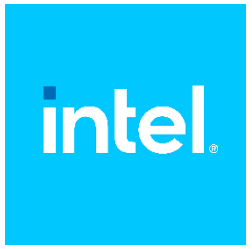
FROM CHIPS TO SYSTEMS — LEARN TODAY, CREATE TOMORROW

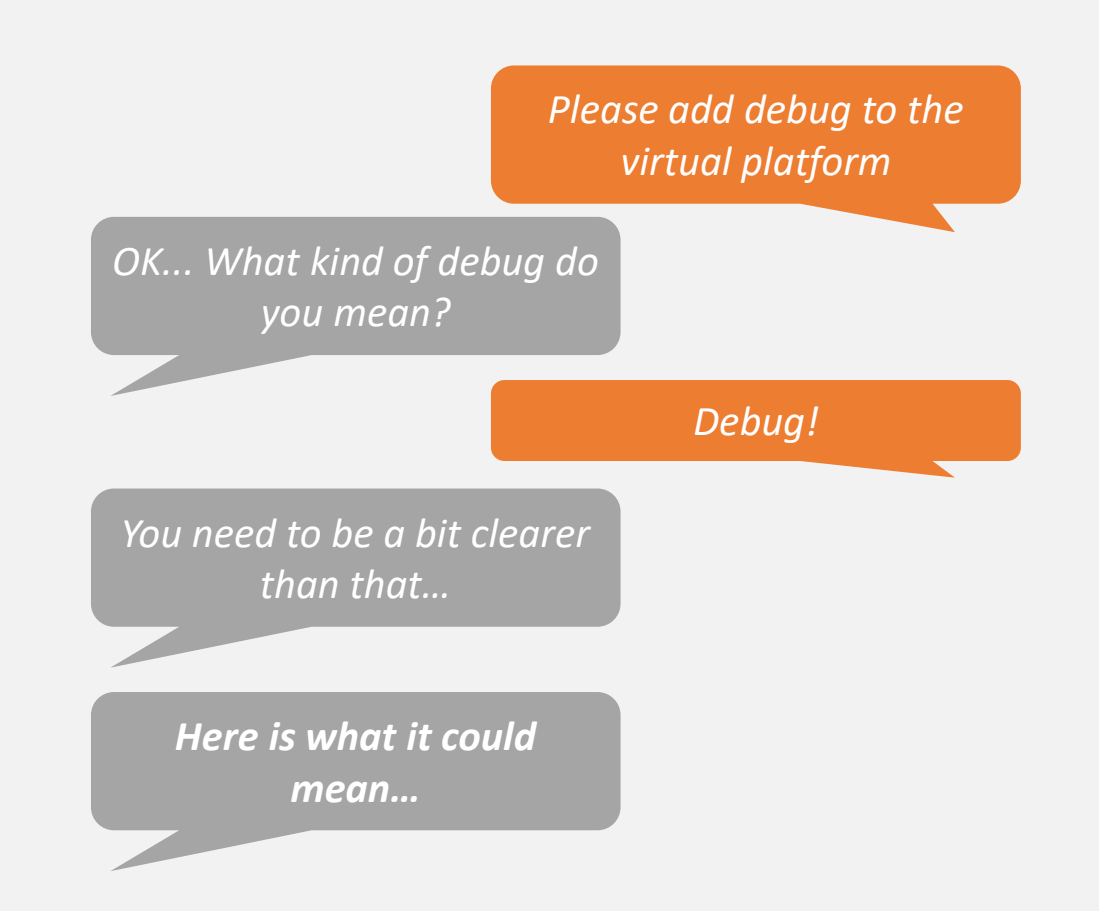
“Debug in/on/with the Virtual Platform” – Please Clarify!

