
GPower
Power

GPower

Steen Schmidt

Don’t make VIs **reentrant!**

LVOOP?

LabVIEW topics

Memory management

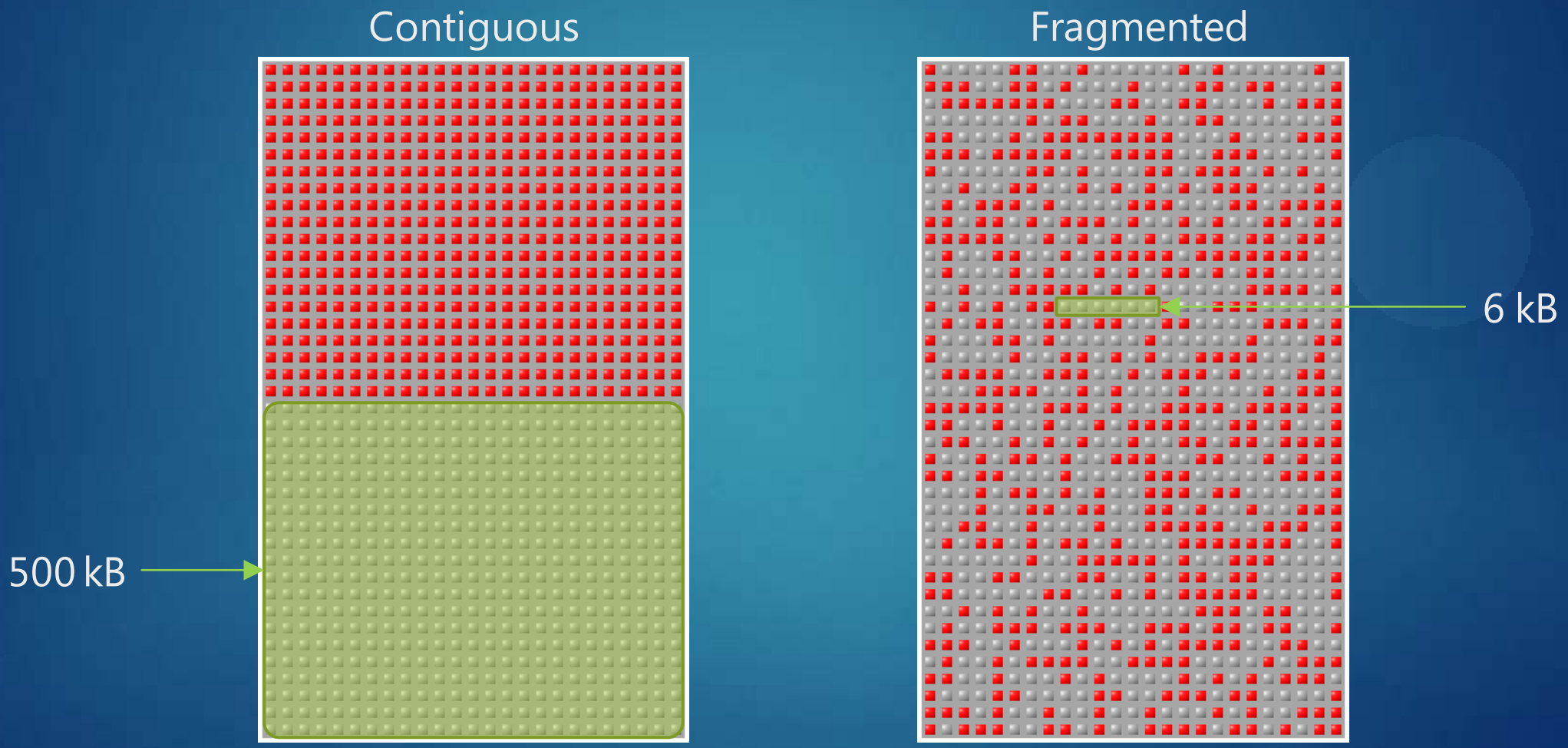
- Contiguous vs fragmented memory
- The memory sub-allocator
- Copy vs in-place operations
- Non-operations
- Don’t eliminate coercion dots
- Don’t worry about buffer allocations
- Clusters don’t exist
- How to deallocate queues

The Execution System

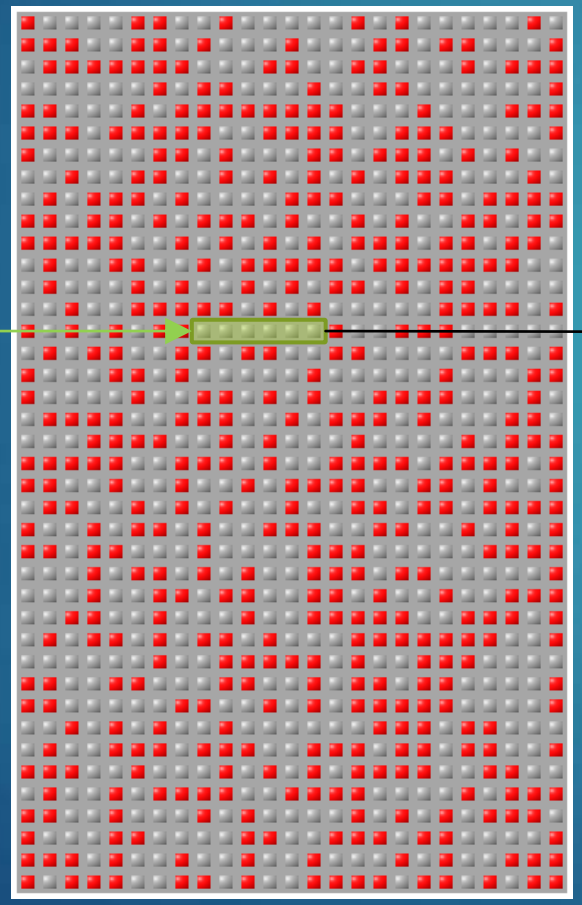
- Please set VI execution settings
- The transfer buffer
- High priority = low performance
- The root loop
- Don’t make VIs reentrant

Fragmented Memory

1000 kb memory, 50% utilized

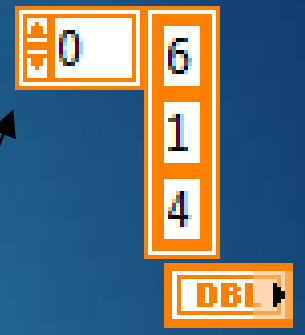


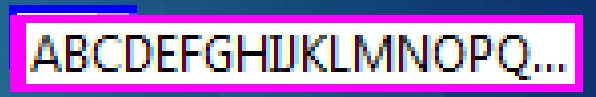
max 6 kB ?

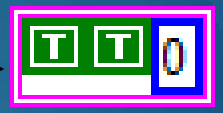


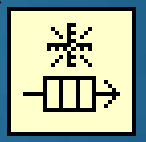
1x

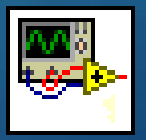
Memory Sub-Allocator

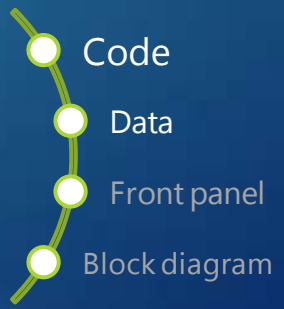




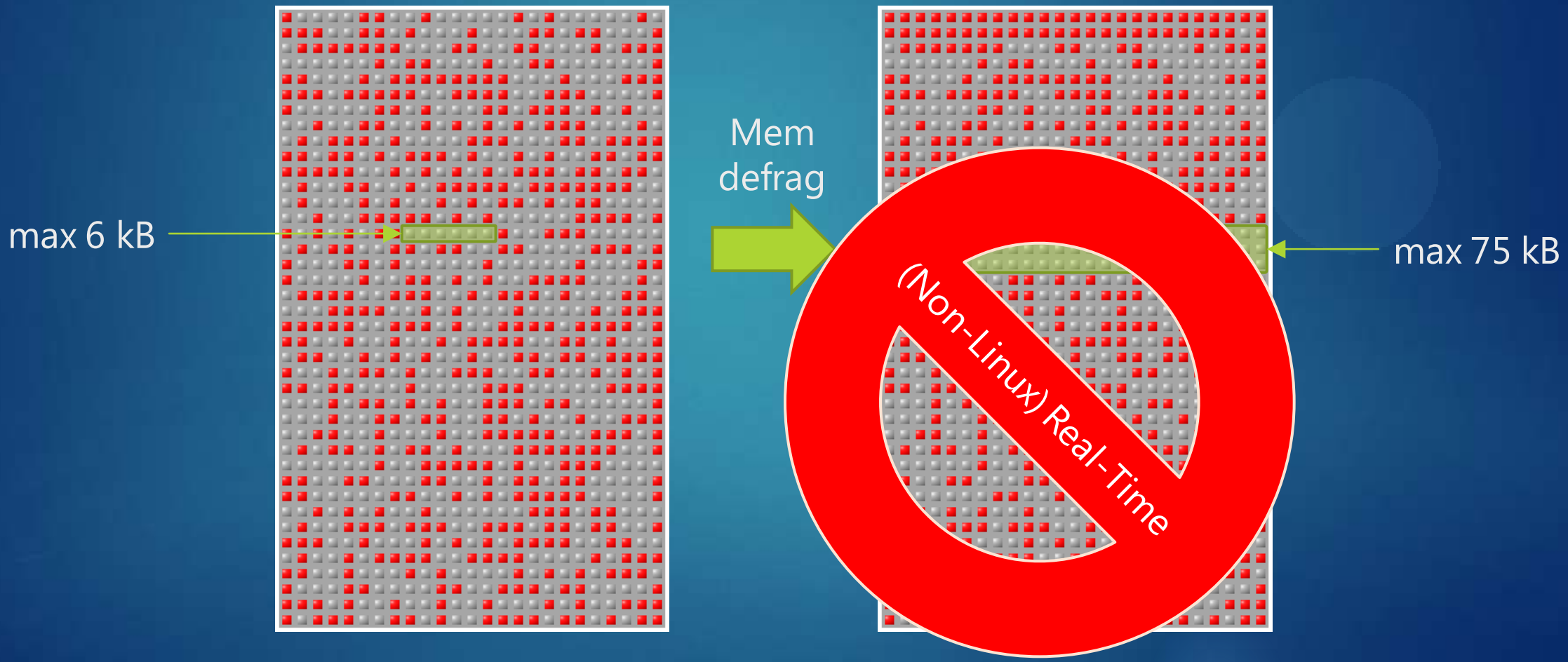








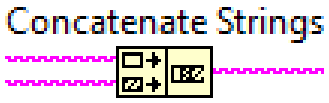
Out of **contiguous** memory?

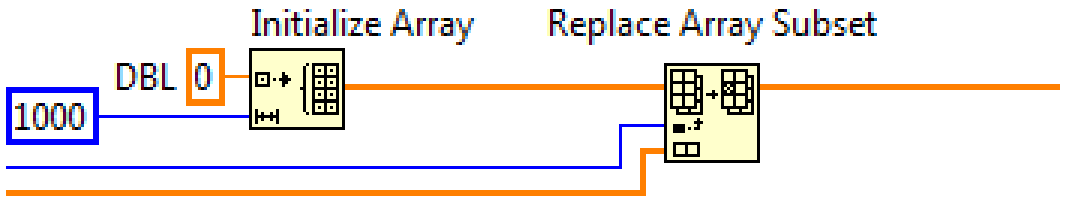

max 6 kB
Mem
defrag
max 75 kB
(Non-Linux) Real-Time

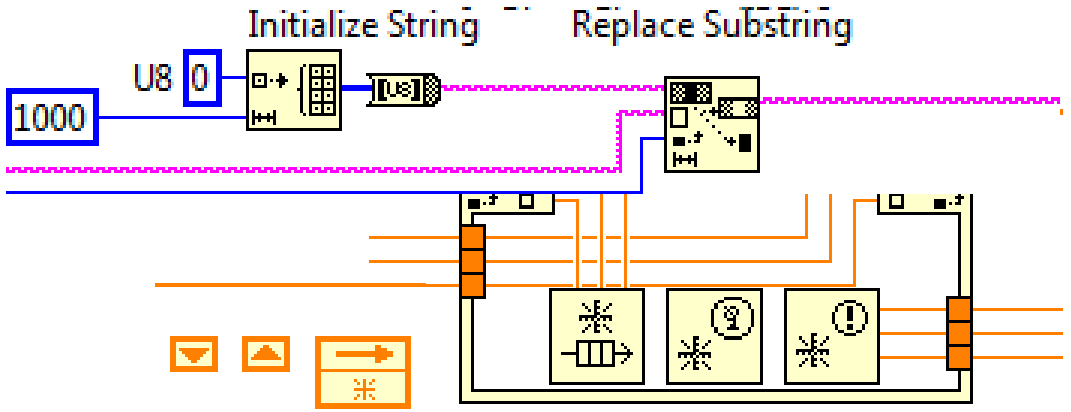
So, how to minimize **memory fragmentation**?

