CENAT

CONTROL ENGINEERING & AUTOMATION

LabVIEW Consultancy

Custom Circuit Design

Hardware in the Loop Test Systems

Automated Test Systems

Inter Thread Data Communication in LabVIEW

Presentation:
ir. Stijn Schacht

Table of Contents

- Inter thread data communication: overview
- Local & global variables: not done
- Notifications
- Queues
- Events
- Functional Global Variables
- Shared Variables
- Data DLL
- Data scope
- Data framework examples

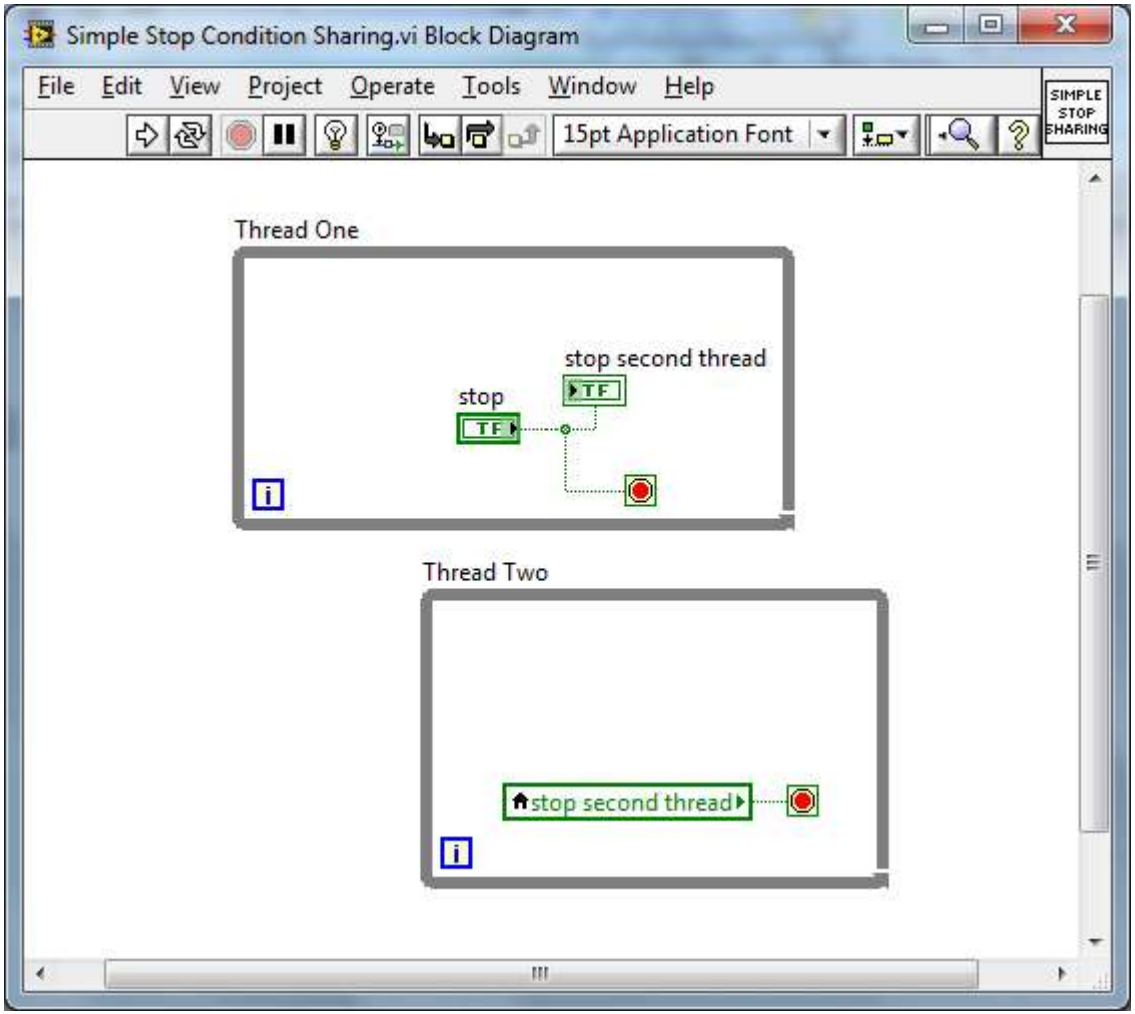
Inter Thread Data Sharing Techniques

- Local Variables
- Global Variables
- FGV’s
- Notifications
- Events
- Queues
- Shared Variables
- TCP/IP (loopback)
- DLL
- Datasocket
- Object Oriented (OO)
- File I/O
- Memory Mapping (RAM)
- FIFO’s
- Database SQL’s
- Custom/industrial Protocols
- .NET/ActiveX
- …

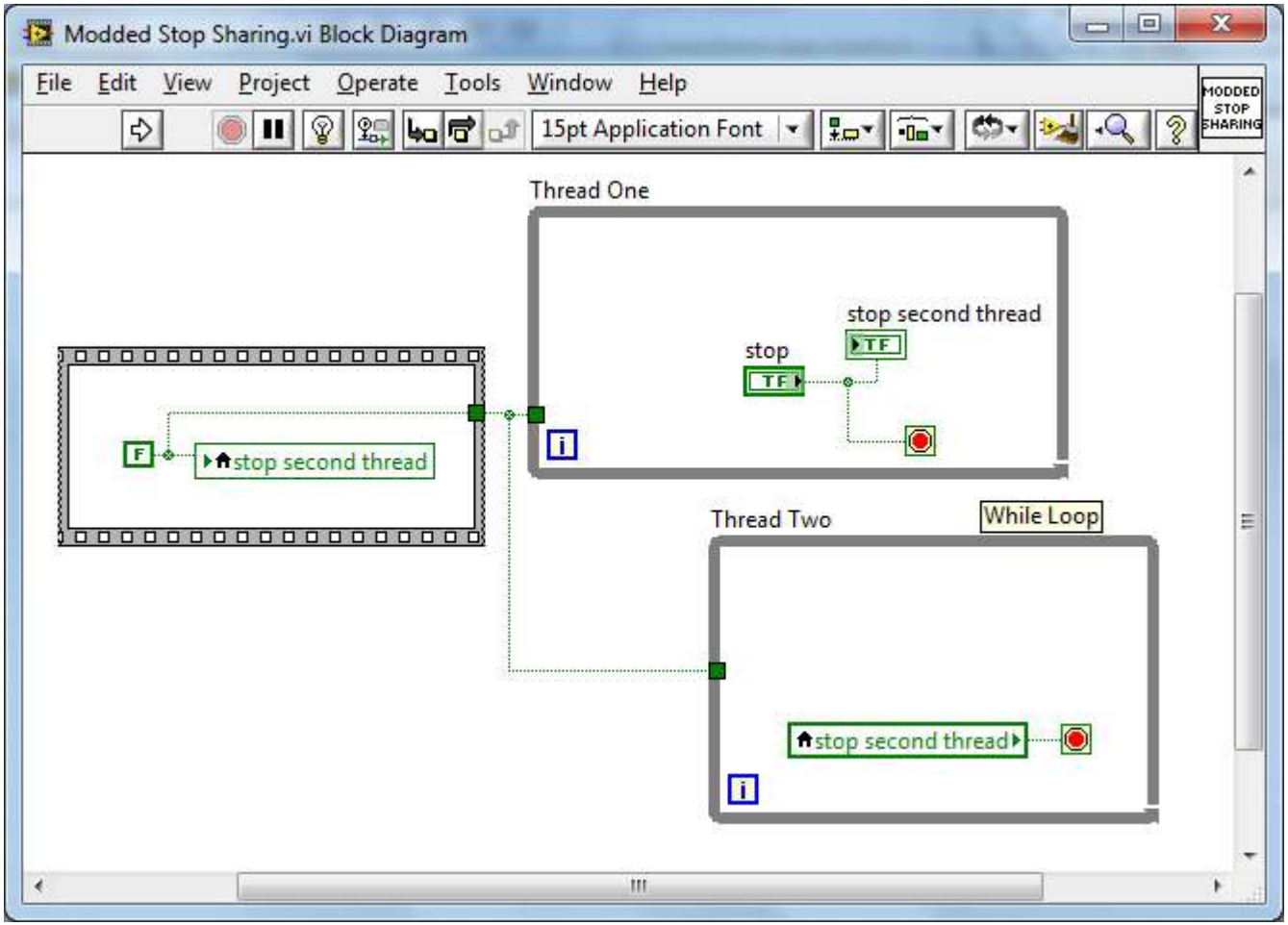
Inter Application Data Sharing Techniques

- Shared Variables
- TCP/IP (loopback)
- DLL
- Datasocket
- File I/O
- Memory Mapping (RAM)
- Database SQL's
- Custom/industrial Protocols
- .NET/ActiveX
- …