Jakob Engblom, jakob.engblom@intel.com



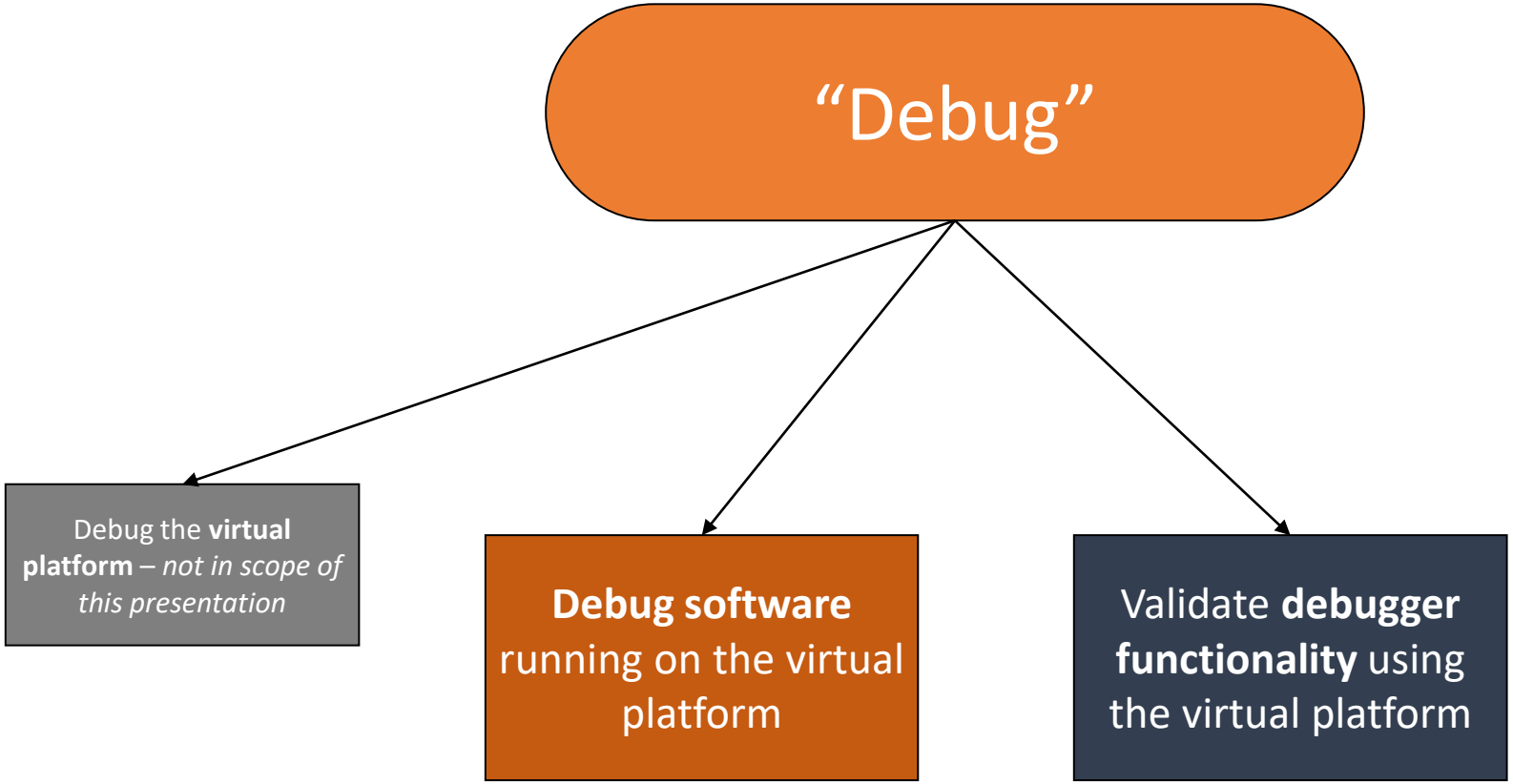
Requirements are being collected… when we get…

“[Please] Add Debug to the Virtual Platform”



