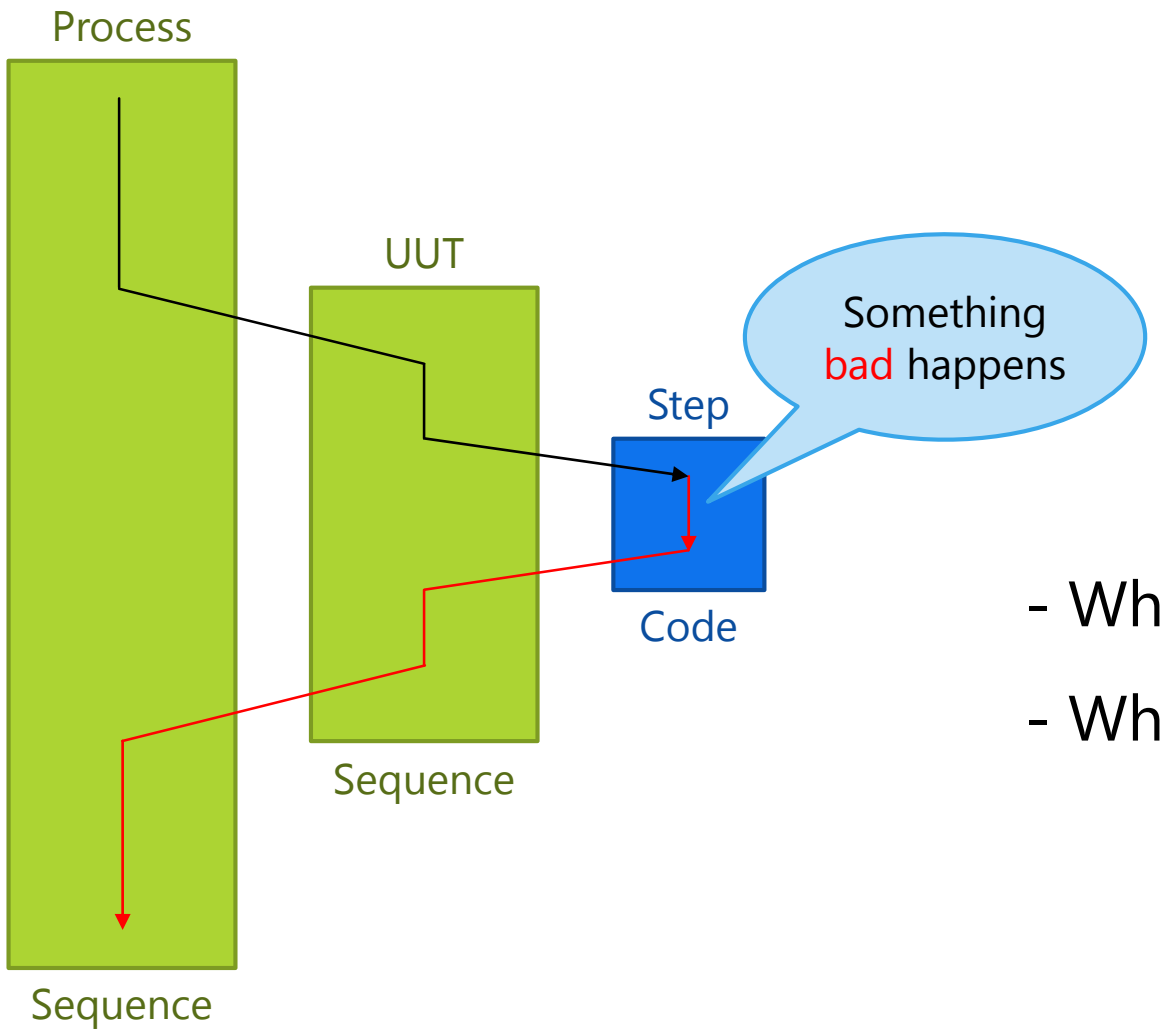
www.gpower.as +45 29 70 89 00 gpower@gpower.as

GPower

Good

TestStand Error Management

GPower



- What is the bad?
- Where does the bad go?