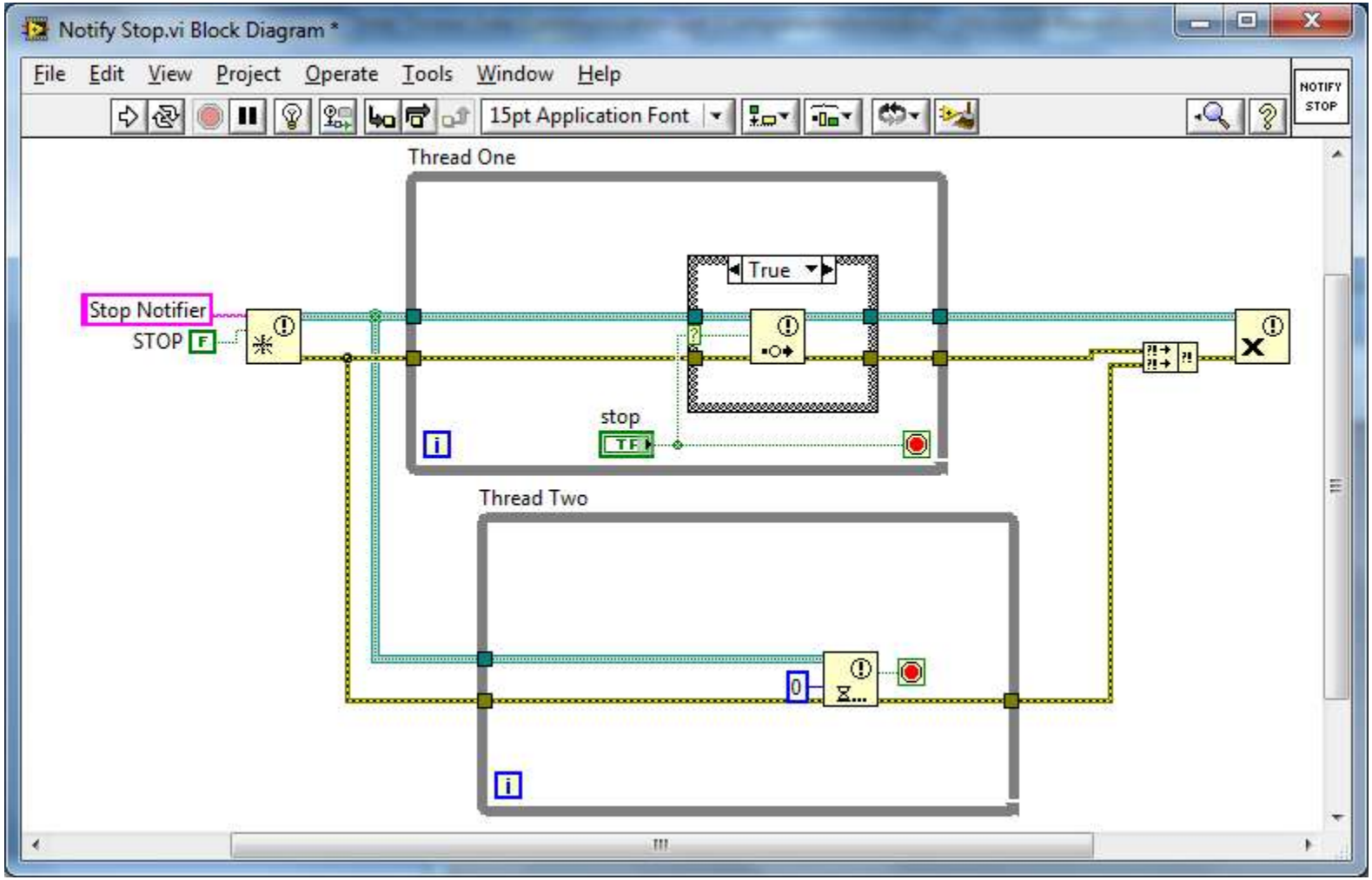
ITDS: Local and Global Variables



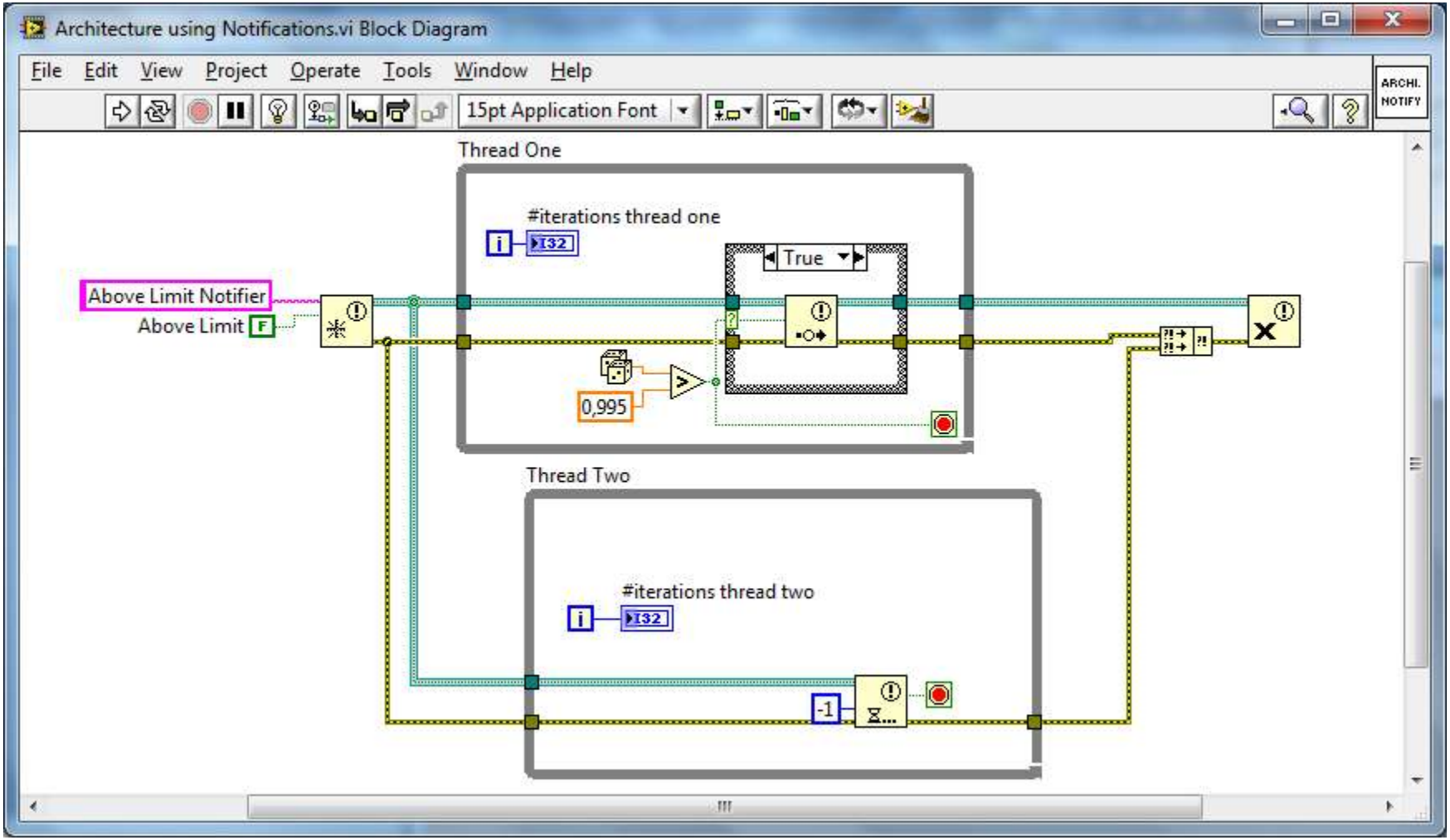
ITDS: Local and Global Variables