Minimize Dynamic Memory (De)allocation

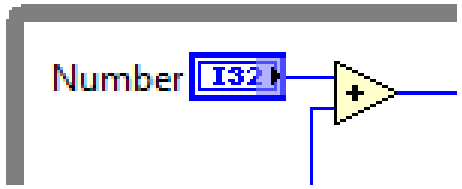
Building and copying

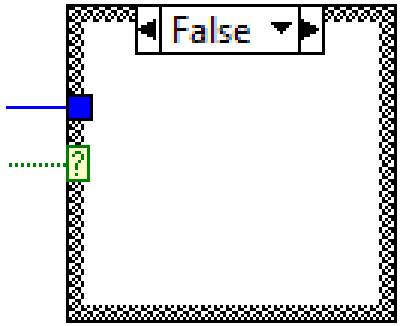


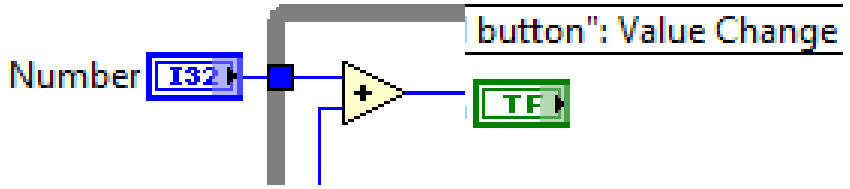


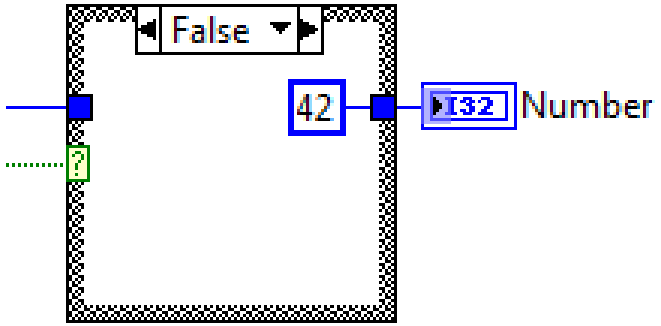


Terminals

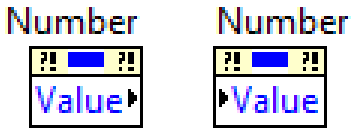

Number
I32

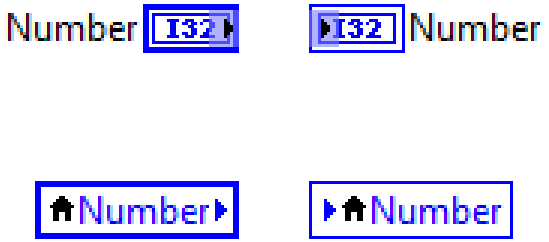

False
?


Number
I32
button": Value Change
TF


False
42
I32
Number
?

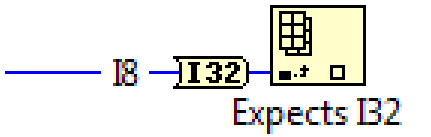
Property nodes

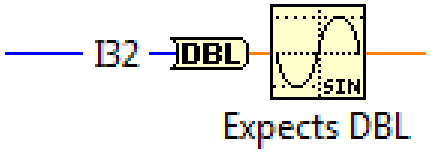


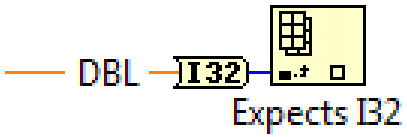


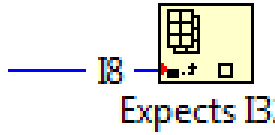
Anything really…

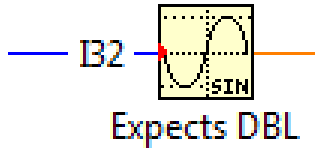
Coercion dots

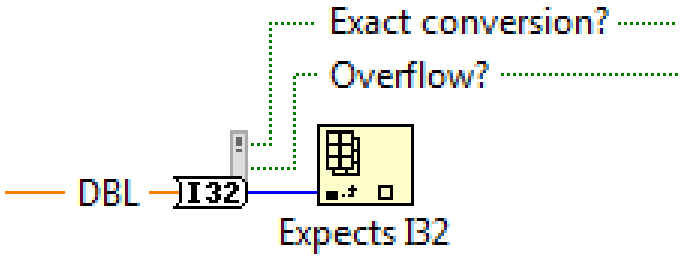




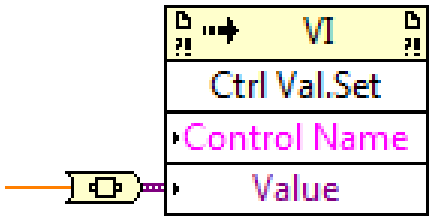


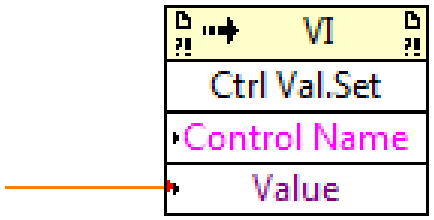




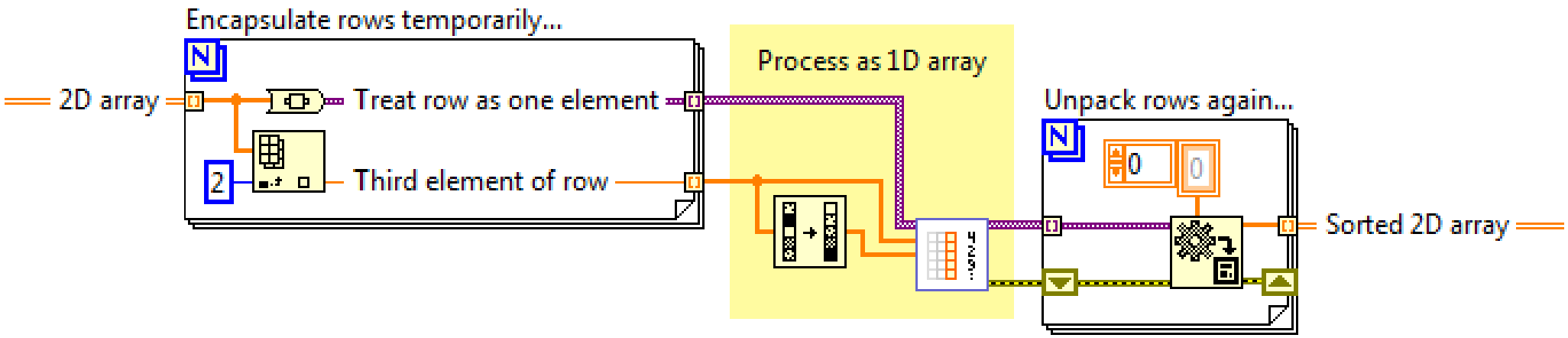


To Variant


VI
Ctrl Val.Set
Control Name
Value

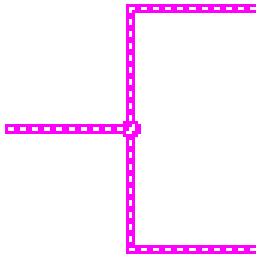

VI
Ctrl Val.Set
Control Name
Value

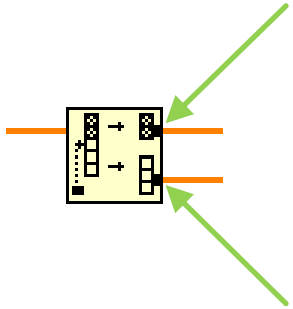
When it's **OK** to use To Variant



A few things
not
to worry about...

No (certain) data copies

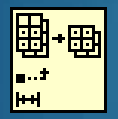




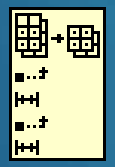
Array slices

Array

1	2	3	4	5	6	7	8	9



1	2	3
4	5	6
7	8	9



In memory

Stride = 1 Stride = 1

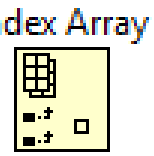
Dim 9	1	Dim 4	3	4	5	6	7	8	9

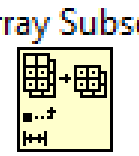
Stride = 1 Stride = 3

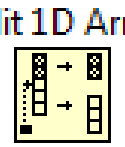
Dim 3	Dim 3	Dim 3	2	3	4	5	6	7	8	9

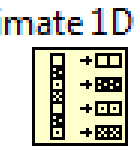
No-ops (and their neighbors)

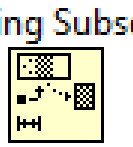
Possible array slices:



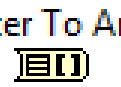


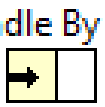


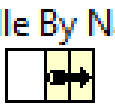




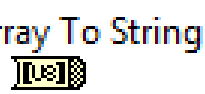
No-ops:

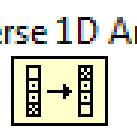


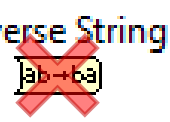




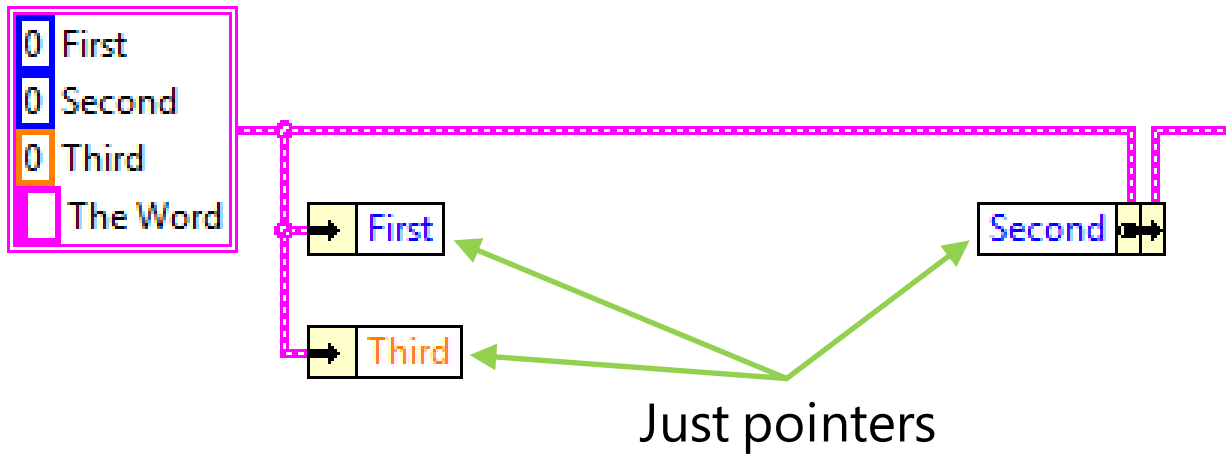
No-op neighbors:



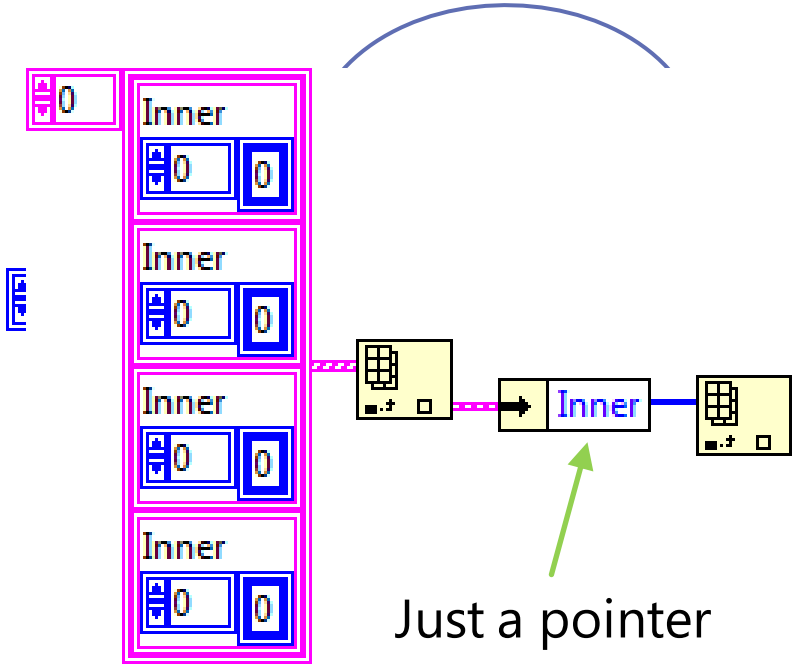




Clusters do not exist (at runtime)



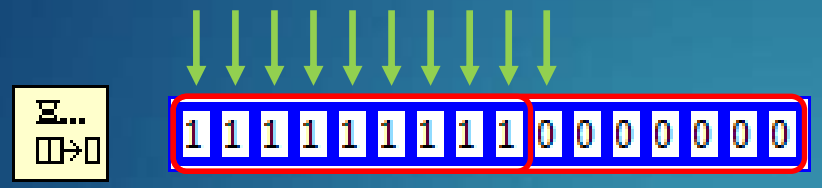
Clusters do not exist (at runtime)