GPower

Today's goal

- The bad is a runtime error (Step.Result.Status = "Error")
  - This happens rarely
  - This happens when something is wrong with the test station
- The bad is *not* a qualification failure (Step.Result.Status = "Failed")
  - This happens more often
  - This happens when something is wrong with the UUT
- Quite often the two get mixed up!
  - This presentation will help you un-mix these failure modes

Agenda

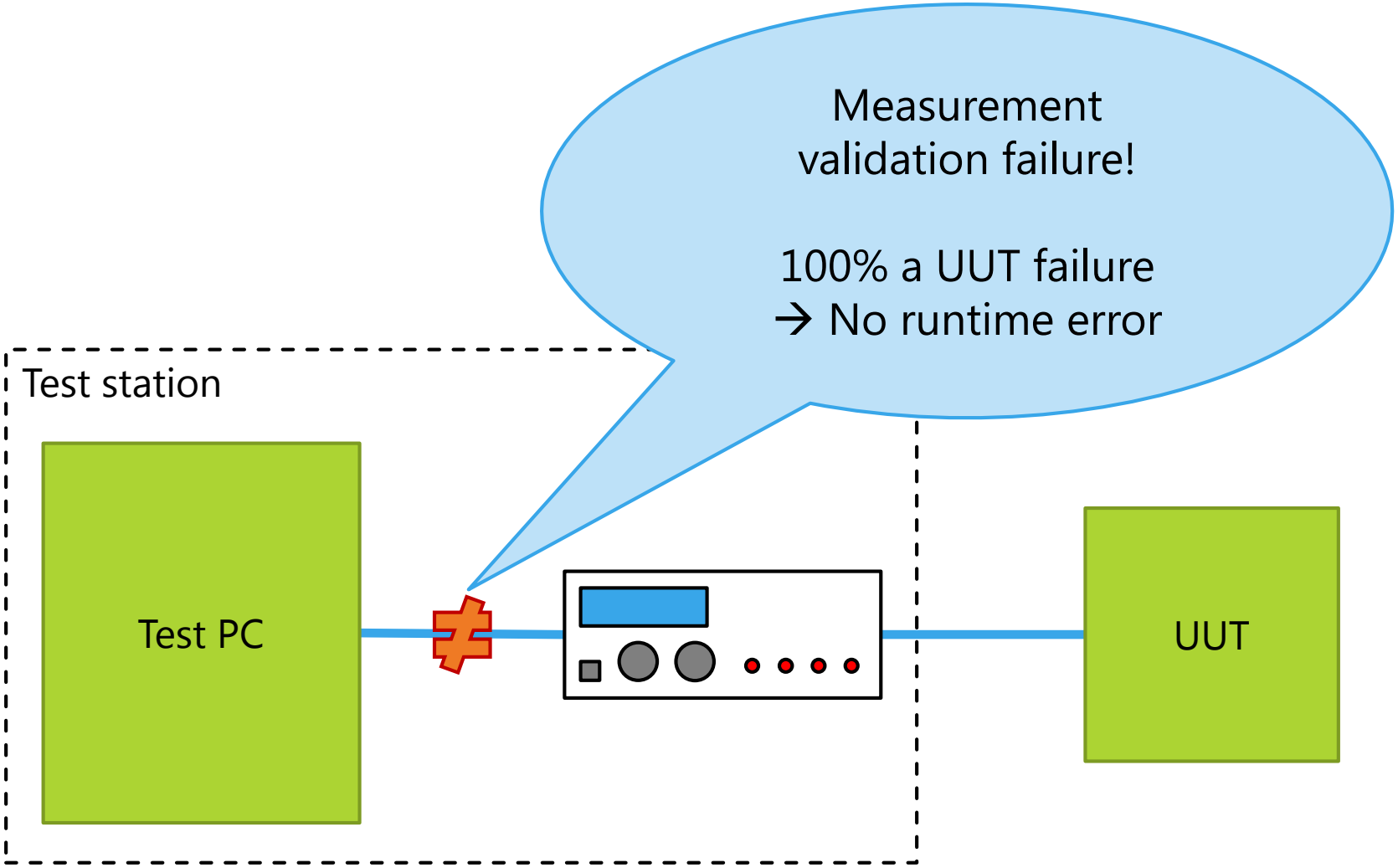
GPower

- Defining runtime errors
- Generating runtime errors
- Presenting runtime errors