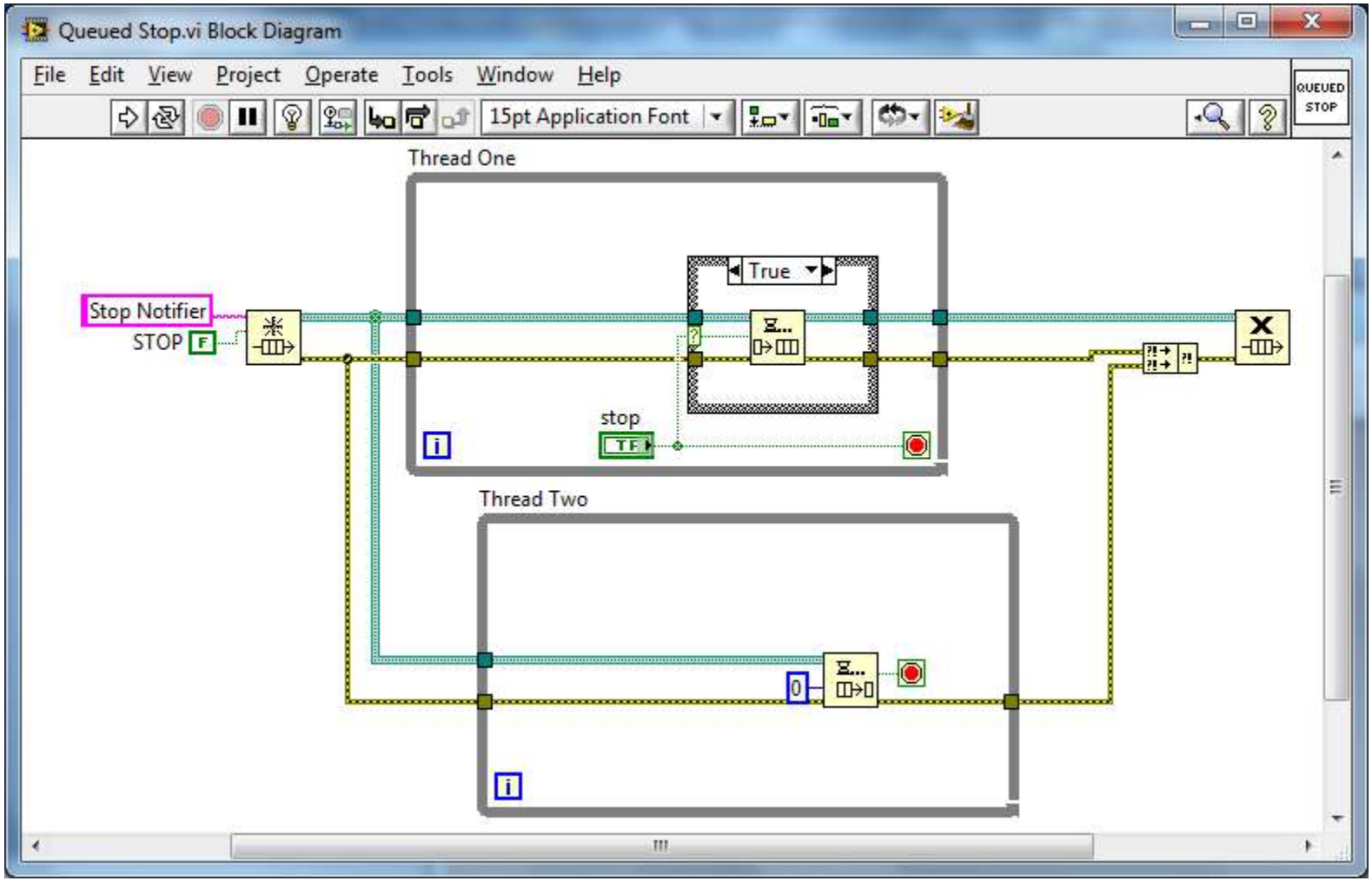
ITDS: Notifications



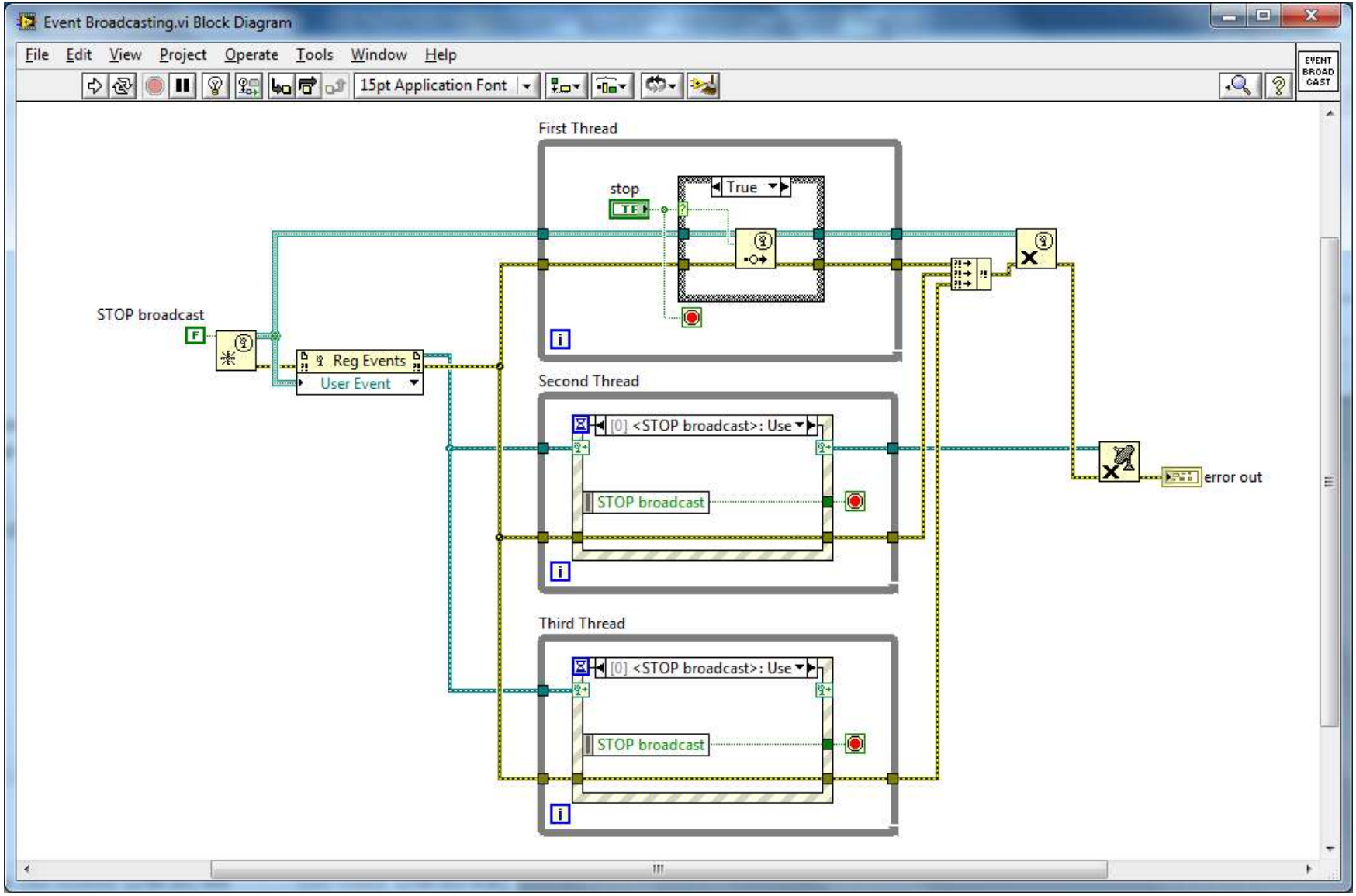
ITDS: Architectural example using Notifications