- “Debug[ging]” in a virtual platform (VP) can mean different things
  - Debugging the VP itself
  - Debugging software running on the VP
  - Debugging a specific part of the software running on the VP
  - Validating silicon debugging functionality using the VP
  - *… and more …*
- VP builders need to know **what** should be debugged and **how** it is to be debugged
  - Only then can a VP be designed

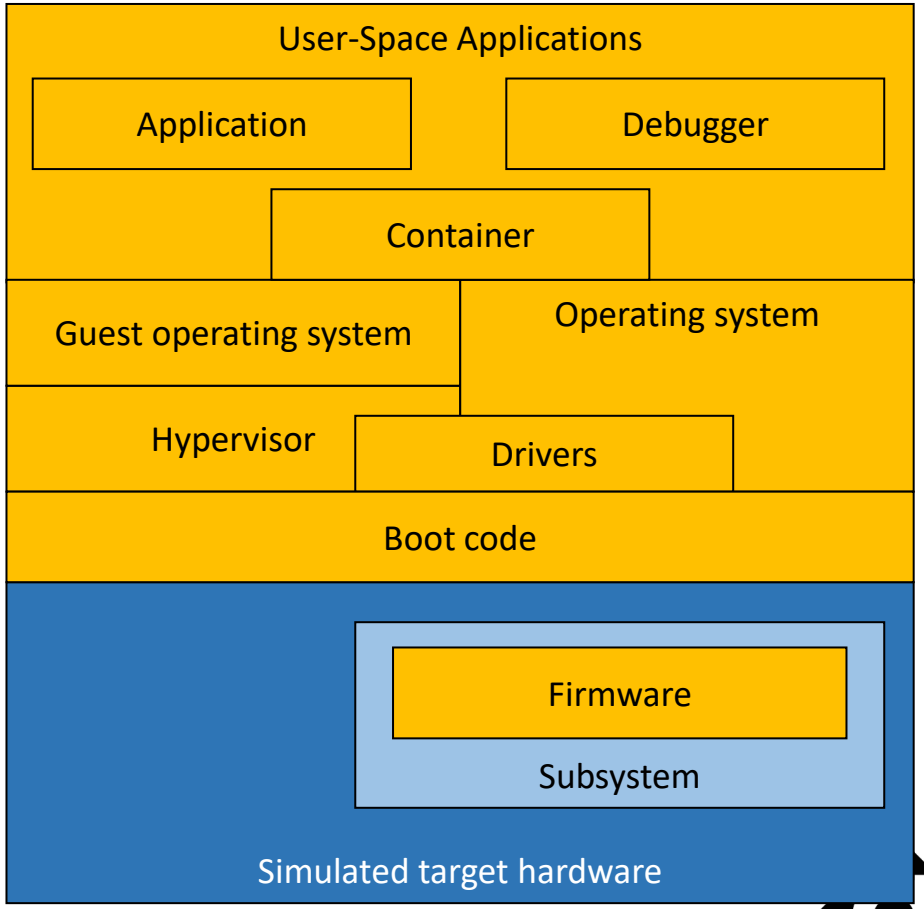
Debug Software or Validate Debug?