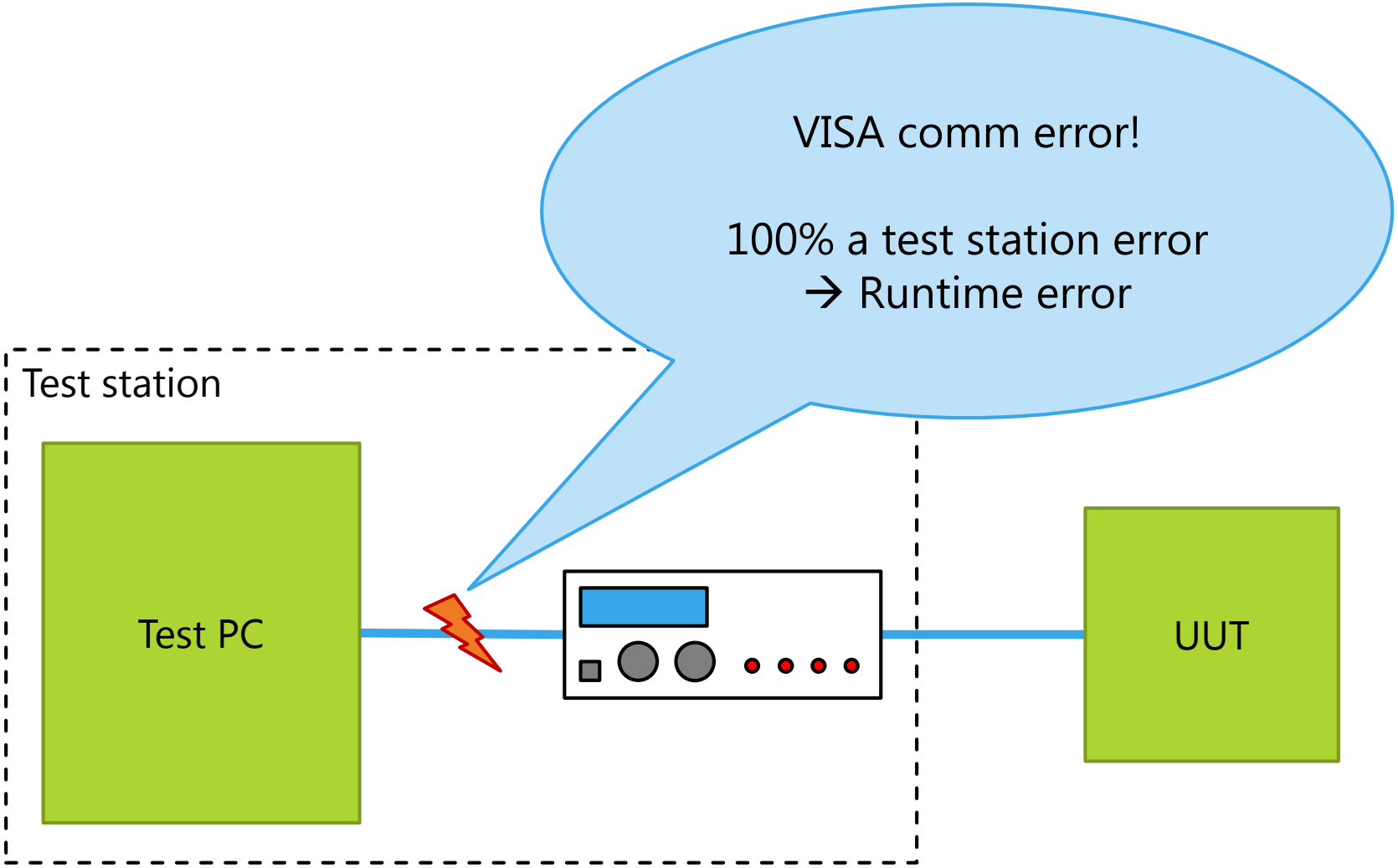
GPower

Defining runtime errors

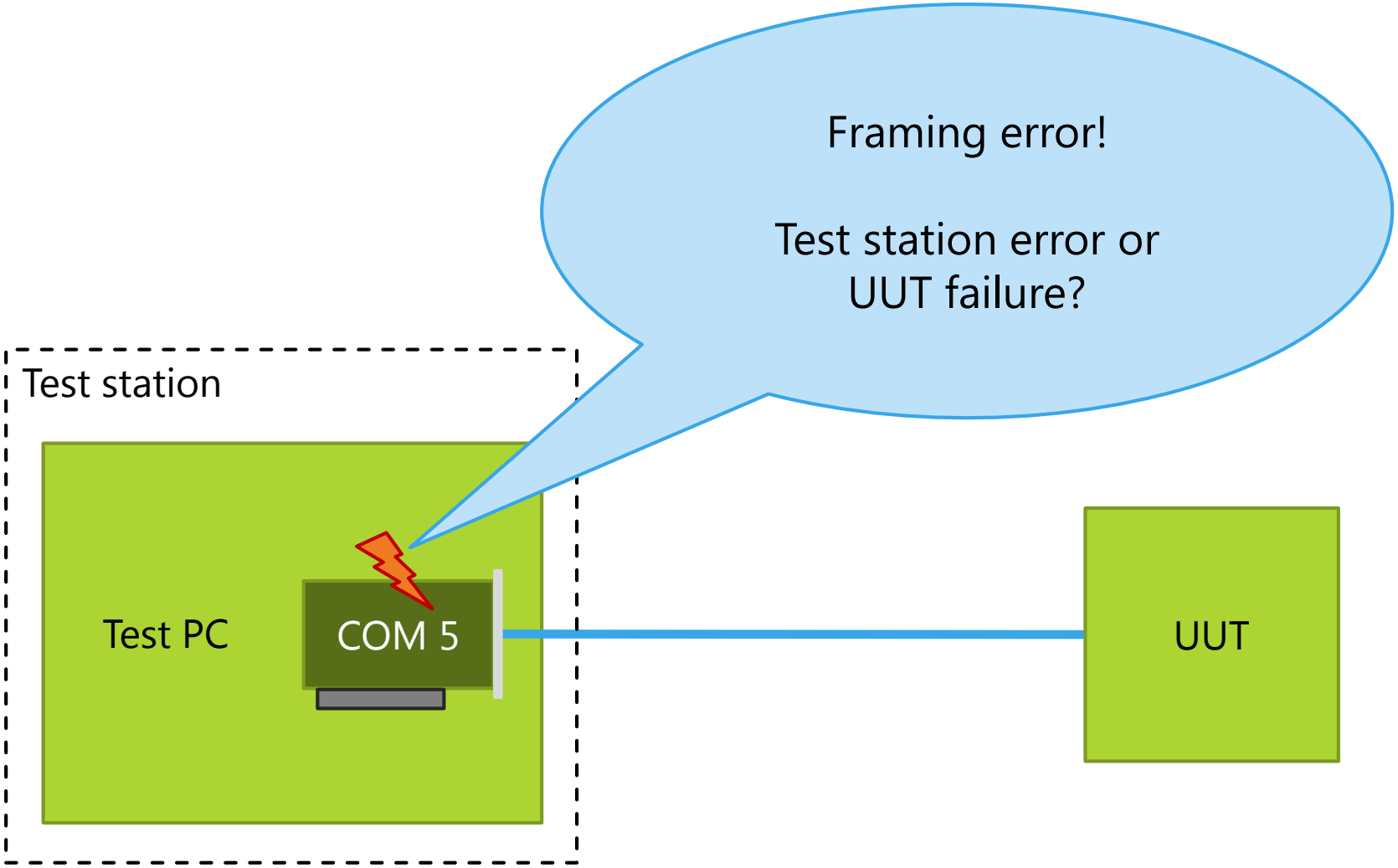
GPower


Measurement
validation failure!
100% a UUT failure
→ No runtime error
Test station
Test PC
UUT

GPower


VISA comm error!
100% a test station error
→ Runtime error
Test station
Test PC
UUT

GPower


Framing error!
Test station error or
UUT failure?
Test station
Test PC
COM 5
UUT

GPower

Best practice 1

Only absolute station errors may cause runtime errors

- If an error *could* be caused by the UUT it *may never* cause a runtime error
- The test step must intelligently decide when to output an error, retrying and filtering internally if necessary
- What to do with other ”errors”? We’ll get back to that…

GPower

Generating runtime errors

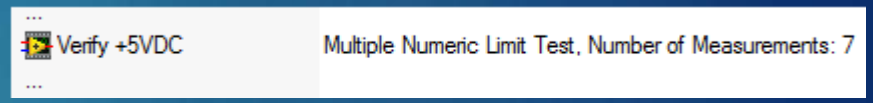
Step.Result.Status

GPower

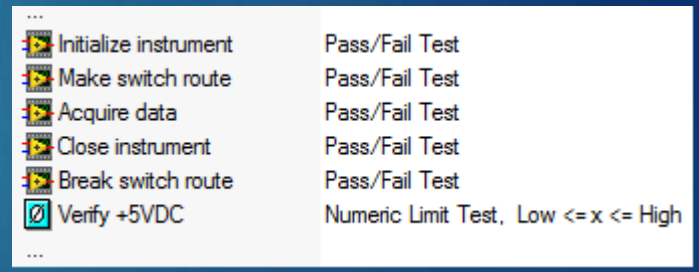
- TestStand calculates Step.Result.Status by combining a number of means:
  - Step.Result.Error container
  - Step type post-step
  - Status Expression
- The calculation rules are convoluted and black-box
  - Thus, it's fragile to manually modify Step.Result.Status

Measurement evaluation methods

- Single-step evalulation
  - Acquisition and evaluation in one step



- Multi-step evaluation
  - N steps that "feel their way along", talks to the instrument, communicates with the UUT, and acquires the data
  - A final step that evaluates the acquired data if all went well
  - Any error along the way places responsibility on that step, and can skip the rest
  - A bit fragile, can be hard to maintain