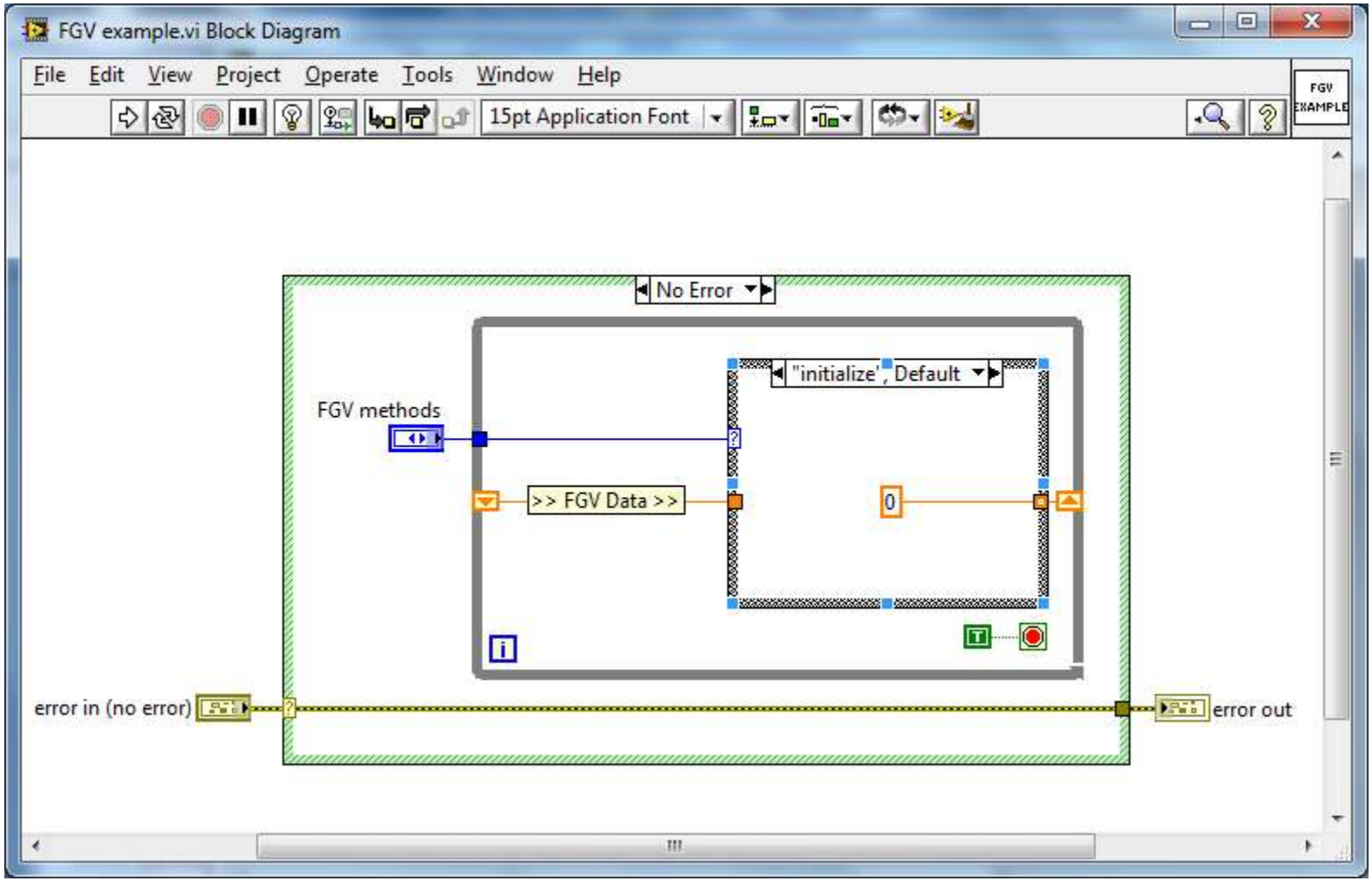
ITDS: Queues


Queued Stop.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
QUEUED STOP
Thread One
True
Stop Notifier
STOP
stop
TF
Thread Two
0

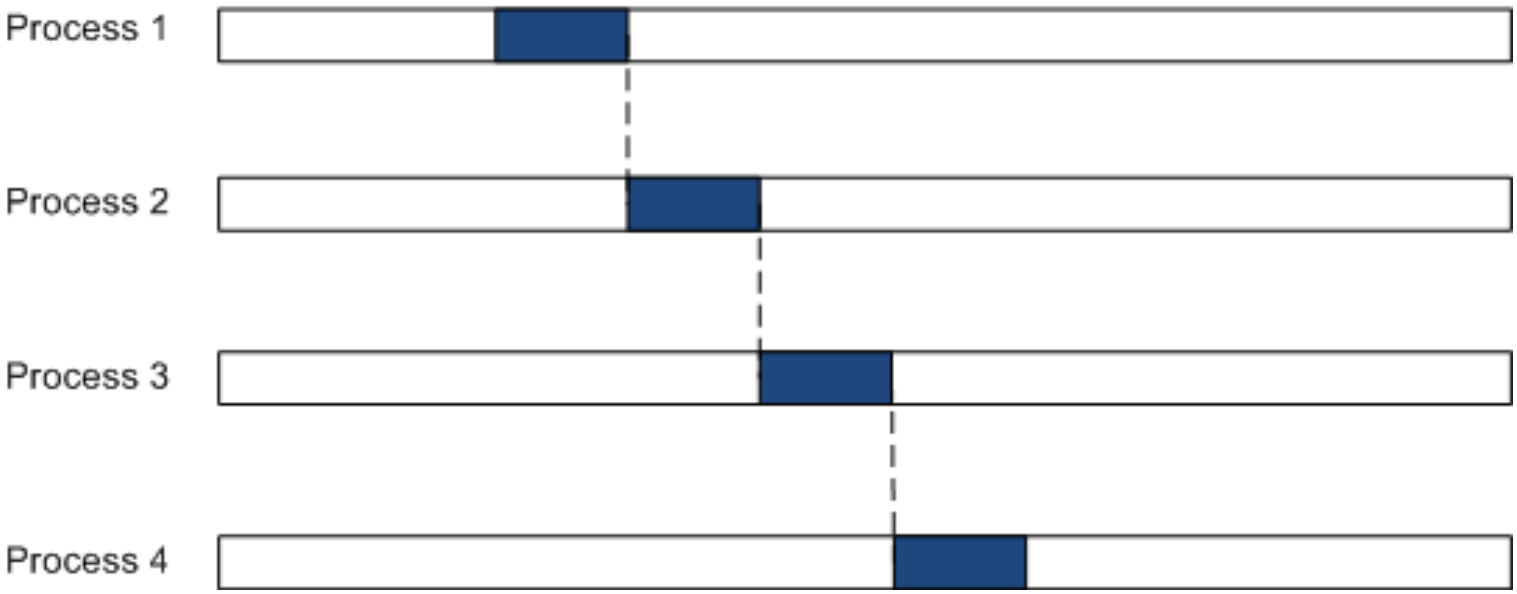
ITDS: Events


Event Broadcasting.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
EVENT BROAD CAST
First Thread
stop
TF
True
STOP broadcast
F
Reg Events
User Event
Second Thread
[0] <STOP broadcast>: Use
STOP broadcast
Third Thread
[0] <STOP broadcast>: Use
STOP broadcast
error out

ITDS: Functional Global Variables or FGVs



ITDS: FGVs a shared resource



The FGV shares the data among the different processes but each thread must wait on the other thread before it releases the resource.

ITDS: Can I make speed improvements for an FGV?

Reentrant FGV

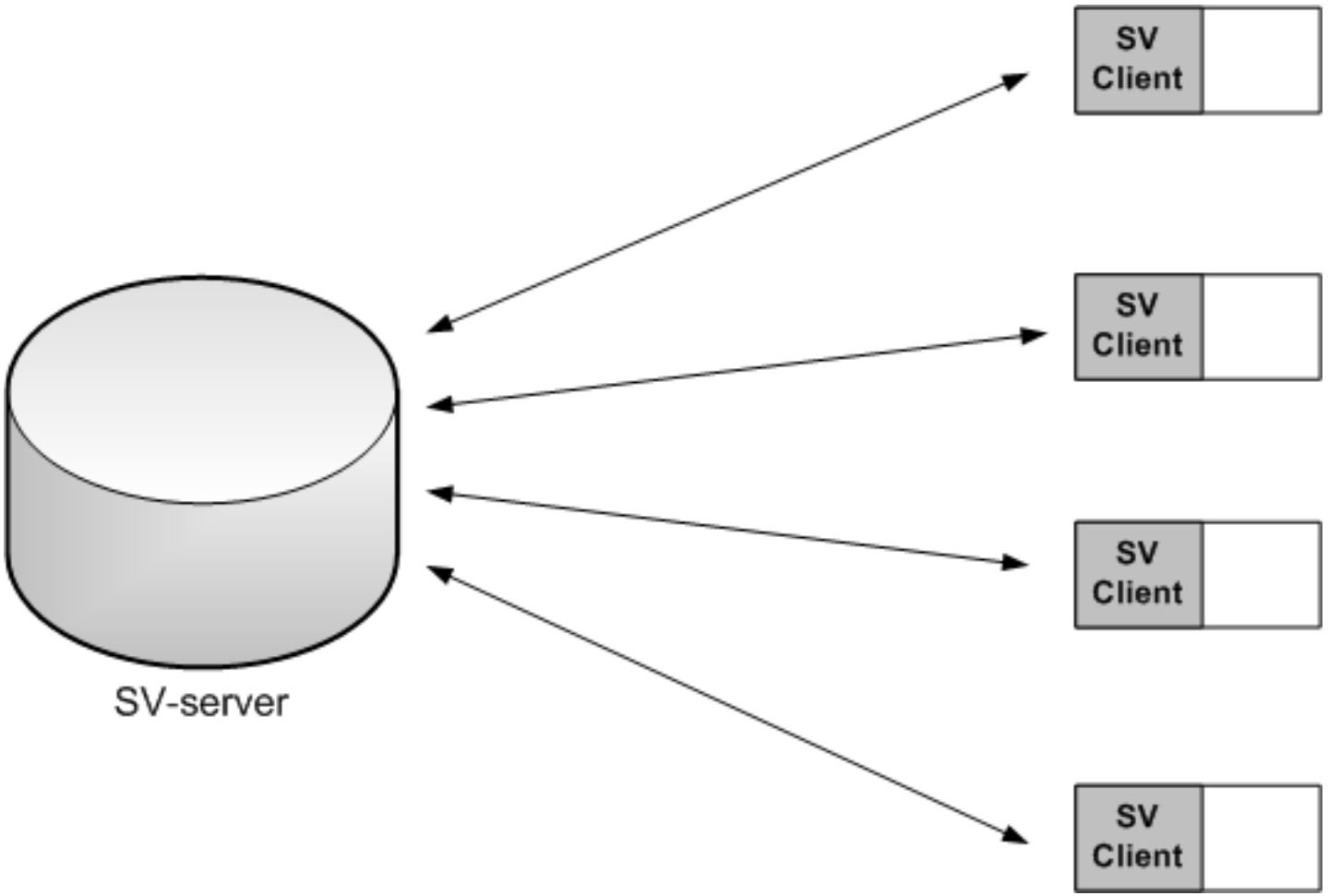
→ Making code reentrant will eliminate the shared resource problem
→ Each clone will hold its own data copy so the concept of data sharing is undermined…