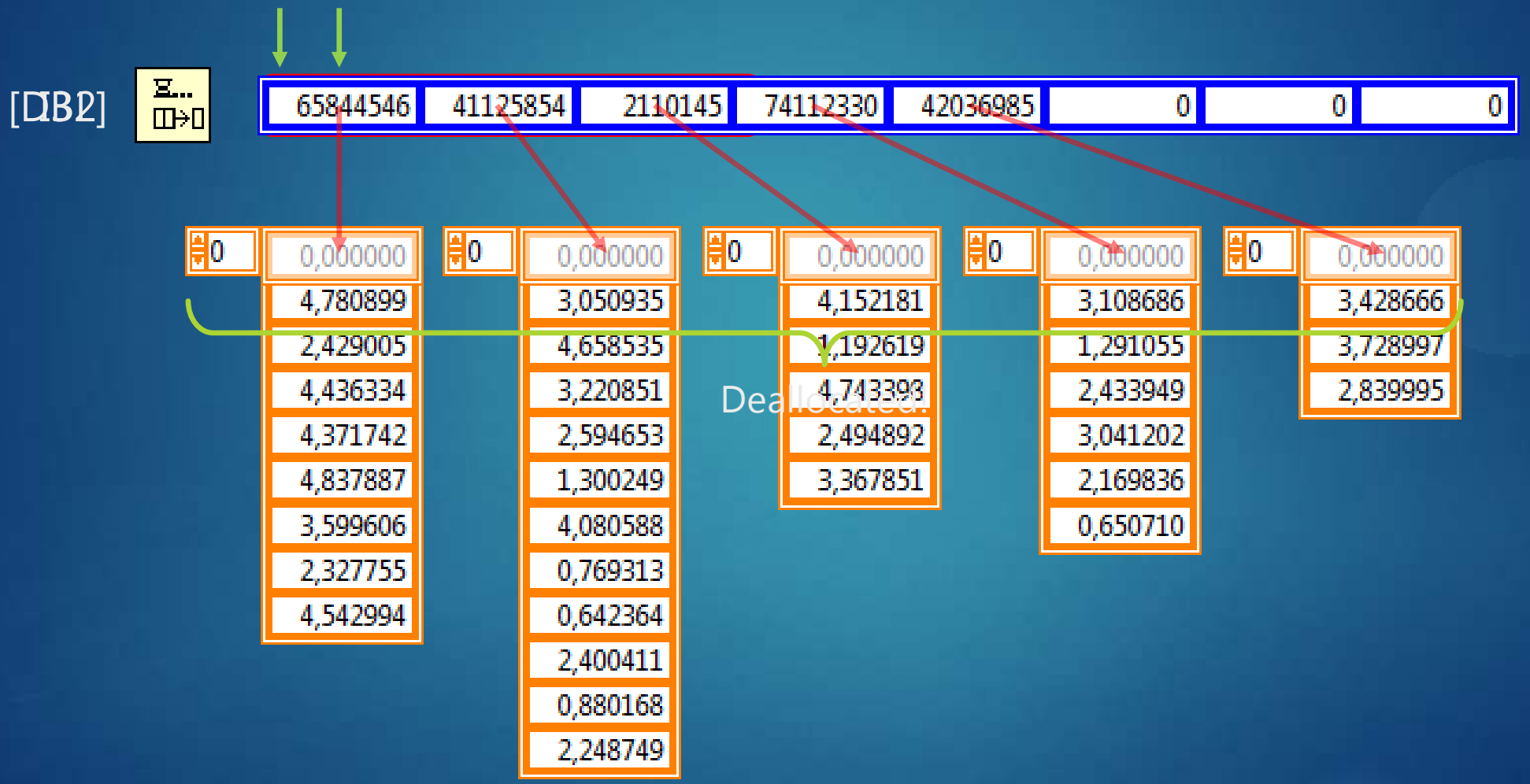
Queues

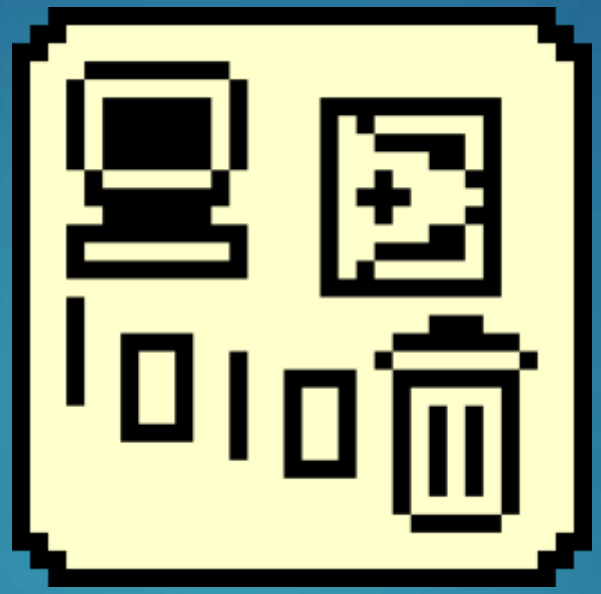
Queue memory allocation

I32



Queue memory (de)allocation

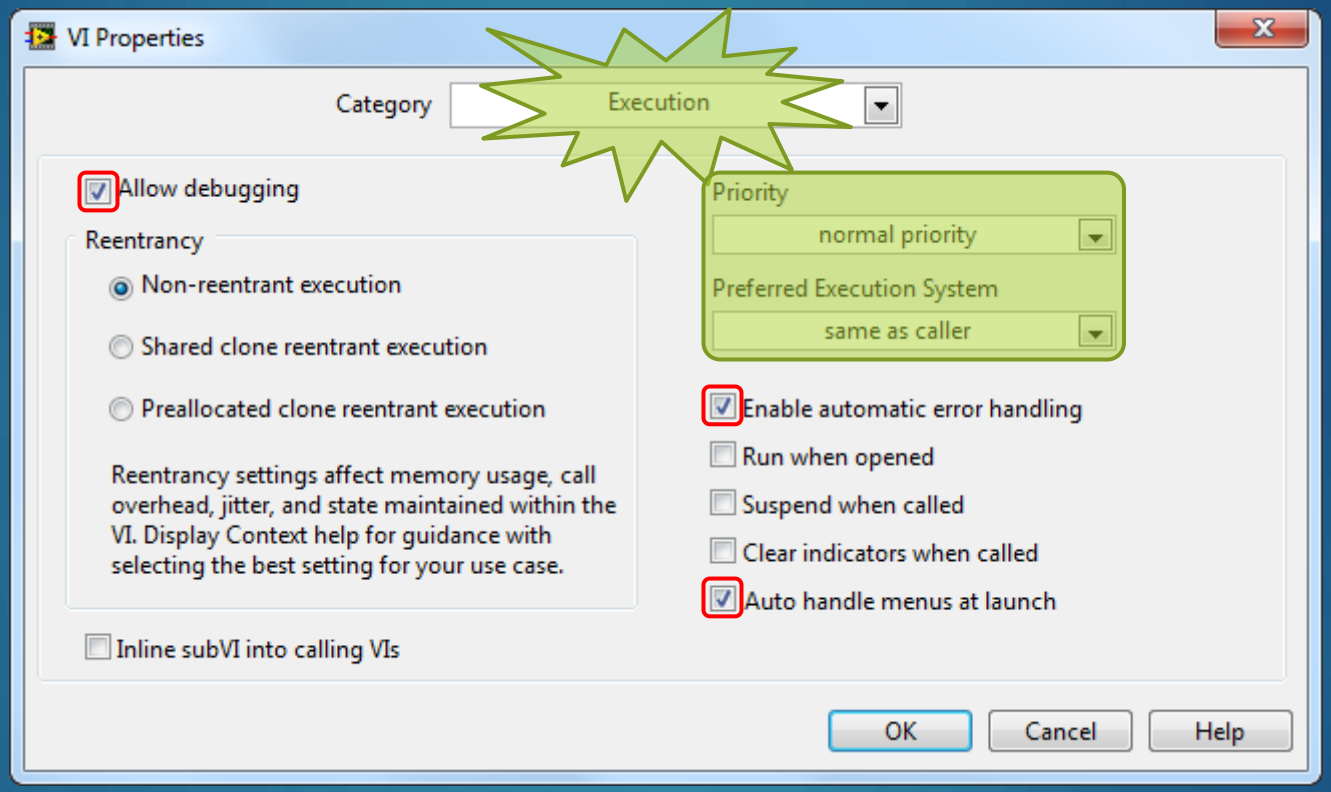




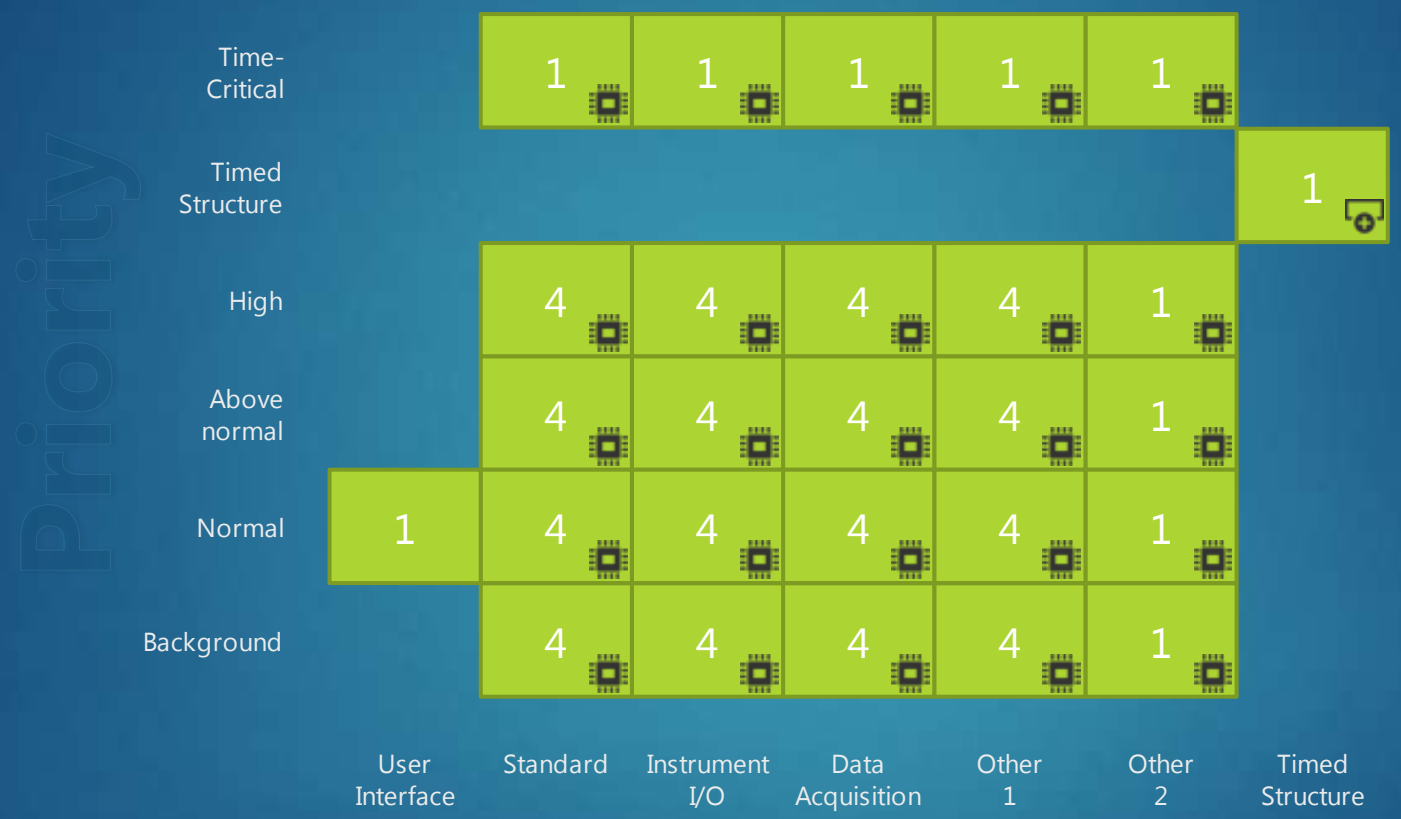
JUST DON’T.

The LabVIEW Execution System

Please set VI execution settings…

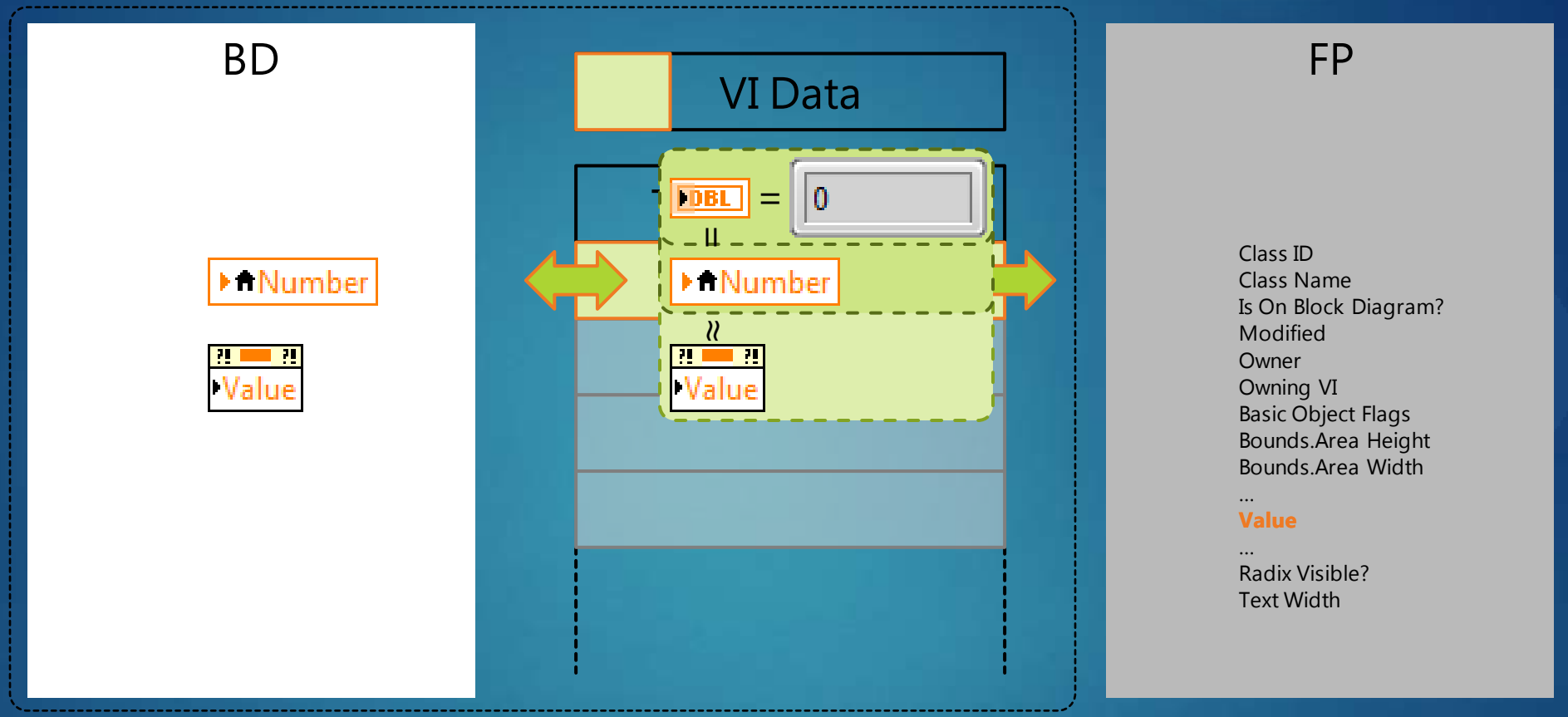


Threads

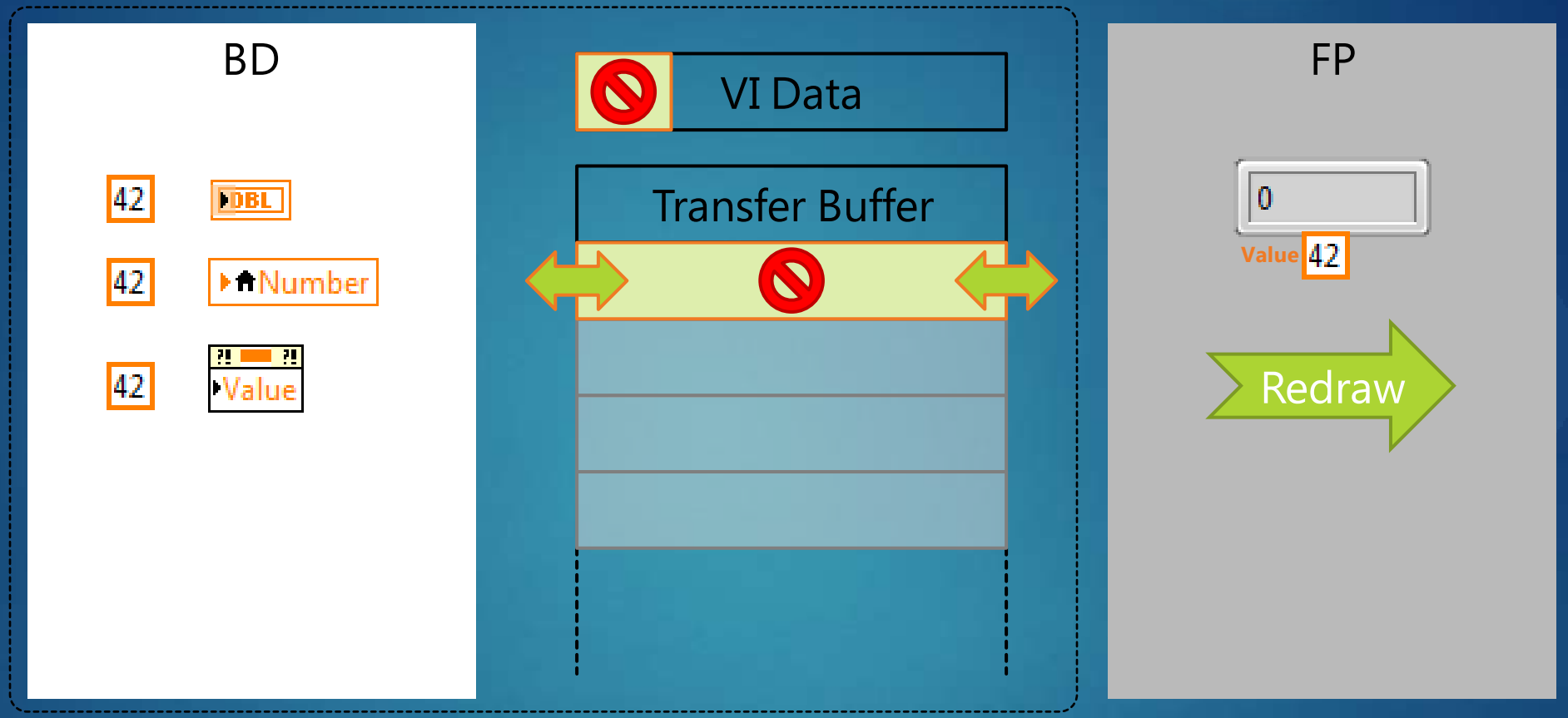

Time-Critical
Timed Structure
High
Above normal
Normal
Background
1
1
1
1
1
1
4
4
4
4
1
4
4
4
4
1
1
4
4
4
4
1
4
4
4
4
1
User Interface
Standard
Instrument I/O
Data Acquisition
Other 1
Other 2
Timed Structure
Priority
Execution System