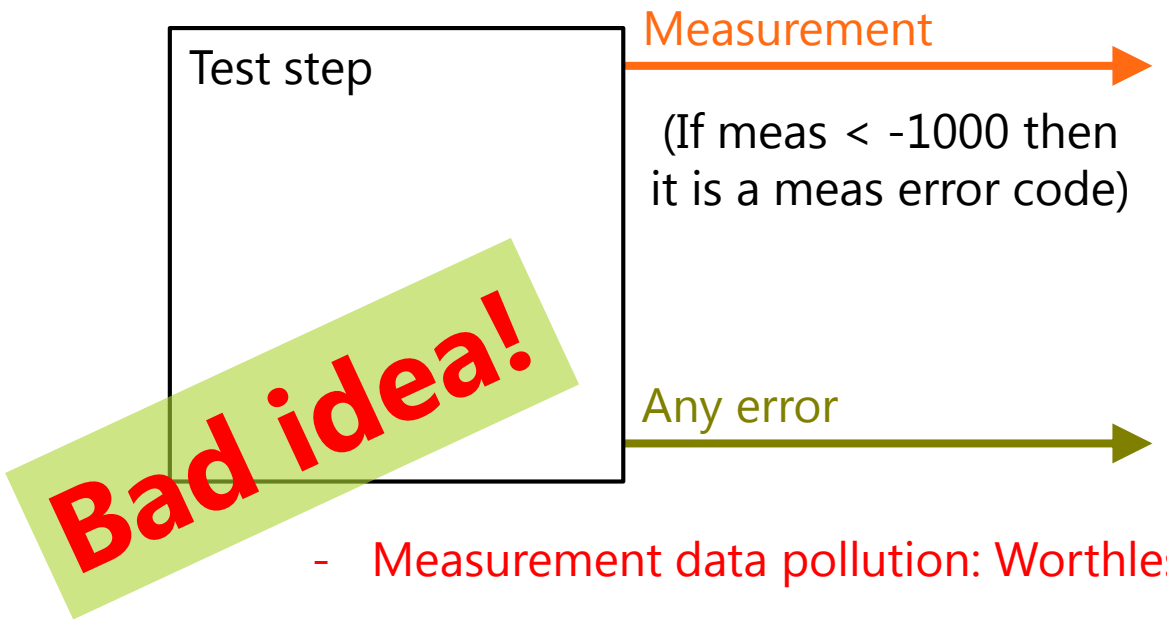
GPower

Best practice 2

One step per test

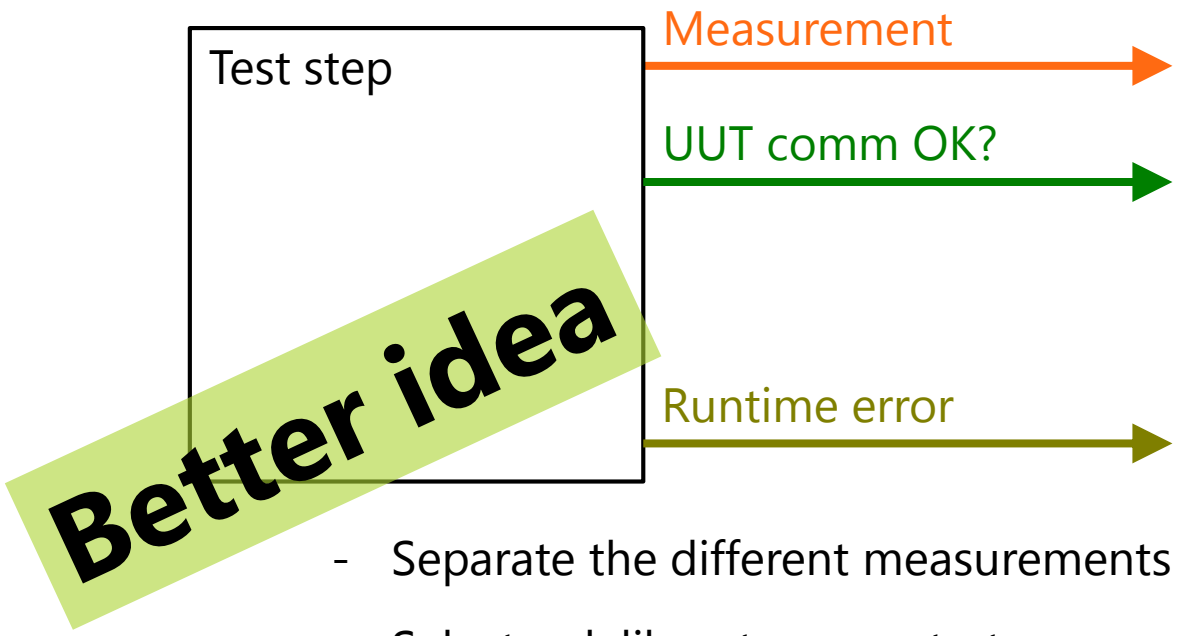
- Single-step evaluation
- Or, if multi-step evaluation for component re-use, must be in subsequence
  - Don’t trace into that subsequence, present it as a single step to the operator

