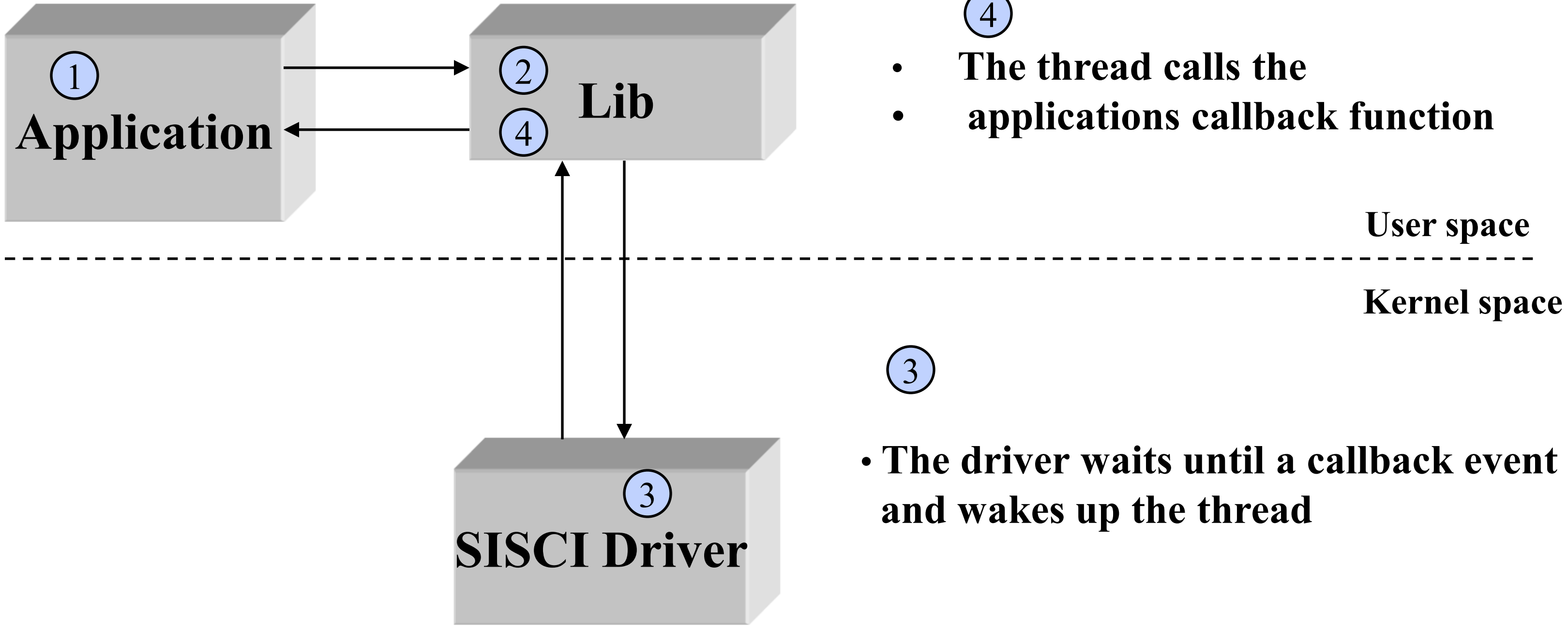
1 The application do a call to the function with CALLBACK specified.

2

- A thread is created in the library
- The library thread calls waitFor...()
- The application thread returns from the function and can continue the execution

4

- The thread calls the
- applications callback function



3

- The driver waits until a callback event occurs and wakes up the thread

SISCI API – DMA Callback

- **The DMA callback is trigged when the DMA has finished**
  - Either completed successfully or failed
- **Specify SCIPostDMAQueue(CALLBACK)**