UI thread and the Transfer Buffer

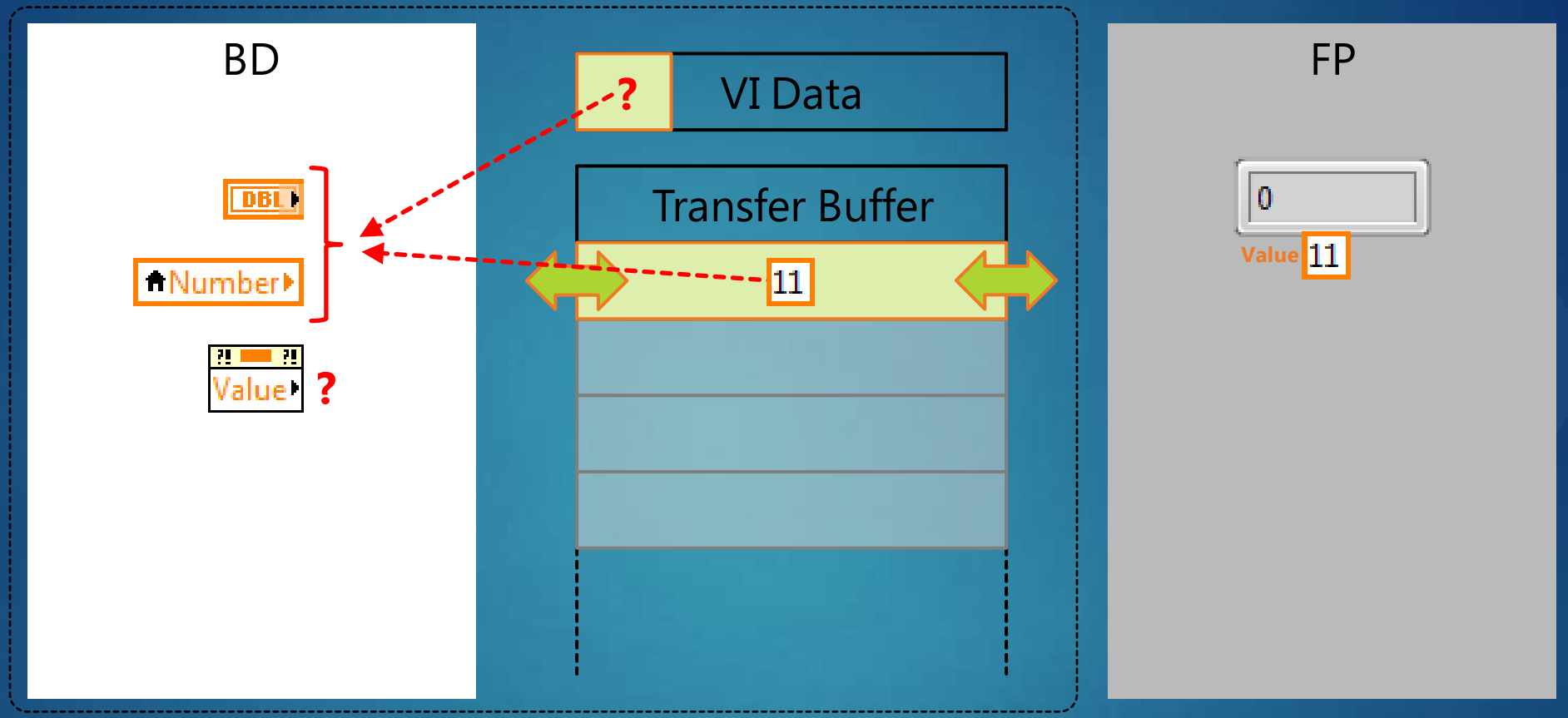
The transfer buffer



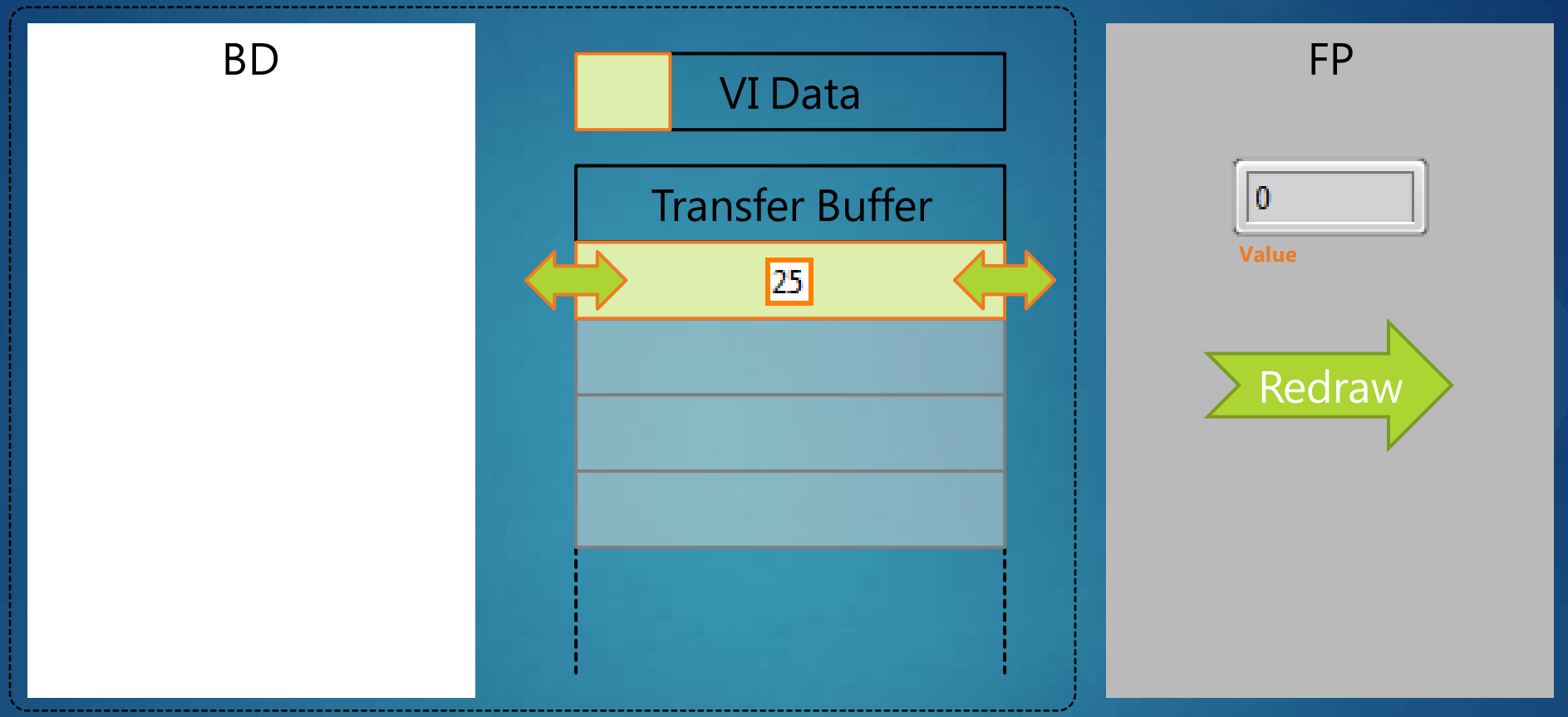
The transfer buffer (Write)



The transfer buffer (Read)

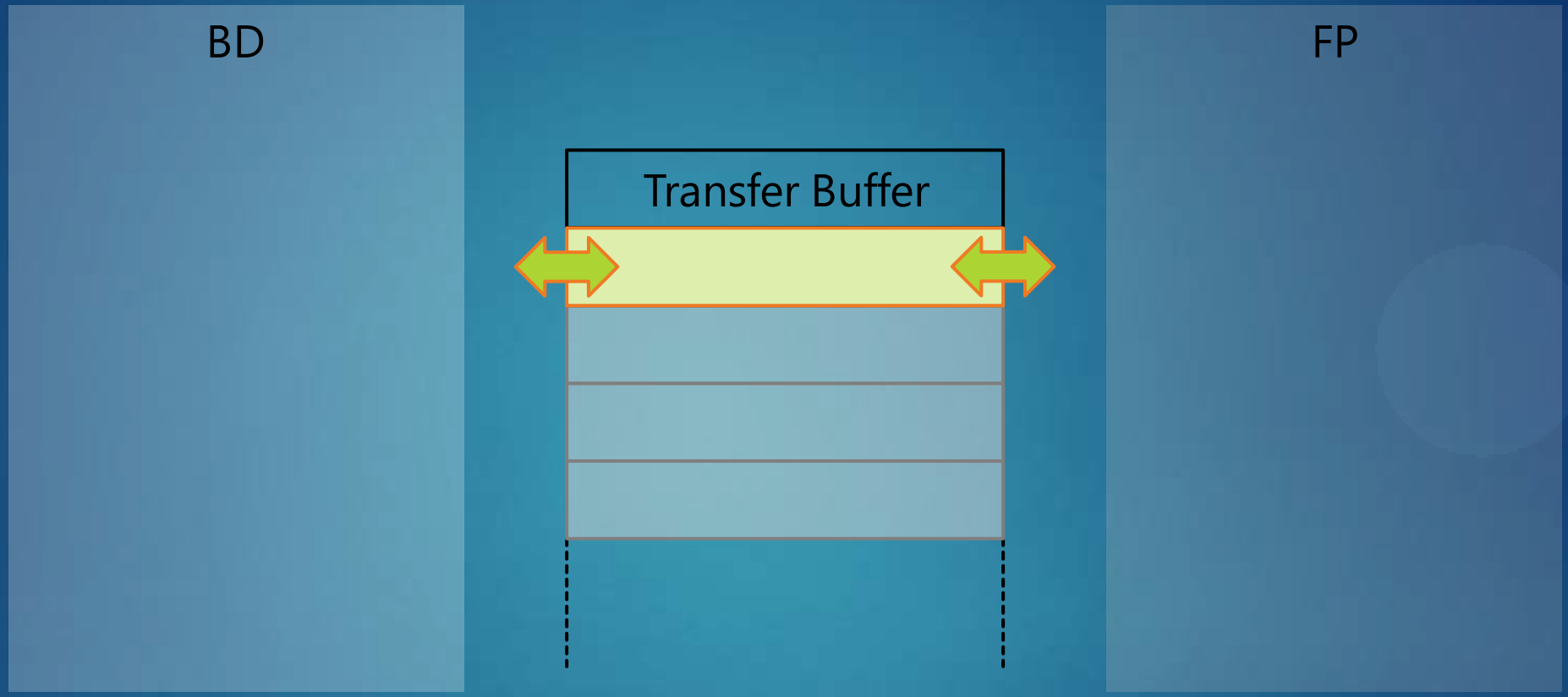


The transfer buffer (FP update)



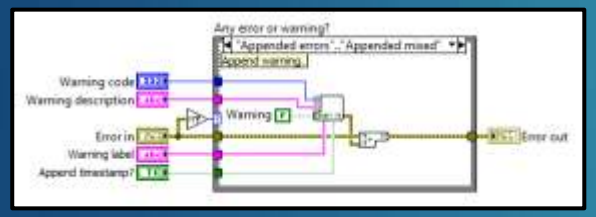
60 Hz

If priority allows!


BD
Transfer Buffer
FP

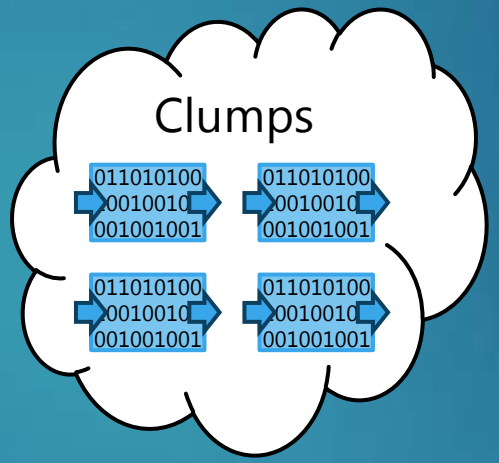
High Priority = Low Performance

LabVIEW compiler and runtime



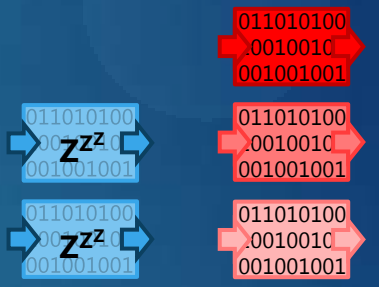
Compile

Clumps

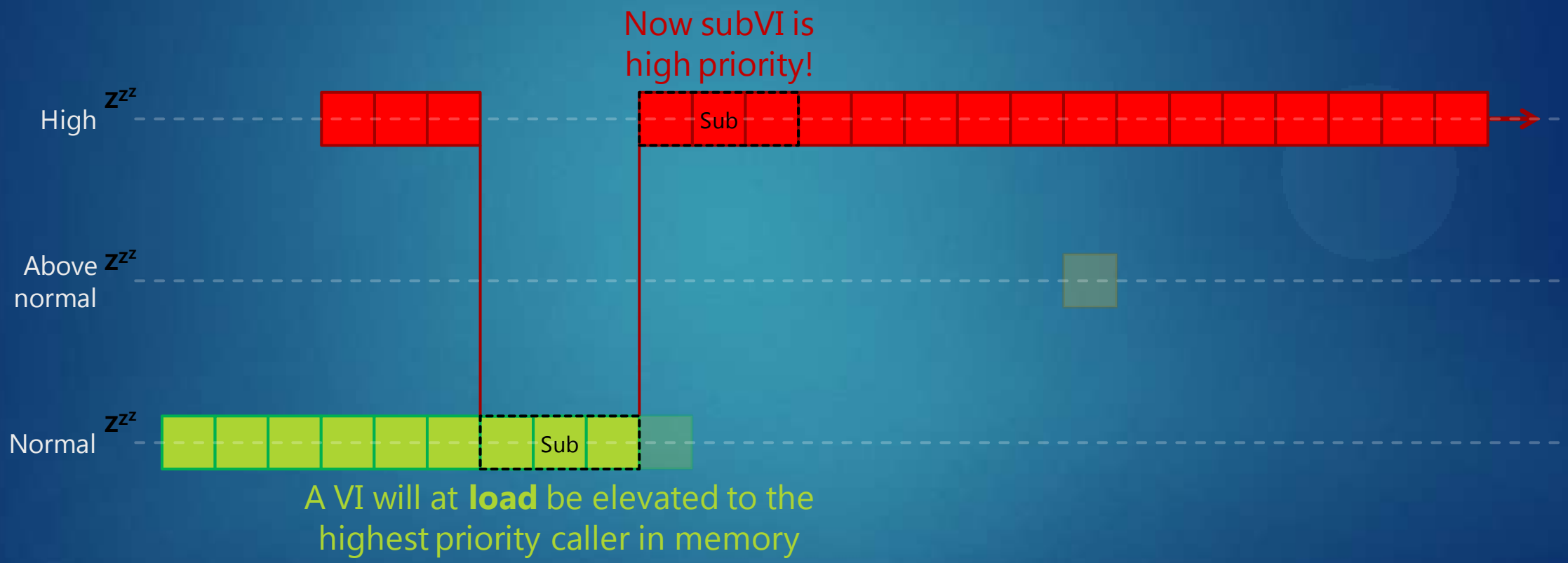


Runtime

Ready to run?
Priority?