GPower



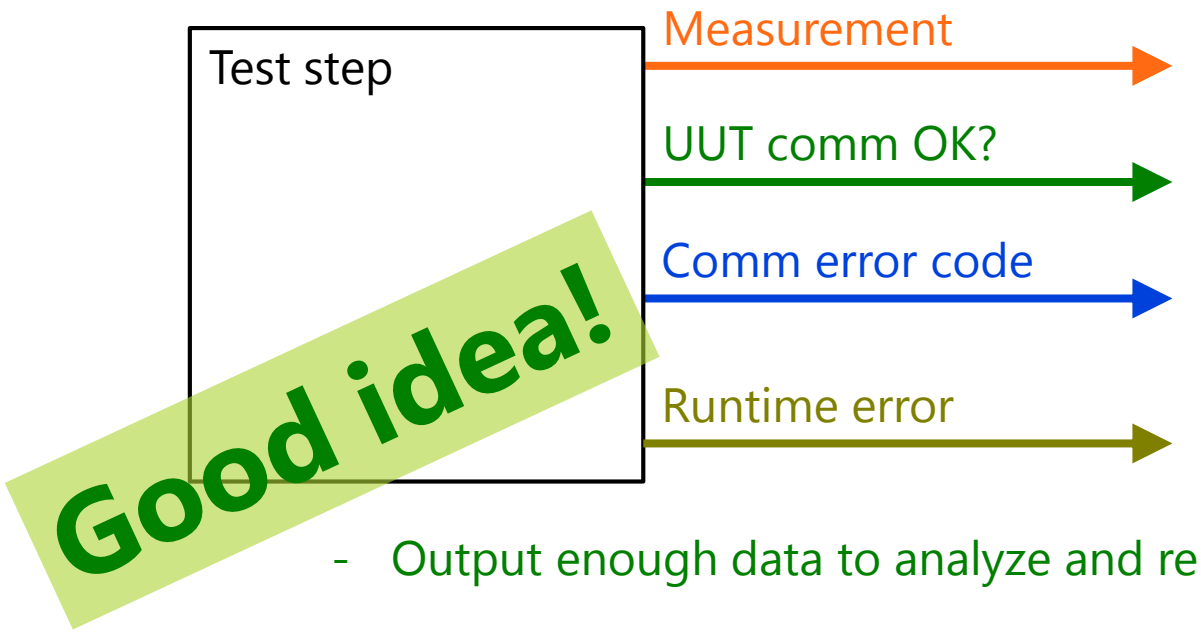
- Measurement data pollution: Worthless SPC
- Proprietary meas error codes: Unmaintainable
- Unchecked error output: Surprises, surprises
- Have to modify Step.Result.Status, SequenceError.Occurred, SequenceFailed etc. by expression: Fragile

GPower



- Separate the different measurements
- Select a deliberate error strategy

What do we output from test steps?