Inline FGV

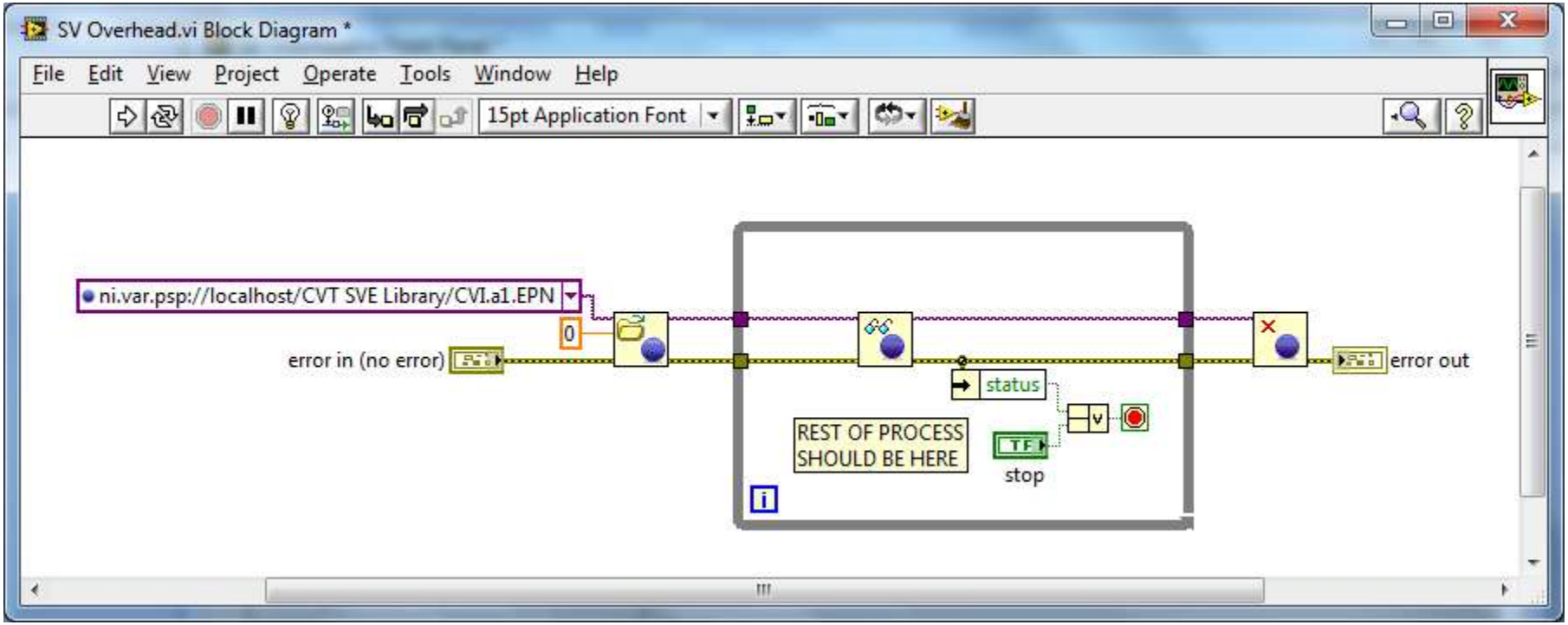
→ Inlining code will eliminate the code calling overhead
→ However, as with reentrancy the data sharing concept is undermined…

The only way to make speed improvements is making use of inplace data methods (if your FGV uses data methods at all).

ITDS: Shared Variabes



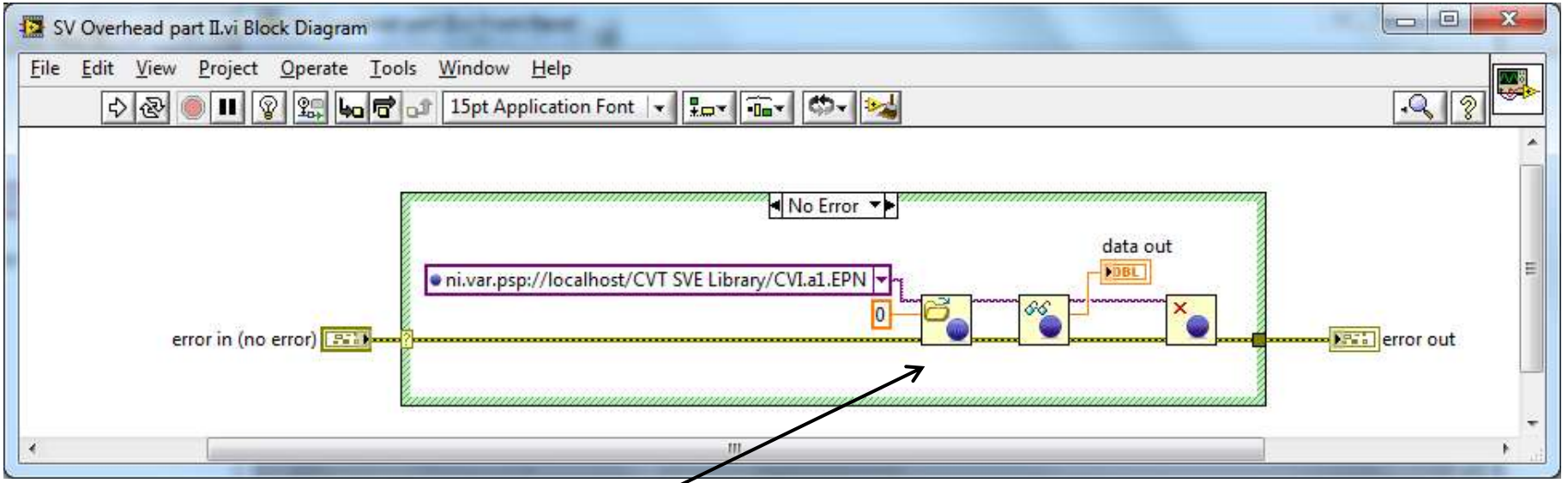
ITDS: Shared Variables, access overhead part I



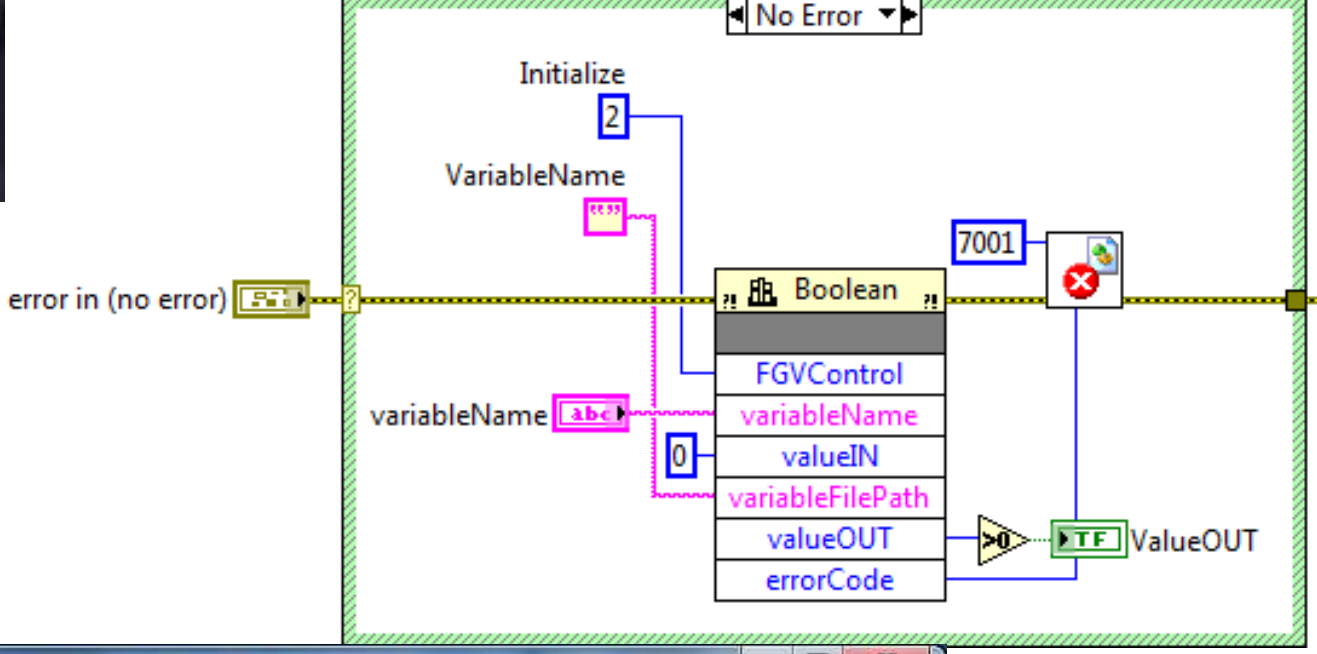
+ Shared Variable client thread
+ Shared Variable server thread
→ These run transparently for the user/programmer
→ Do not forget to deploy them when distributing your app