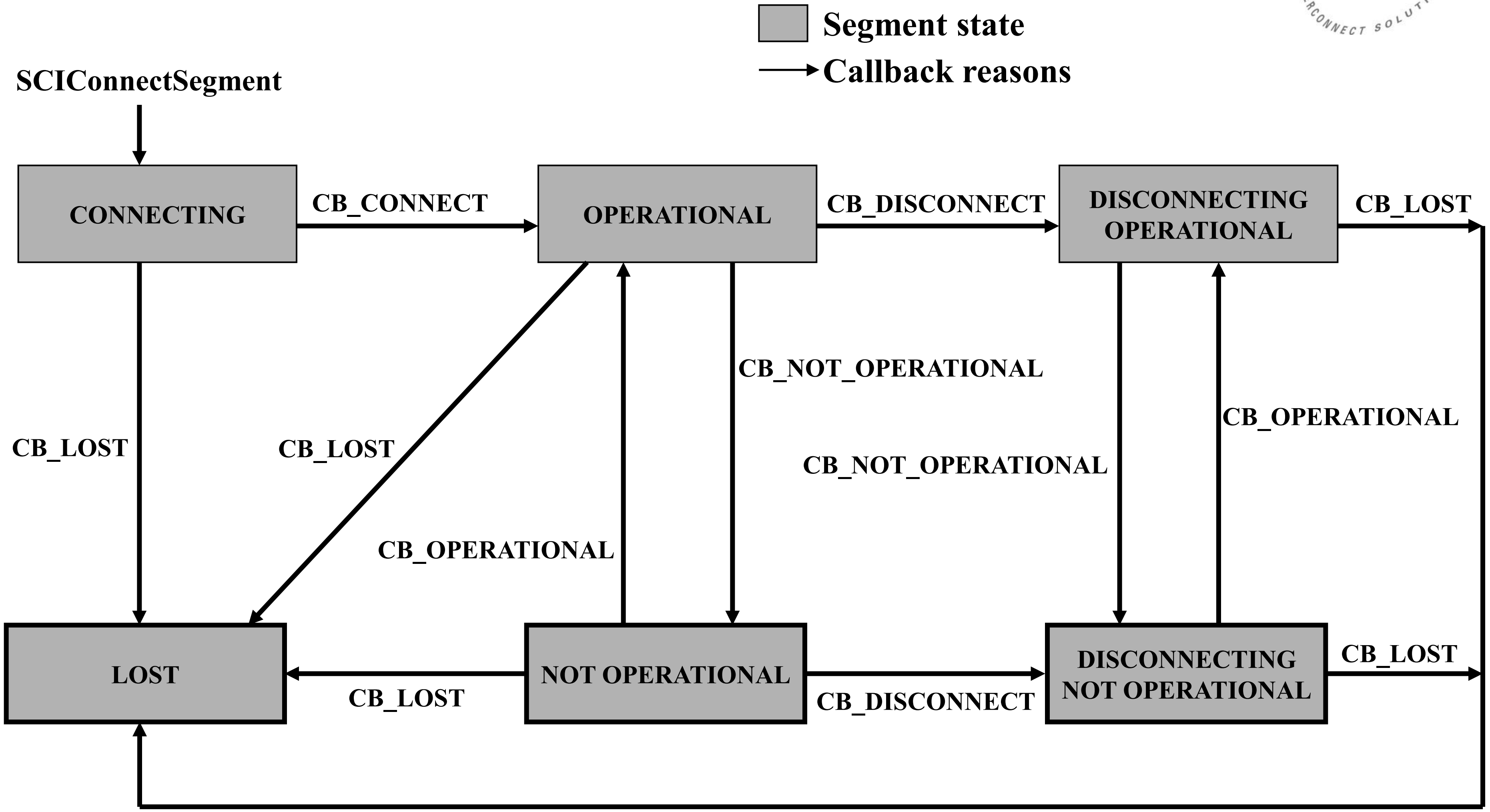
SISCI API – Local Segment Callback

- **A local segment callback event is issued every time the segment state is changed**
  - Someone connects or disconnects to the local segment
  - SCI link status change (operational/not operational)
  - A connection is lost from a remote node
- **Specify SCICreateSegment(CALLBACK)**
- **Callback reasons:**
  - SCI_CB_CONNECT
  - SCI_CB_DISCONNECT
  - SCI_CB_OPERATIONAL
  - SCI_CB_NOT_OPERATIONAL
  - SCI_CB_LOST

SISCI API – Remote Segment Callback

- **A remote segment callback event is issued every time the segment state is changed**
  - SCI link status change (operational/not operational)
  - The connection is lost to the remote node
- **Specify SCIConnectSegment(CALLBACK)**
- **If a CB_DISCONNECT callback is issued, it’s a request from the remote node to ”please” disconnect**
  - It’s a request not a demand

State diagram for remote segment callback



SISCI API – Interrupt callback

- An interrupt callback is issued every time an interupt from a remote node is seen on the interrupt handle
- Specify SCICreateInterrupt(CALLBACK)