- Output enough data to analyze and repair failures

GPower

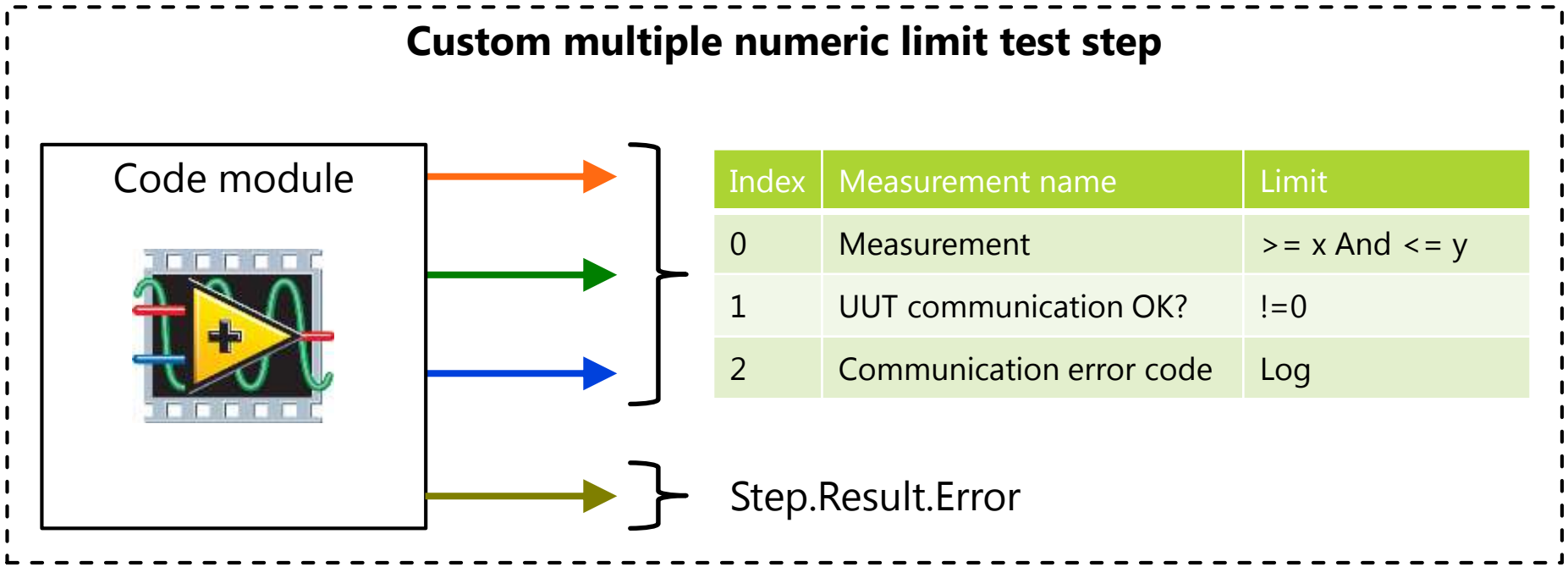
Best practice 3

One result per measurement

- Output multiple measurements in each their own data channel
  (don't create fake measurement results and attempt to modify Step.Result.Status yourself)

- Output runtime errors directly into Step.Result.Error (LV: error out cluster)
  (but *only* runtime errors)

Accept that you actually measure multiple things on the UUT at the same time



GPower

But what when I get ”framing errors” all the time? Can’t I categorize them as runtime errors?
(They are hardware errors after all…)

- A framing error (or similar) *might* be a hardware error, but not necessarily on the test station. Jumping to conclusions not based on fact is unprofessional

- If a framing error (or similar) *really* disturbs your production that much it’s just a proof that your SPC works!

  (You have unacceptably low production yield: analyze and then fix. Repeat as necessary)

- Yes, it might turn out to be a station fault haunting you, but then fix that and watch your framing errors drop and your yield go up as a consequence.

GPower

Best practice 1 (repeat)

Only absolute station errors may cause runtime errors

- If an error *could* be caused by the UUT it *may never* cause a runtime error
- The test step must intelligently decide when to output an error, retrying and filtering internally* if necessary