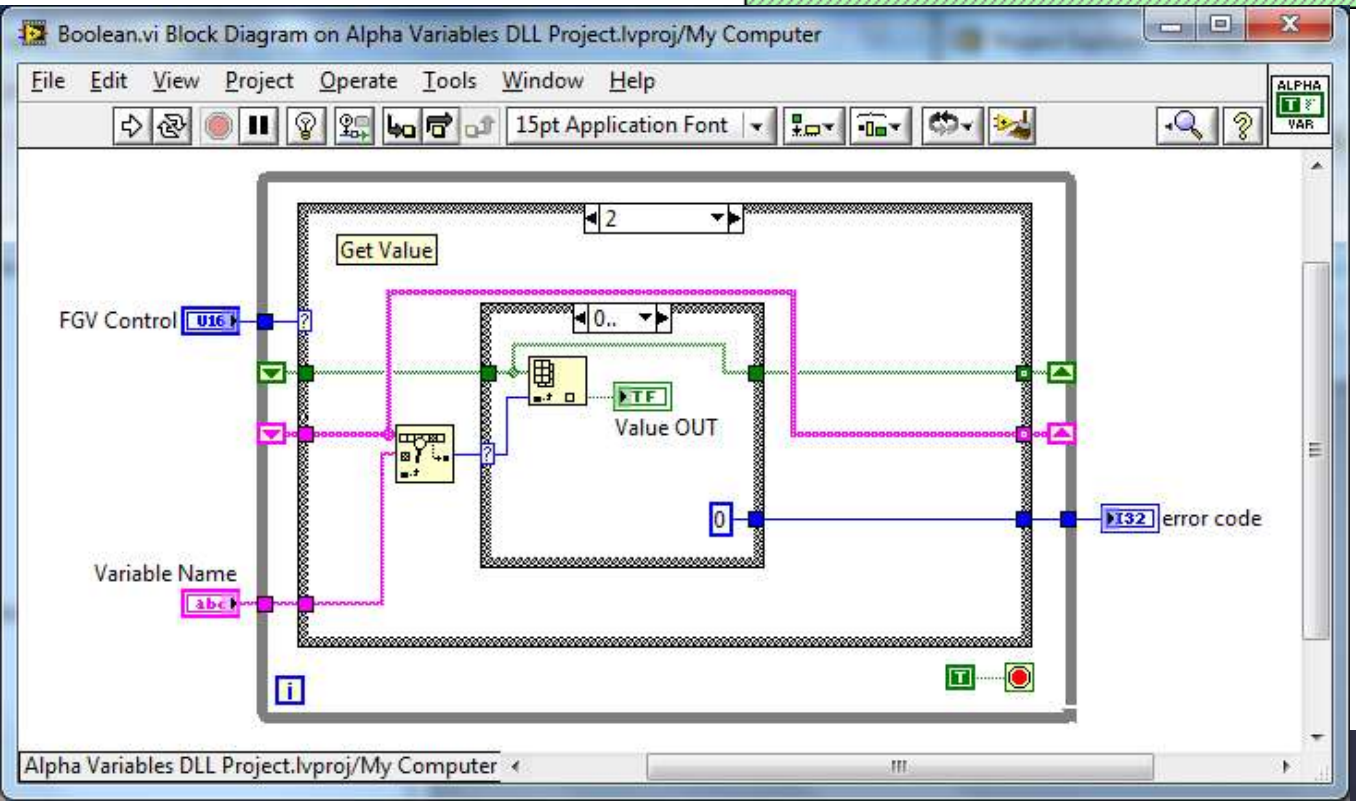
ITDS: Shared Variables, access overhead part II

Using an encapsulation subVI for shared variables
→ great overhead with each call(!)
→ sometimes there is no other option
→ if overhead is too big, use another data transfer mechanism



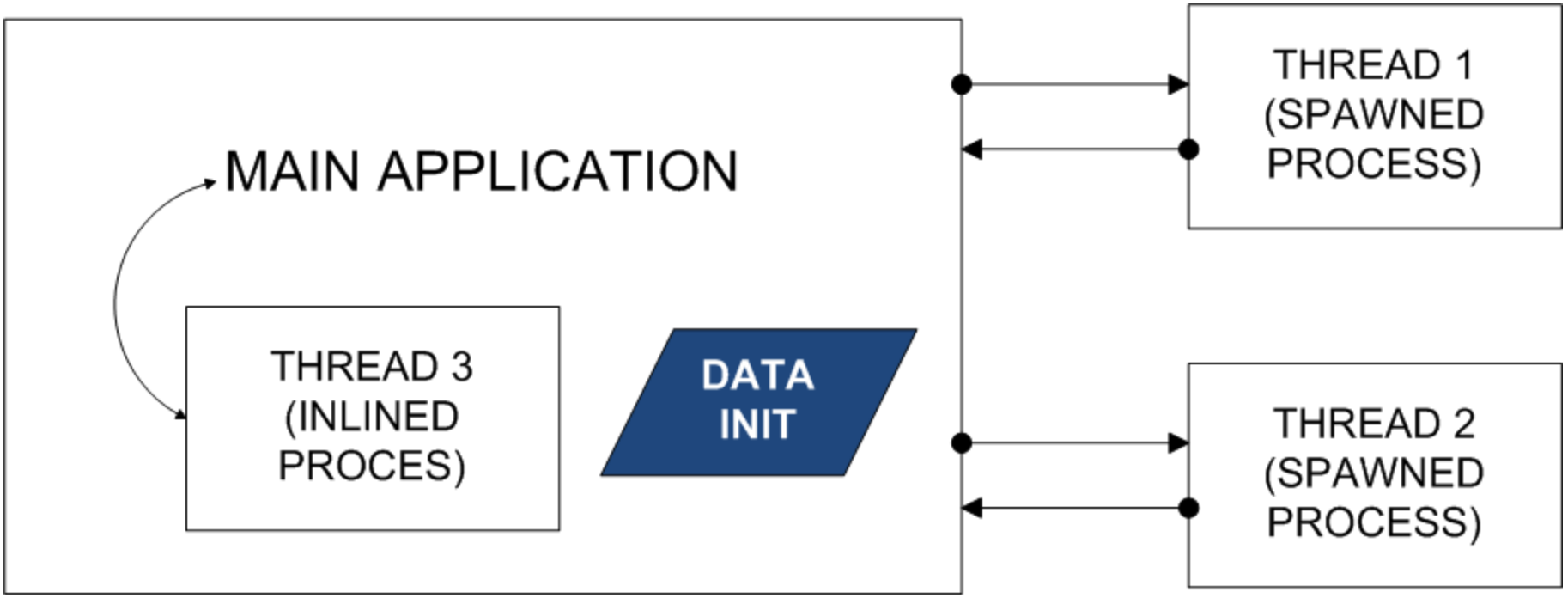
Initialization overhead with each call to the subVI!





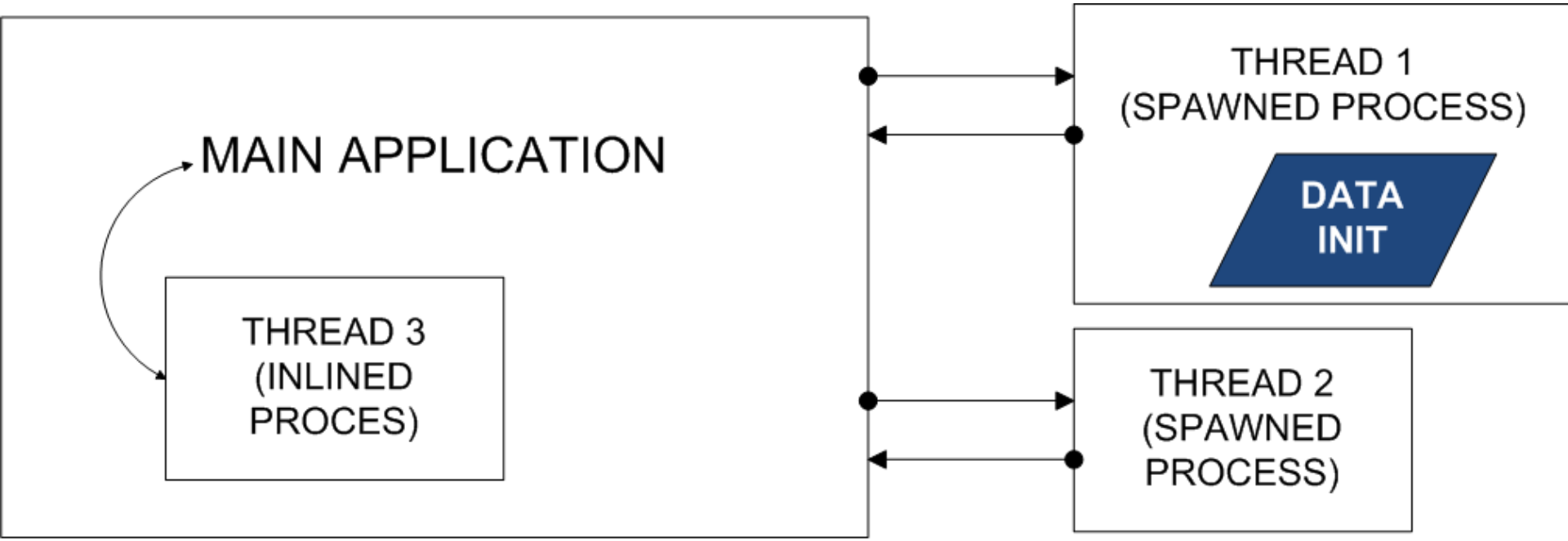
void Boolean(int16_t FGVControl, CStr variableName, uint8_t *valueIN, CStr variableFilePath, uint8_t *valueOUT, int32_t *errorCode);