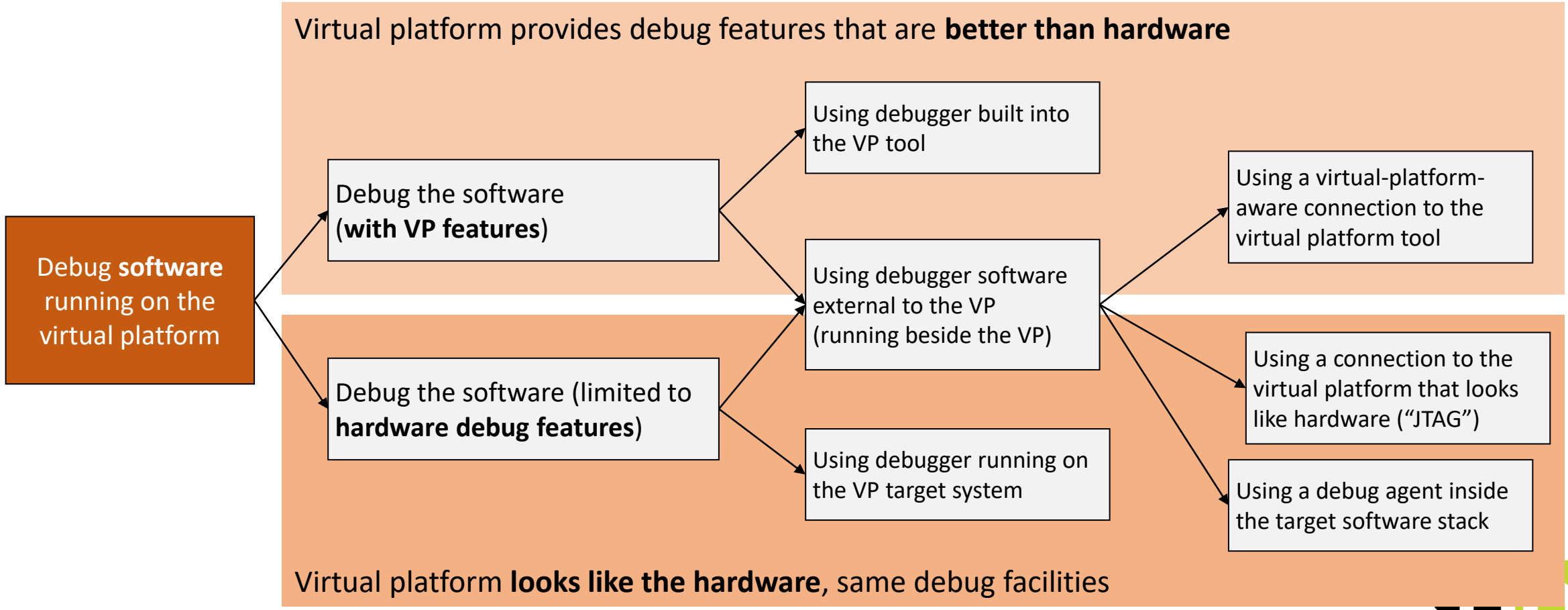
Debugging Software: Which Software?

Debug **software** running on the virtual platform

- Determining the **precise software components** to debug is important
- For example: low-level software is very different from higher-level software
  - Application-level developers expect debuggers to “just work” and not really see the hardware at all
  - Kernel developers need more details and expect to see the hardware
  - IP-block firmware used to very limited visibility into software (in system context)