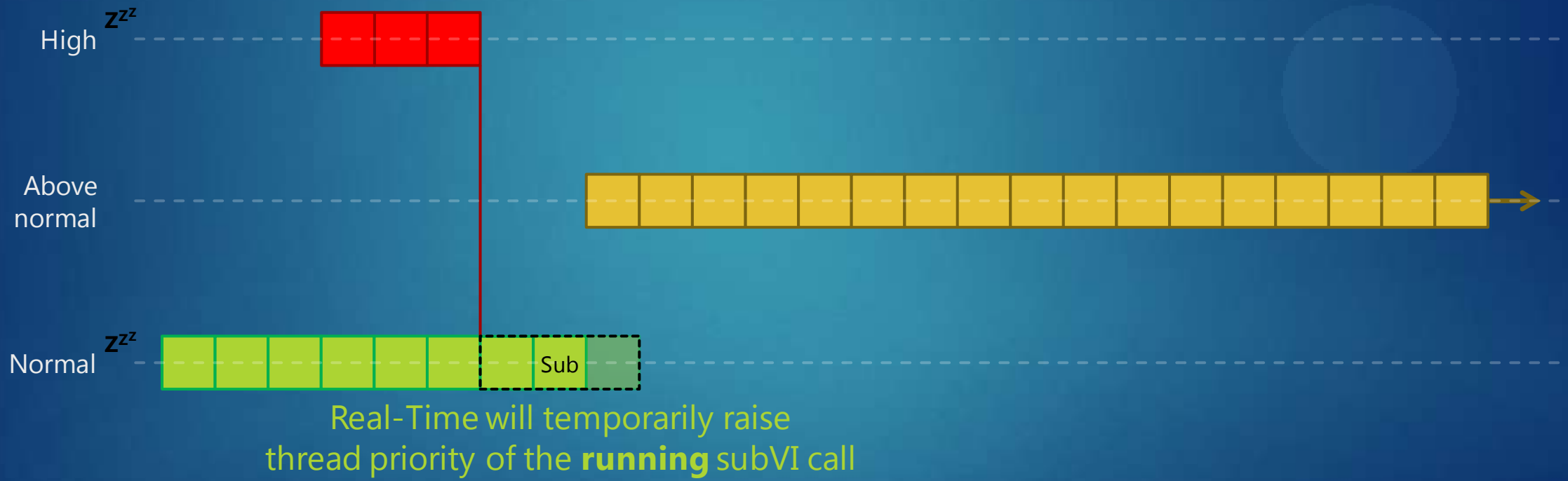


Priority inheritance



A VI will at **load** be elevated to the highest priority caller in memory

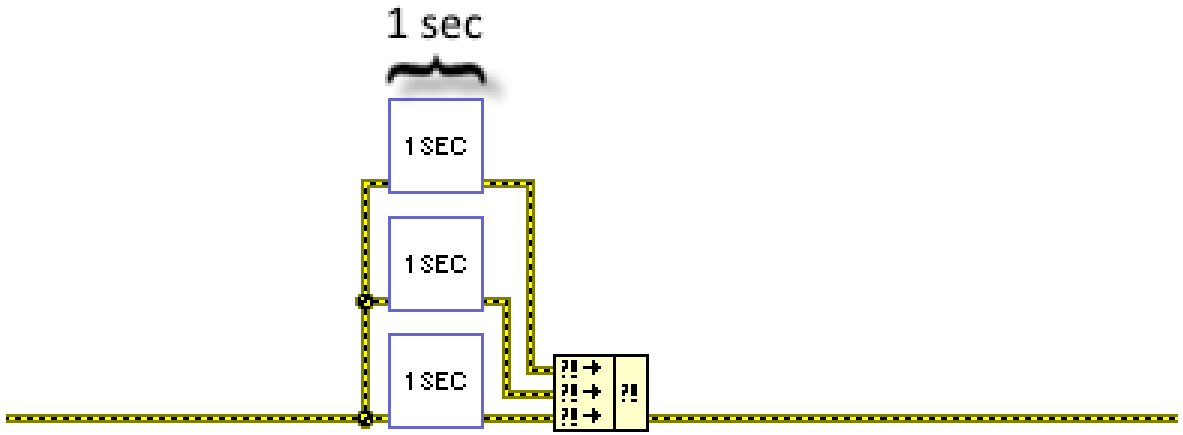
Priority inversion



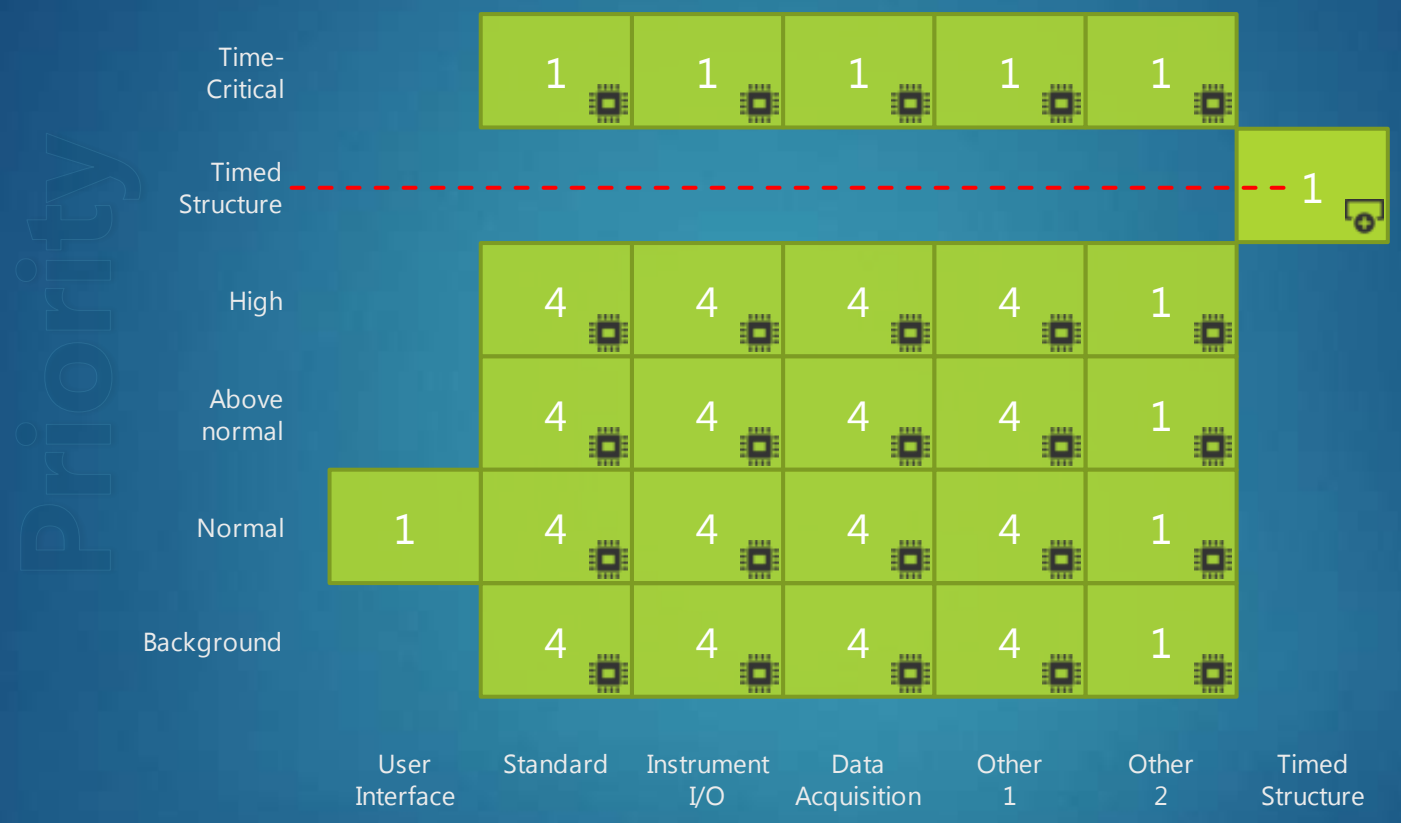
Real-Time will temporarily raise thread priority of the **running** subVI call

No **shared** resources in high priority code!