ITDS: data scope



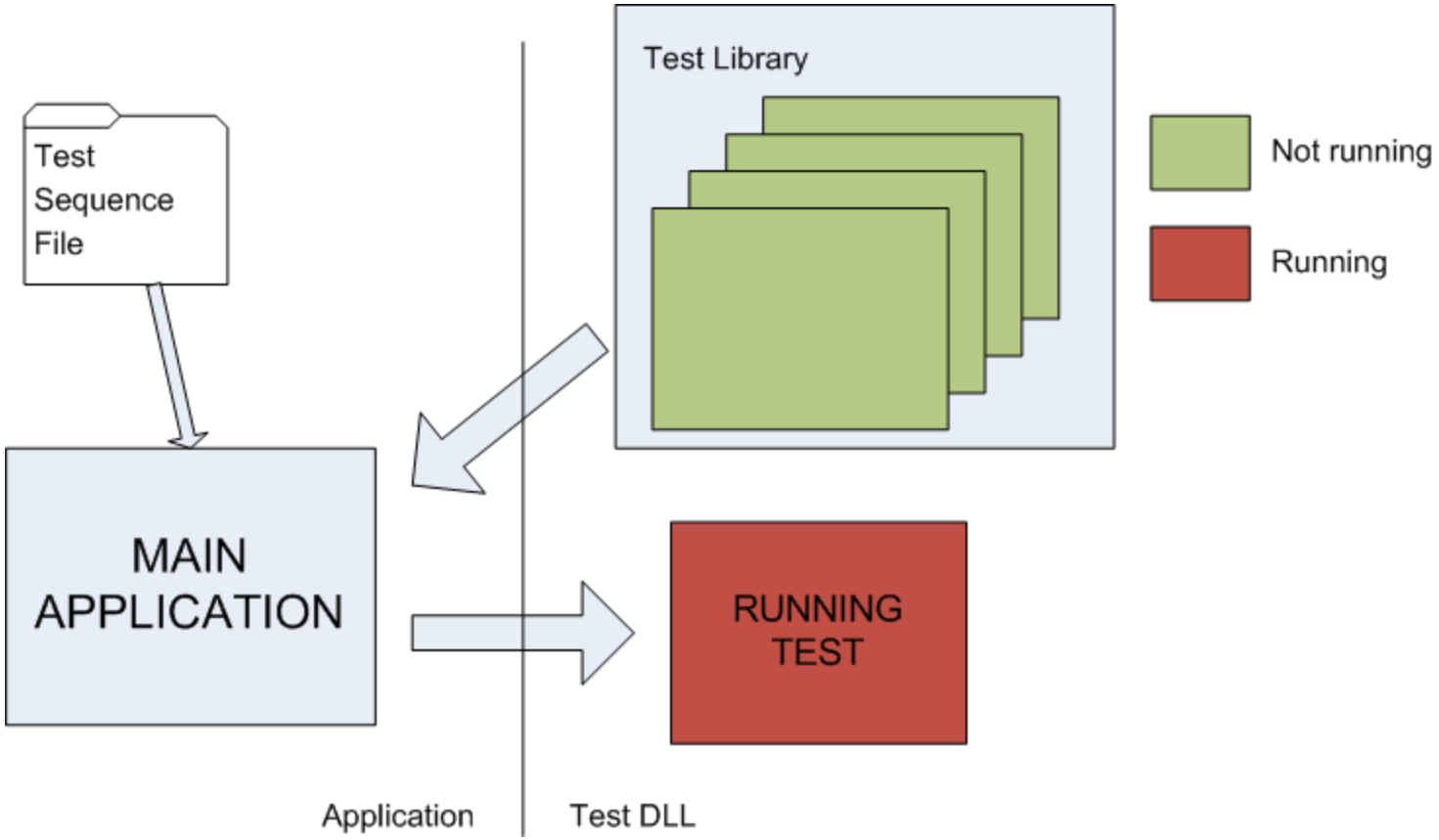
DATA INITIALIZED INSIDE THE ENTIRE APPLICATION SCOPE.
THEREFORE THE DATA IS WIDELY ACCESSIBLE

ITDS: data scope



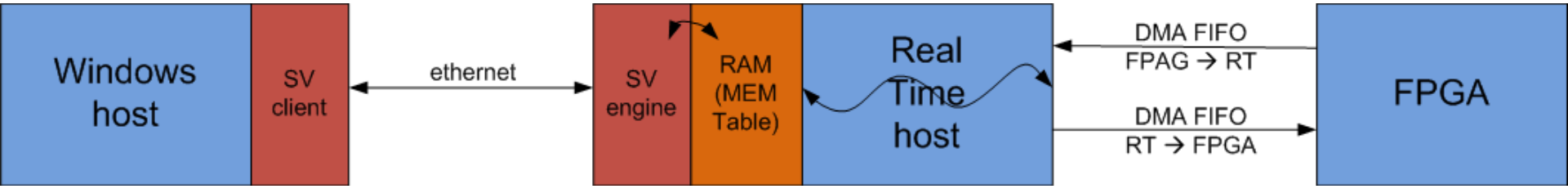
DATA INITIALIZED INSIDE THE MEMORY SCOPE OF A SPAWNED PROCESS. WHEN THREAD DIES, THE LABVIEW MEMORY MANAGER CLEANS UP THE DATA!!!