SISCI API – Direct Transfer

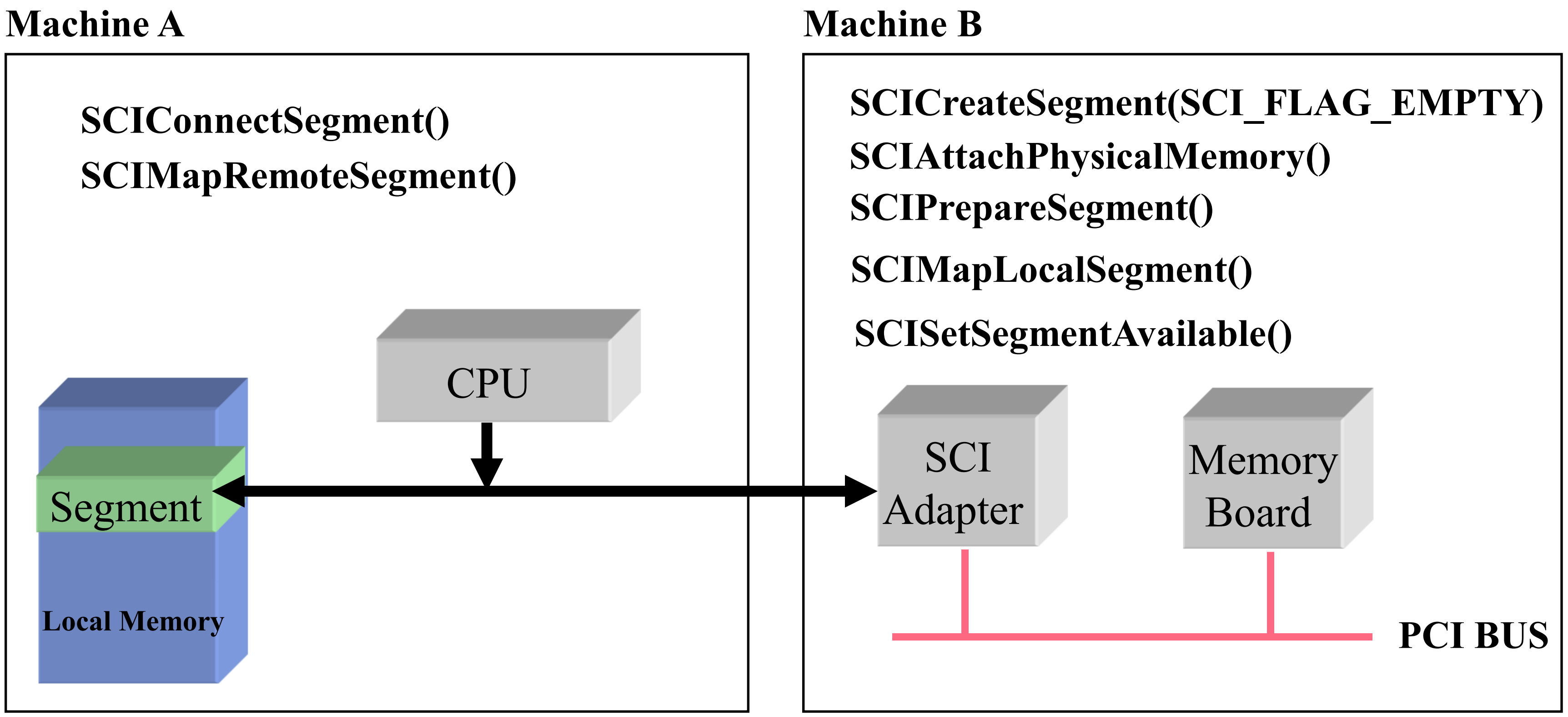
DIRECT TRANSFER

SISCI API - SCIAttachPhysicalMemory

- **Enable the possibility to set up a transfer into other devices than main memory**
- **In normal case, the transfer is between the local segment allocated in local memory and the remote node**
- **This function enables attachment of physical memory regions where the Physical PCI/PCIe bus address (and mapped CPU address) is already known**
- **The function will attach the physical memory to the SISCI segment which later can be connected and mapped as a regular SISCI segment**
- **The mechanism can can attach and transfer data directly to/from an extern PCI board through the Dolphin adapter**
  - Memory boards
  - Preallocated main memory
  - IO device