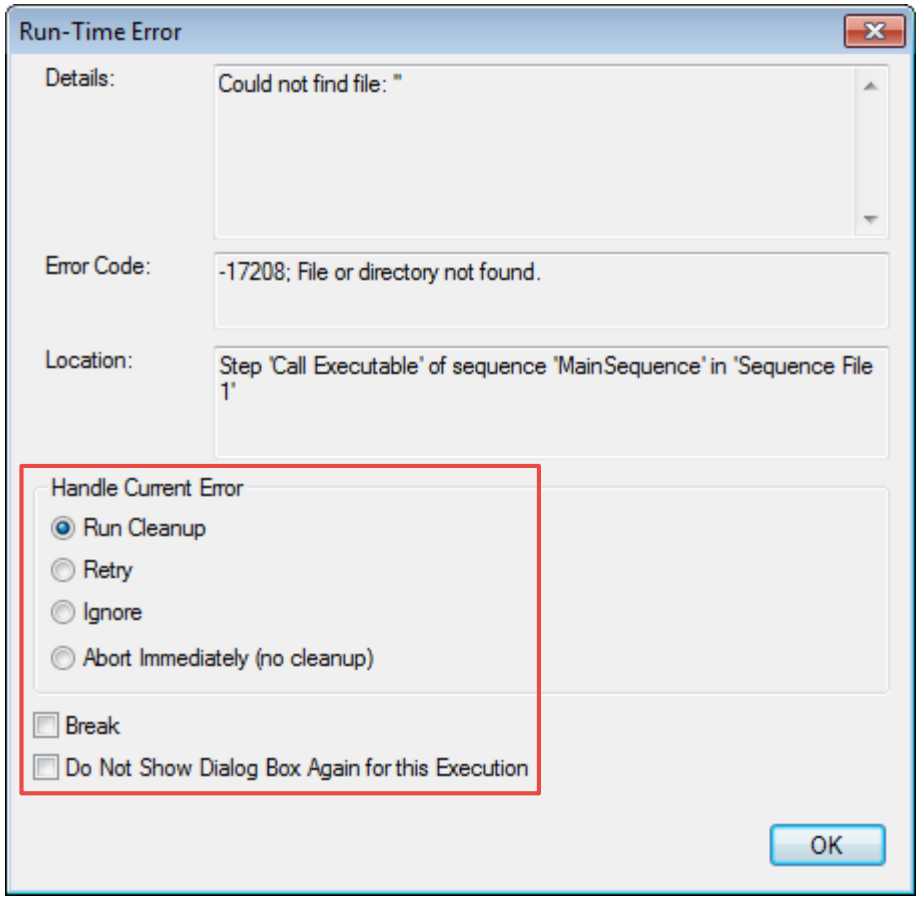
  * In the code module/subsequence

GPower

Presenting runtime errors

Default TestStand dialog on runtime error

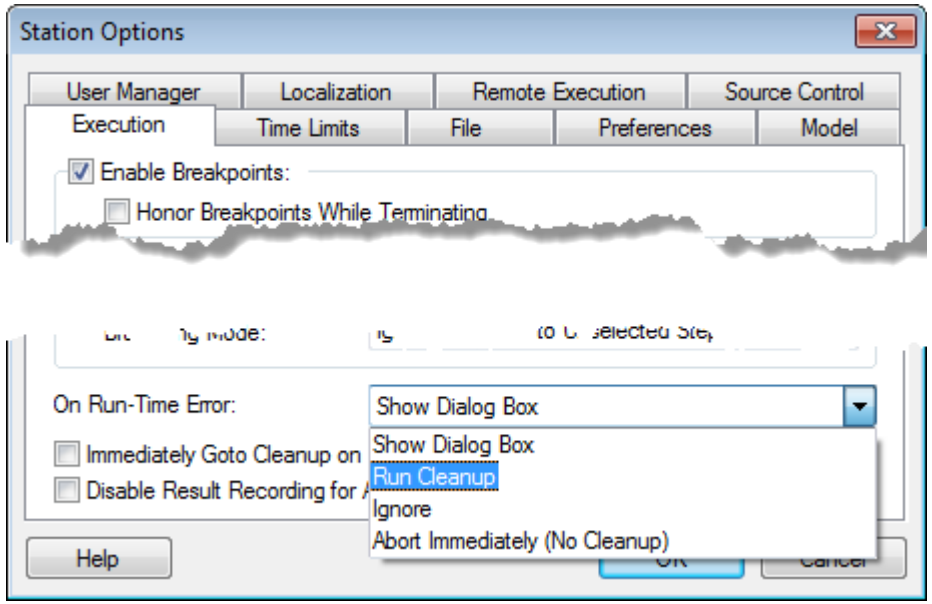
GPower



- **Operator** isn't fit to make this choice
- **Unknown** final system state (what will operator choose?)
- **Blocks** execution, doesn't support parallel execution
- For run-in and debug only

Set station option: Always run cleanup on runtime error

GPower



- **Test developer** is equipped to design escape path
- **Known** final system state
- **Central** error collection and presentation (non-blocking, can support parallel executions)

GPower

Why never **ignore** runtime errors?
(can be done on both step- and sequence-level)

- The runtime error doesn't go away by ignoring it
- Configuring to ignore runtime errors are usually because you're unsure of *which* runtime errors might pop up when you as developer aren't around to help the operator
- Take control: Filter and manage (in the code module) those errors which you are certain are not runtime errors. The rest will be something someone needs to take a look at – also if they happen after you've handed over the sequence

GPower

Why never **abort immediately** on runtime errors?

- That's the most unsafe time to abort
- If something dangerous can happen the operator will always have a safety switch within reach
- Aborting leaves your system in an unknown and possibly energized state. Dare you even restart it from that state? What happened to the data – did it pollute your SPC?
- Shut down safely instead. If you need to shut down one thing faster than other things, then do that. As developer you know of these priorities, the operator does not.

GPower

Best practice 4

Always ‘run cleanup’ on runtime error

Collecting runtime errors

- Central handling
  - Central point of logging, inspection, and action
  - Transfer through sequence/engine callbacks, and model plugins for no modification of process model
- Asynchronous dialog
  - Does not block execution
  - Errors can be appended on the fly, e.g. if multiple parallel executions experience errors

Presenting runtime errors

GPower

- Error attributes included
  - Source (sequence call chain), UUT ID, test socket, timestamp…
  - Can be generated in callbacks or deduced by the central handler
  - Customer specific generation and presentation layer
- Verbose error descriptions
  - Central dialog can play wizard (localization/repair/extra info DB)

GPower

Best practice 5

Build a central runtime error manager

GPower

- 1: Only absolute station errors may cause runtime errors
- 2: One step per test
- 3: One result per measurement
- 4: Always ‘run cleanup’ on runtime error
- 5: Build a central runtime error manager