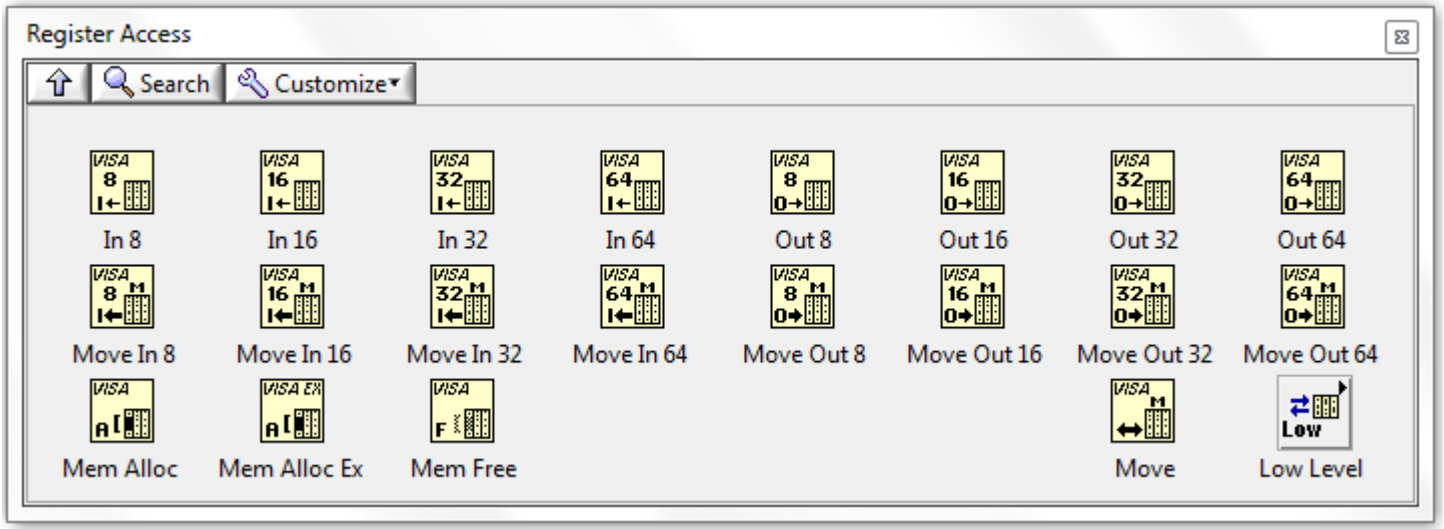
ITDS: data framework example 1



ITDS: data framework example 2

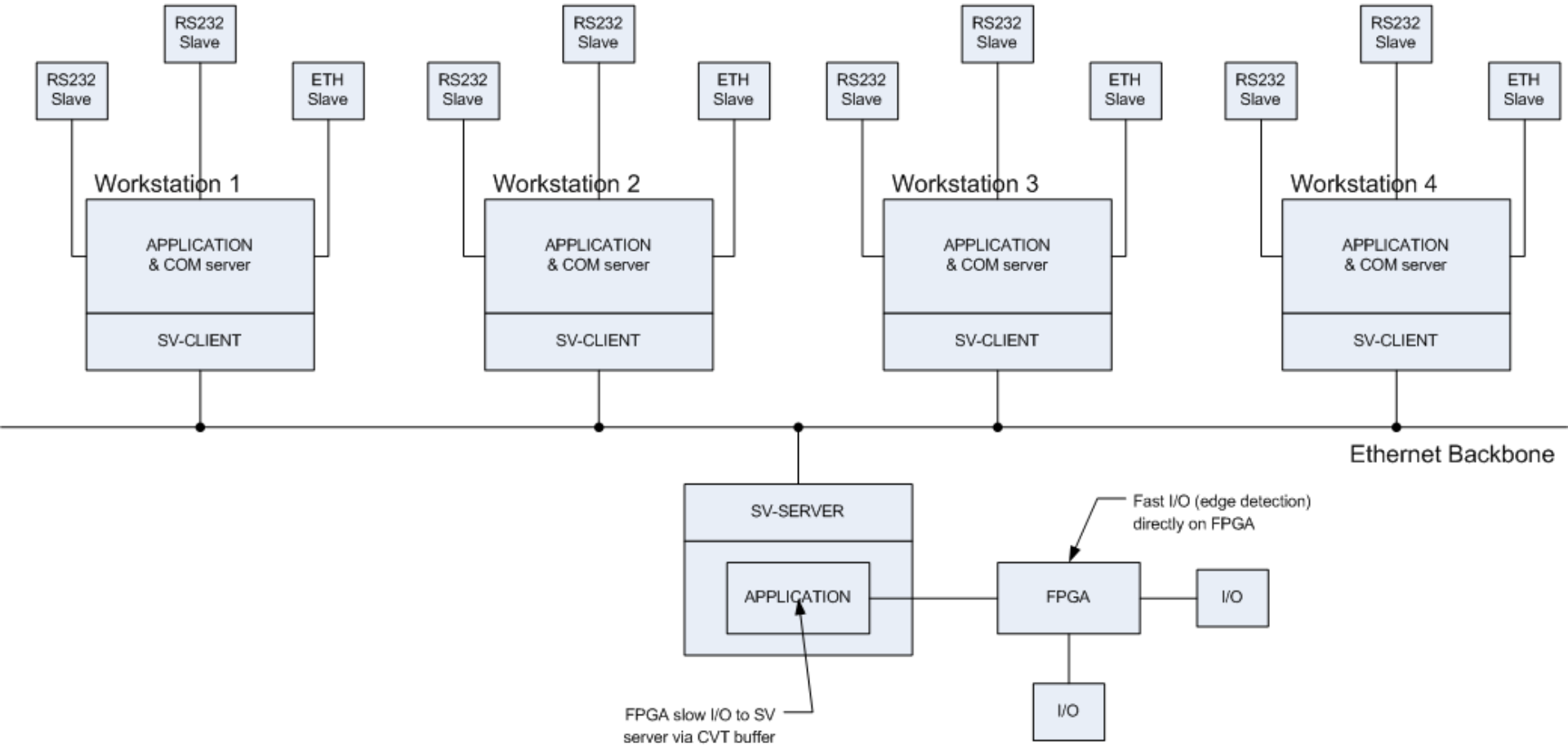
Memory mapping as central data access
→ Direct memory access



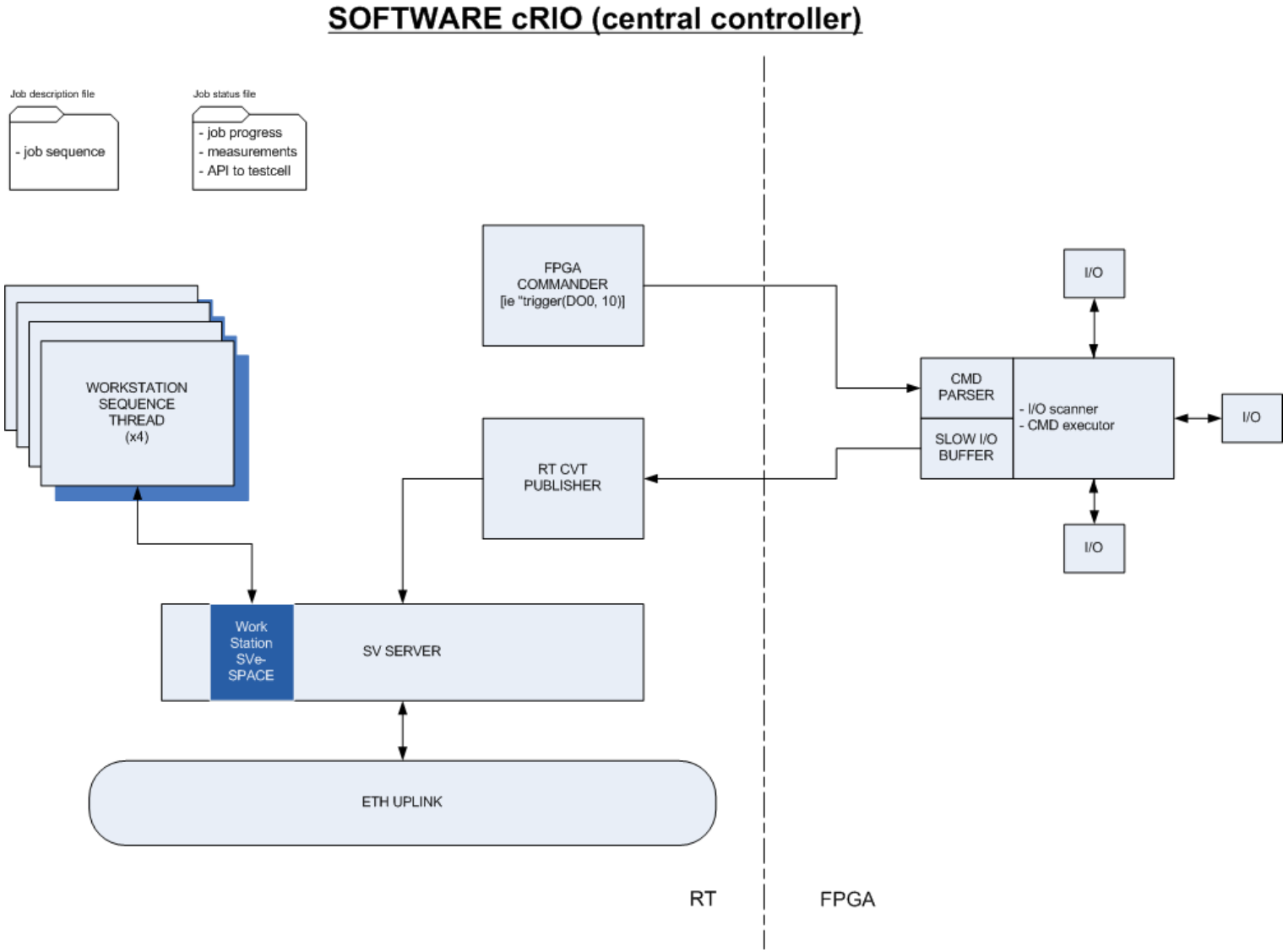


ITDS: data framework example 3

HARDWARE OVERVIEW



ITDS: data framework example 3



Syntheses part I

Local & Global Variables
→Just try not to use them

Notifications
→Use for single thread to single thread communication
→Lossy data transfer
→Can be used for time synchronization, or event occurrence awaking too

Queues
→Use for multi thread to single thread communication
→Lossless data transfer
→Can be used for time synchronization or event occurrence awaking too

Events
→Use for single thread to multi thread communication: broadcasting
→Lossless data transfer, event node acts as a Queue

Syntheses part I

FGV's
→Use for multi thread to multi thread communication
→Embed the "functions" to manipulate the data
→Shared resource, VI code loads at runtime

Shared Variables
→ Use for interthread, inter application or inter computer communication
→ Initialization overhead, do not use for very fast data transfer
→ SV client thread & server thread can have great overhead on embedded systems with low processing power
→ Can be used for application debugging via distributed system manager

Data DLL
→ Use with wrapper DLLs.
→ Typically the DLL itself is written around an FGV
→ remember that the Call Library Function Node is a blocking node, so all other threads are blocked. This is ideally suited for data transfer, since it should be mutexed…

Syntheses part I

Direct Memory Access or ‘File Mapping’
→ Good alternative to all other methods
→ Requires a great deal of code to write
→Not portable to inter computer communication