SISCI API - SCIAttachPhysicalMemory

- SCICreateSegment() with flag SCI_FLAG_EMPTY must have been called in advance

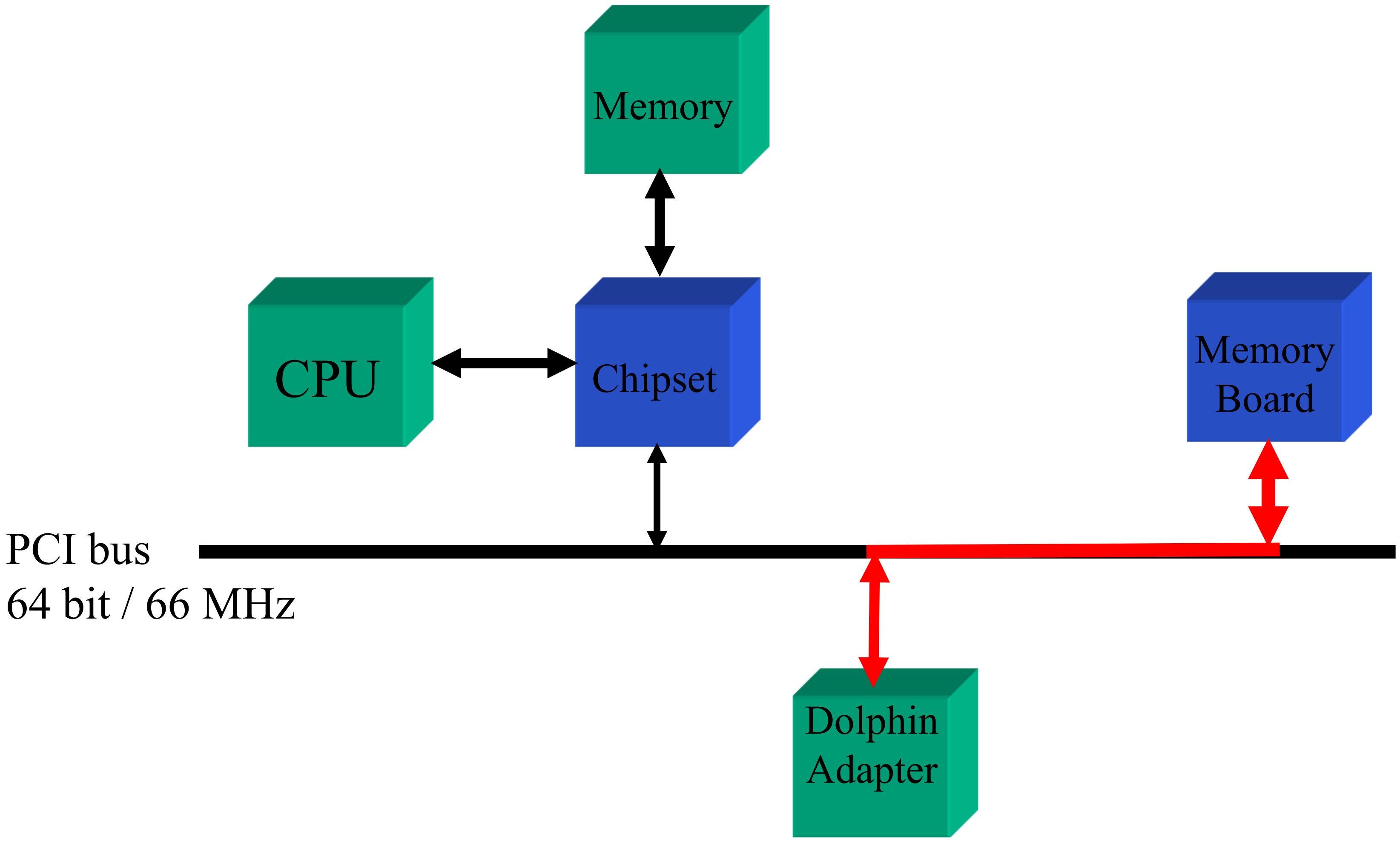


SISCI API - SCIAttachPhysicalMemory

```
SCIAttachPhysicalMemory(sci_ioaddr_t        ioaddress,
                        void               *address,
                        unsigned int        busNo,
                        unsigned int        size,
                        sci_local_segment_t segment,
                        unsigned int        flags,
                        sci_error_t        *error);
```

sci_ioaddr_t ioaddress :	**This is the address on the PCI bus that a PCI bus master has to use to write to the specified memory**
void * address:	**This is the (mapped) virtual address that the application has to use to access the device. This means that the device has to be mapped in advance bye the devices own driver.**
busNo:	**The bus number on the local PCI bus**