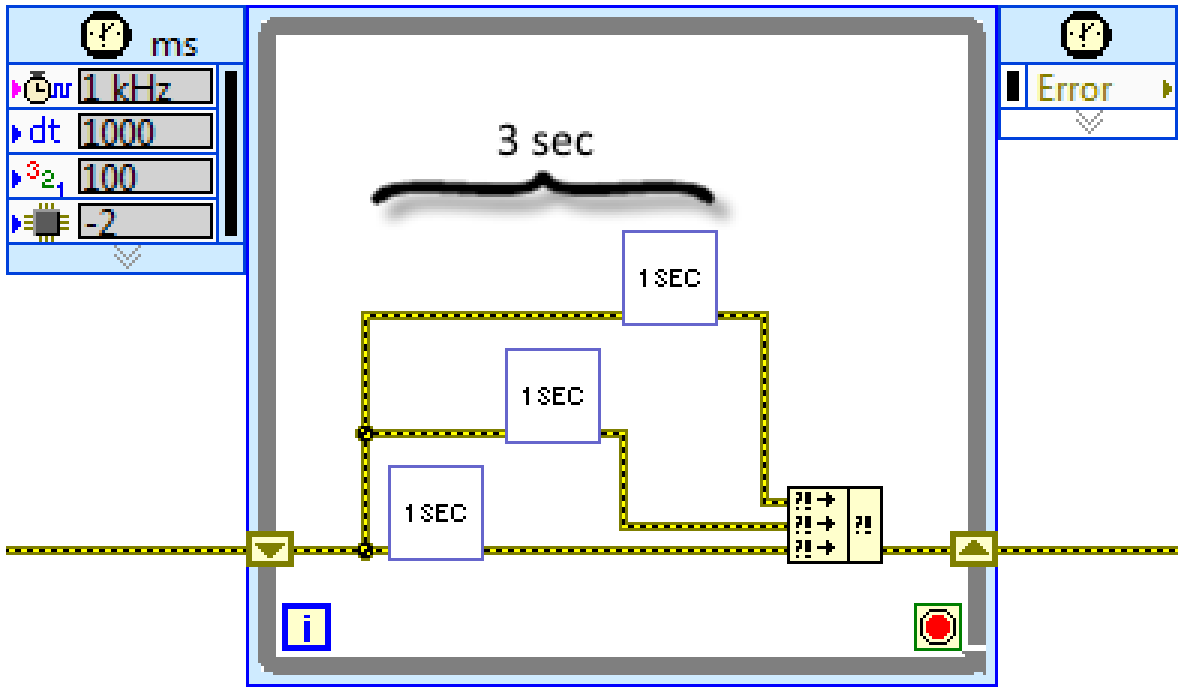
Lower performance by single threading



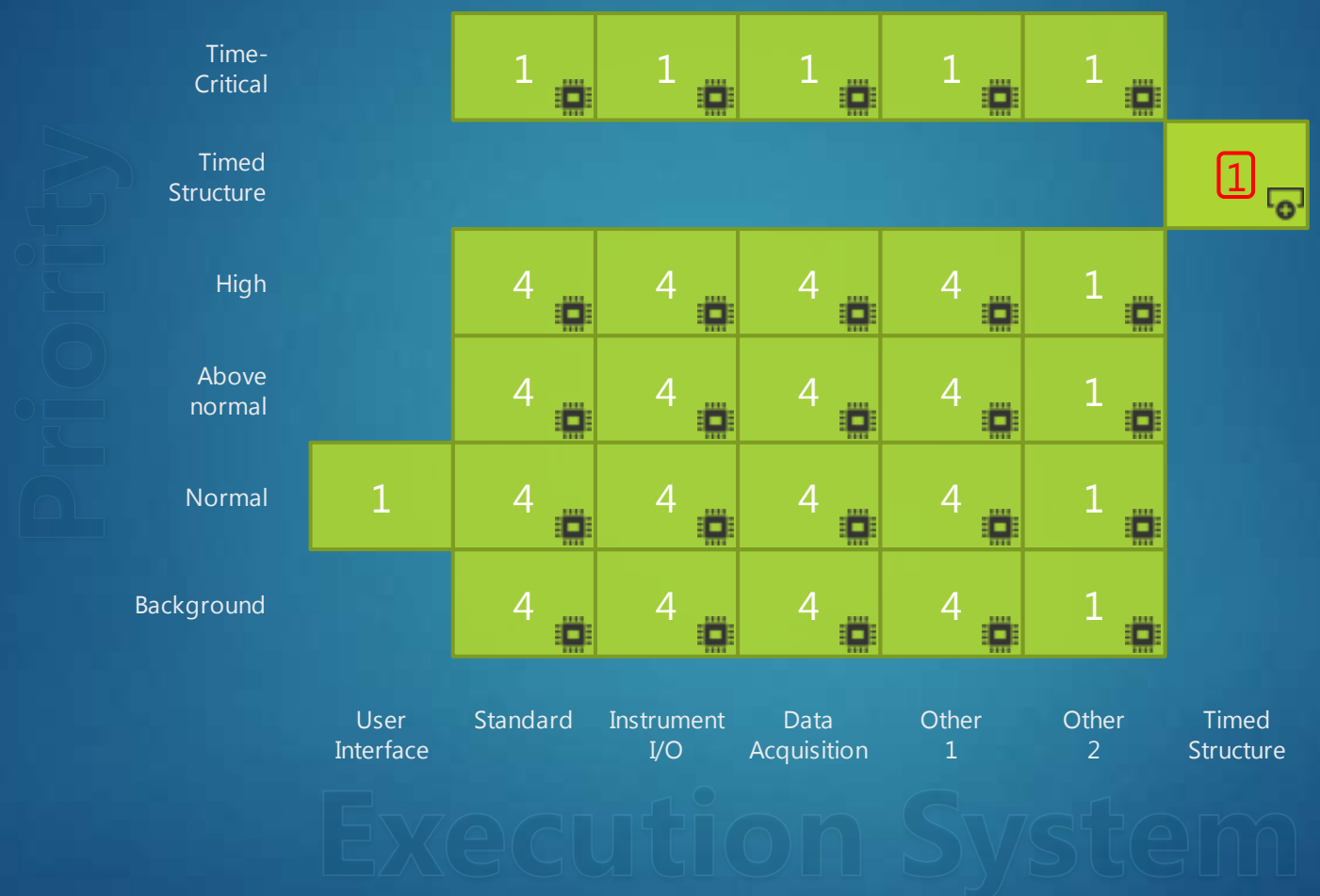
Threads


Priority
Time-Critical
Timed Structure
High
Above normal
Normal
Background
1
4
User Interface
Standard
Instrument I/O
Data Acquisition
Other 1
Other 2
Timed Structure
Execution System

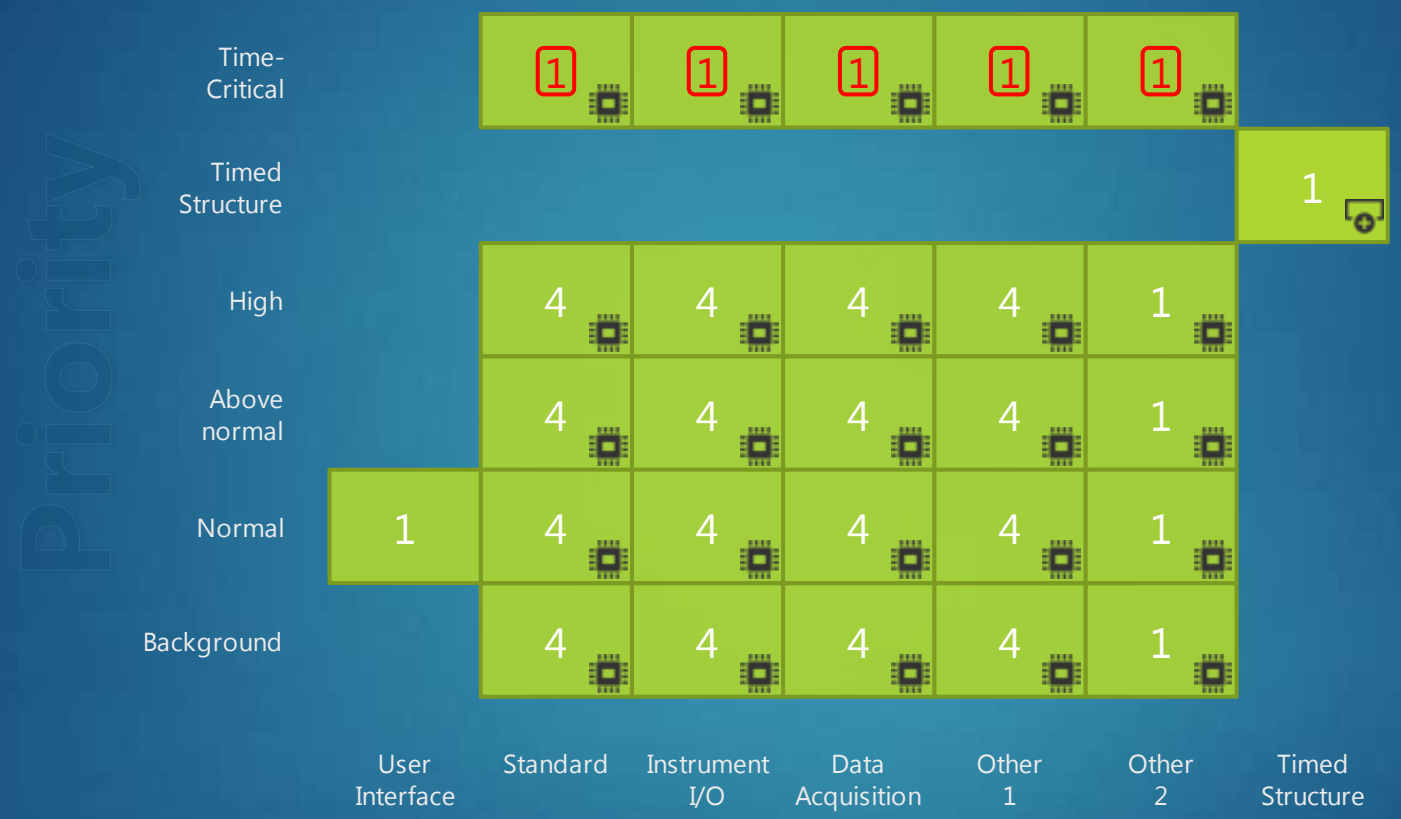
Lower performance by single threading



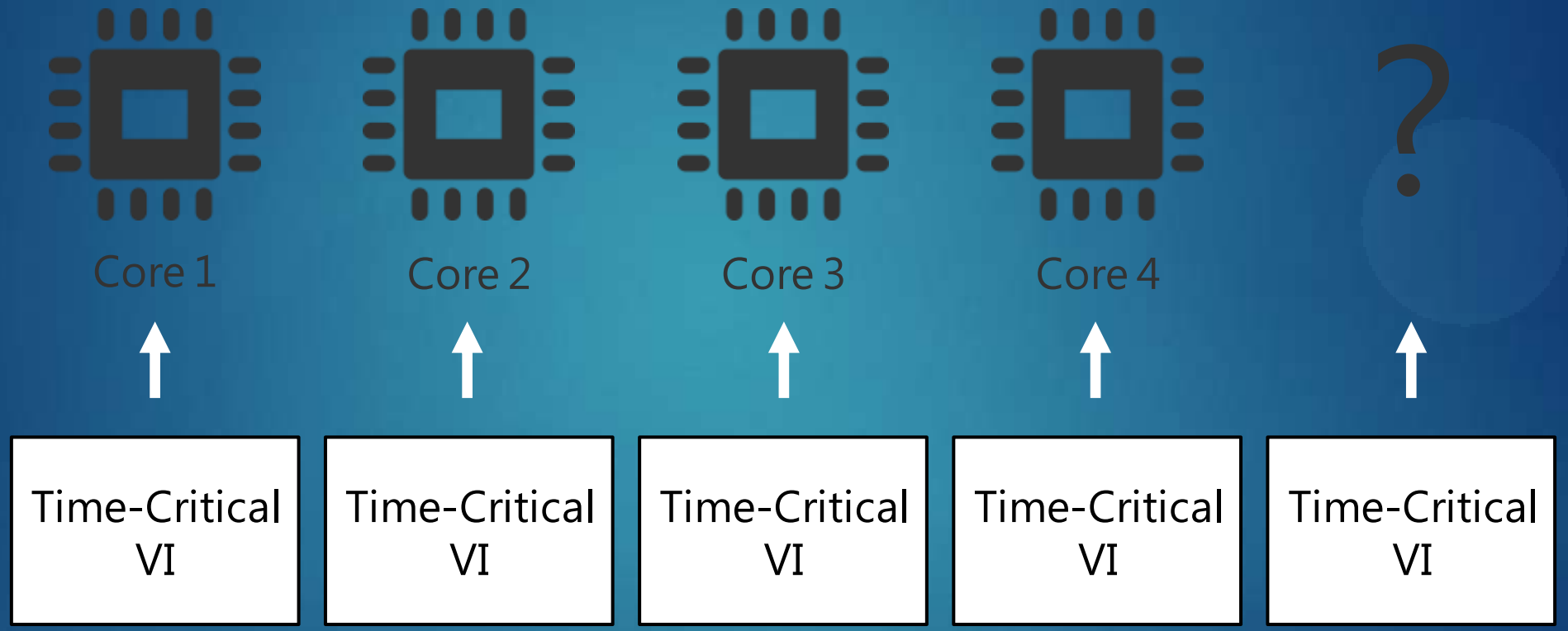
Threads


Priority
Time-Critical
Timed Structure
High
Above normal
Normal
Background
1
4
User Interface
Standard
Instrument I/O
Data Acquisition
Other 1
Other 2
Timed Structure
Execution System

Threads


Priority
Time-Critical
Timed Structure
High
Above normal
Normal
Background
1
4
User Interface
Standard
Instrument I/O
Data Acquisition
Other 1
Other 2
Timed Structure
Execution System

Lower performance by resource starvation



Don’t just raise priority **blindly**!