Debugging Software: VP Implementation Variants

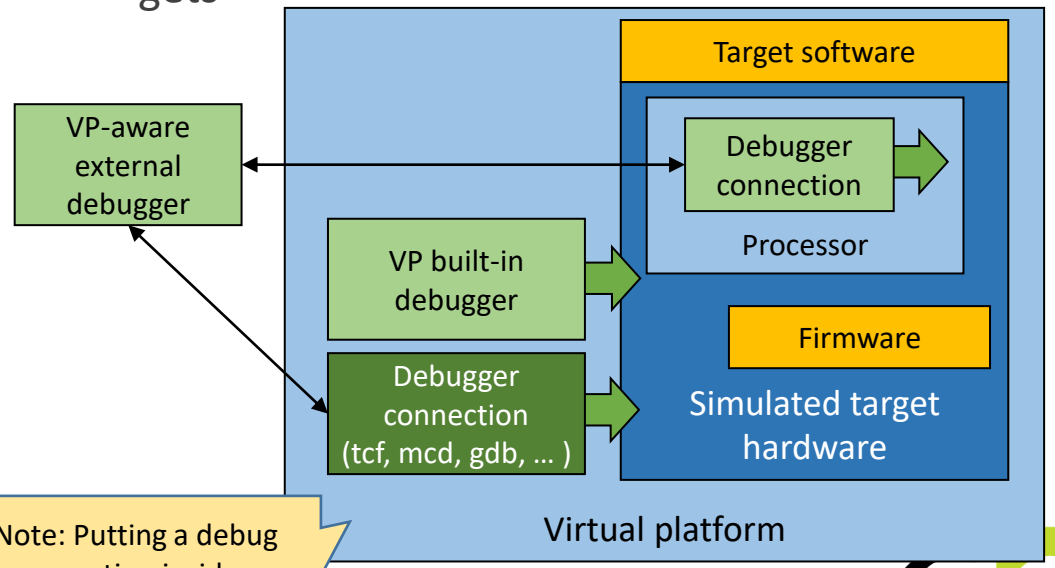


Debugging Software with VP Features

Debug **software** running on the virtual platform

- Debug by **back-door access** to the target system (virtual platform)
- Benefits:
  - Debugger is **invisible to target**
  - Debug a stopped system
  - Single-step anything
  - Debug **any code –** including embedded firmware, boot code, hypervisors, security subsystems, …
  - Use breakpoints unique to VP: exceptions, hardware register access, arbitrary memory, signals…
  - Apply checkpointing, reverse execution, system-level record-replay, …
- External debuggers *can* use VP features
  - With the right connection

- Drawbacks:
  - Operates at hardware level – need special support to identify OS processes, etc.
  - Most external debuggers lack support for the VP debug feature set – "JTAG" is as close as it gets



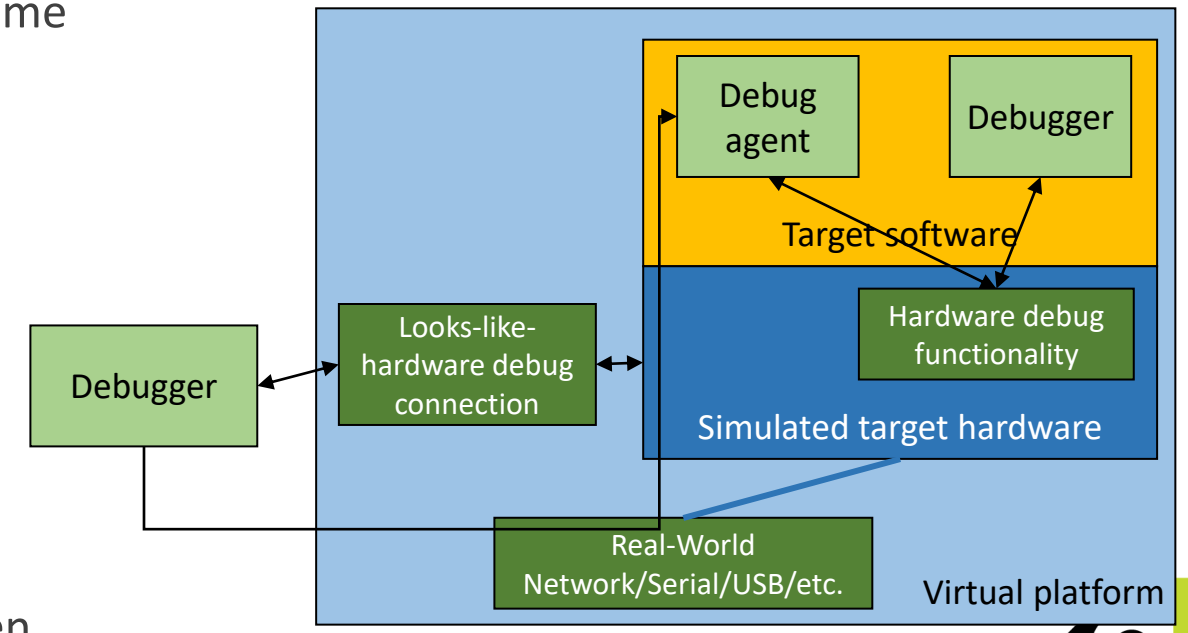
Note: Putting a debug connection inside a processor model is rather suboptimal