SISCI API – SCIAttachPhysicalMemory



PHYSICAL DMA

SISCI API - Physical DMA

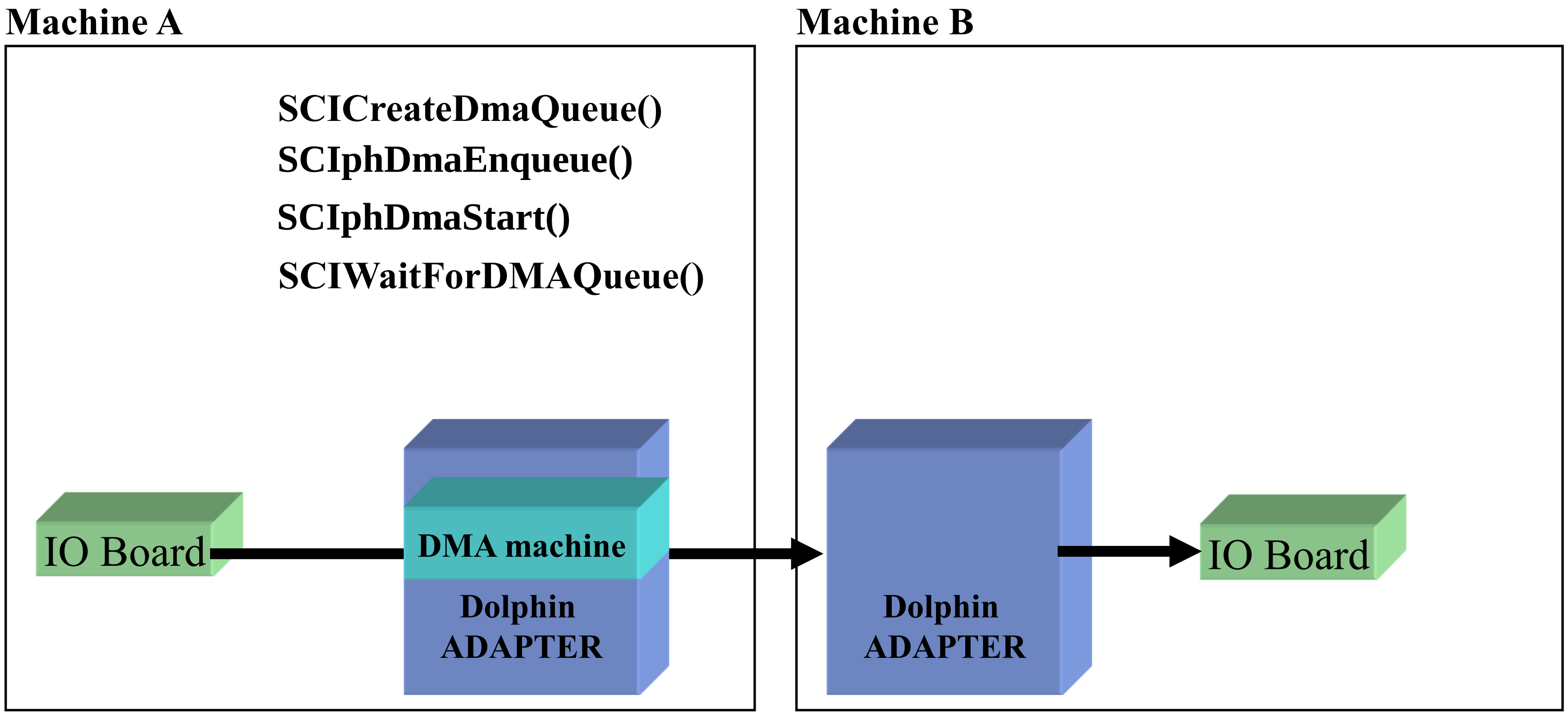
- The standard SISCI DMA functions require local segment and remote segment as input parameter.
- Physical DMA requires the local and remote ioaddr
- The physical DMA can be used to transfer directly data to/from an IO device on the PCI bus / PCI Express

SISCI API - SCIphDmaEnqueue

- **The function has the same functionality as SCIEnqueueDMATransfer()**
- **Instead of using the local and the remote segment, the function has the local and the remote ioaddress as the source and the target specification**
- **To transfer data direct to/from excplit memory on the system**
  - IO board
  - Memory board

SISCI API - SCIphDmaEnqueue

- DMA call sequence on client and server node


Machine A
Machine B
SCICreateDmaQueue()
SCIphDmaEnqueue()
SCIphDmaStart()
SCIWaitForDMAQueue()
IO Board
DMA machine
Dolphin ADAPTER
Dolphin ADAPTER
IO Board