The Root Loop

Root loop
Demo

The root loop

Uses the root loop

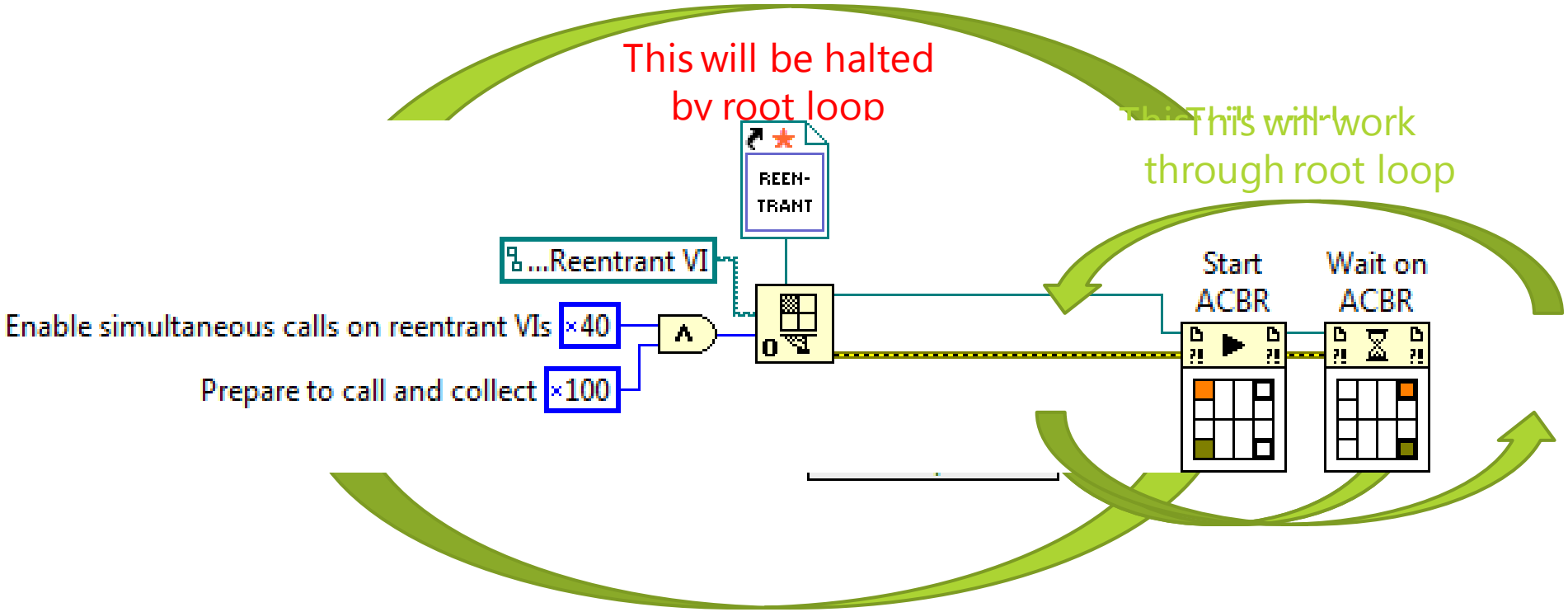
- Menu activation*
- One and two button dialogs*
- Open file dialog*
- Open any typed VI refnum
- Open untyped VI refnum by path
- Run/Abort/Save VI methods

Doesn't use the root loop (examples)

- BD↔FP IO (incl. Get/Set methods)
- Three button dialog
- Open untyped VI refnum by name
- CBR and ACBR nodes
- Growing the shared clone pool
- Dynamic class load by path

*Can run simultaneously in some cases

The "VI pool"


This will be halted
by root loop
This will work
through root loop
REEN-
TRANT
...Reentrant VI
Enable simultaneous calls on reentrant VIs
×40
Prepare to call and collect
×100
Start
ACBR
Wait on
ACBR

Reentrancy

Static subVI call methods

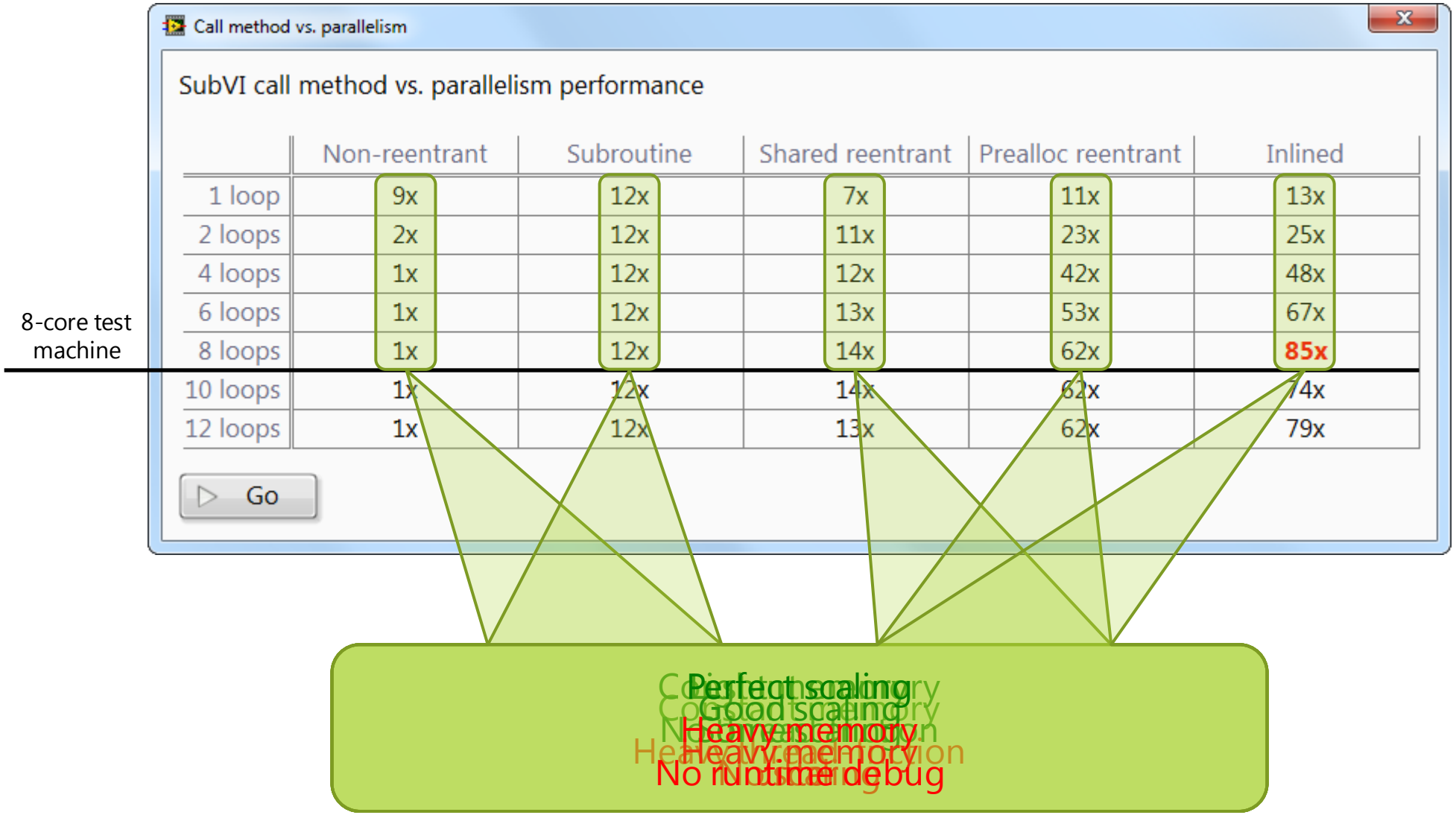
- Non-reentrant
- Subroutine
- Shared reentrant

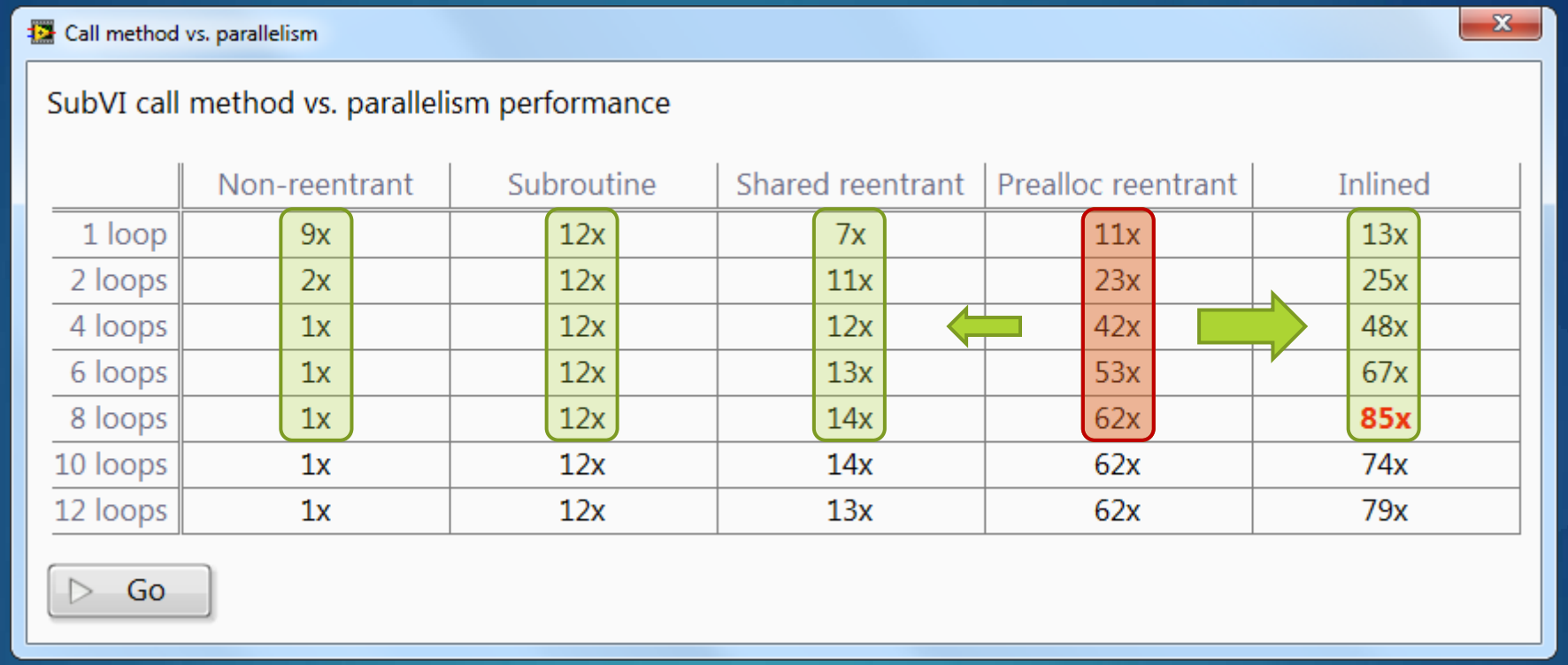
Stateless

- Preallocated reentrant
- Inlined

Stateful

Static subVI call methods **Demo**

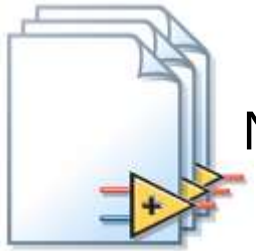

Call method vs. parallelism
SubVI call method vs. parallelism performance
Non-reentrant
Subroutine
Shared reentrant
Prealloc reentrant
Inlined
1 loop 9x 12x 7x 11x 13x
2 loops 2x 12x 11x 23x 25x
4 loops 1x 12x 12x 42x 48x
6 loops 1x 12x 13x 53x 67x
8 loops 1x 12x 14x 62x 85x
10 loops 1x 12x 14x 62x 74x
12 loops 1x 12x 13x 62x 79x
Go
8-core test machine
Perfect scaling
Good scaling
Heavy memory
No runtime debug


Call method vs. parallelism
SubVI call method vs. parallelism performance
Non-reentrant
Subroutine
Shared reentrant
Prealloc reentrant
Inlined
1 loop 9x 12x 7x 11x 13x
2 loops 2x 12x 11x 23x 25x
4 loops 1x 12x 12x 42x 48x
6 loops 1x 12x 13x 53x 67x
8 loops 1x 12x 14x 62x 85x
10 loops 1x 12x 14x 62x 74x
12 loops 1x 12x 13x 62x 79x
Go

Inlining drawbacks

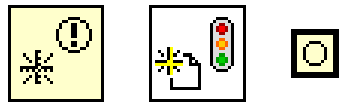
- Preallocated (mem usage)
- Changes to inlined subVI requires caller recompile
- Can be heavy in edit and build (>16 kB)
- No debug
- No front panel
- Some limitations on content

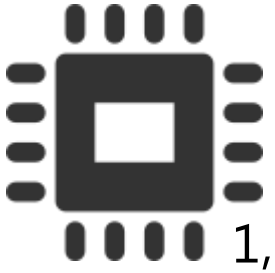
Really shared resources



Non-reentrant

UI thread





1, 2, 3…

Memory Sub-Allocator

Root loop

Minimize **dynamic** mem alloc

Think hard about **resources**

Design **right**, don't fix