Debugging Software Like on Hardware

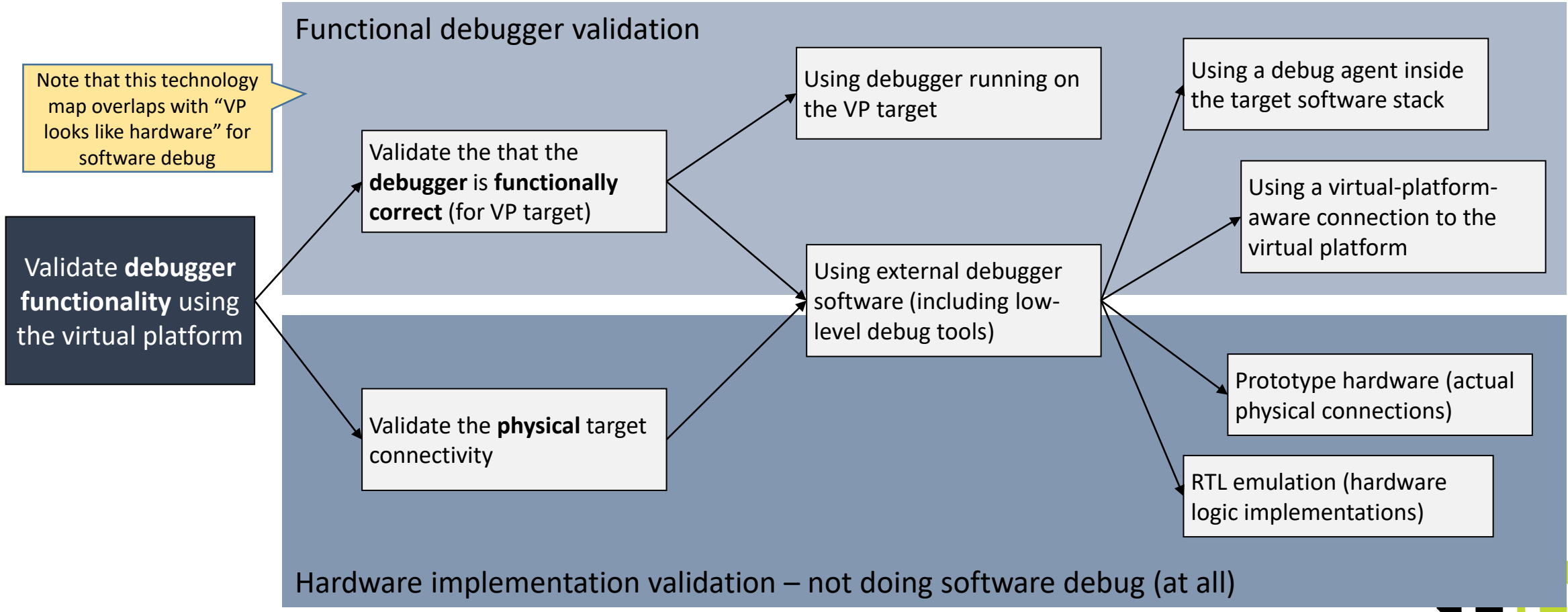
Debug **software** running on the virtual platform

- Software debug works like it does **with a hardware system**
  - The virtual platform looks like a hardware board
  - Debugger talks to platform using the same connections as used with hardware
- Benefits:
  - Existing debugger tools “just work”
  - Seamless transition between VP and hardware
- Drawbacks:
  - VP must have a working connection
  - Cannot take advantage of VP features
  - Difficult to debug low-level code and firmware
  - VP must be running for debug to happen

- Modeling required:
  - Hardware debug facilities as seen from software
  - Hardware connectivity interfaces



Validating a Debugger and its Connections

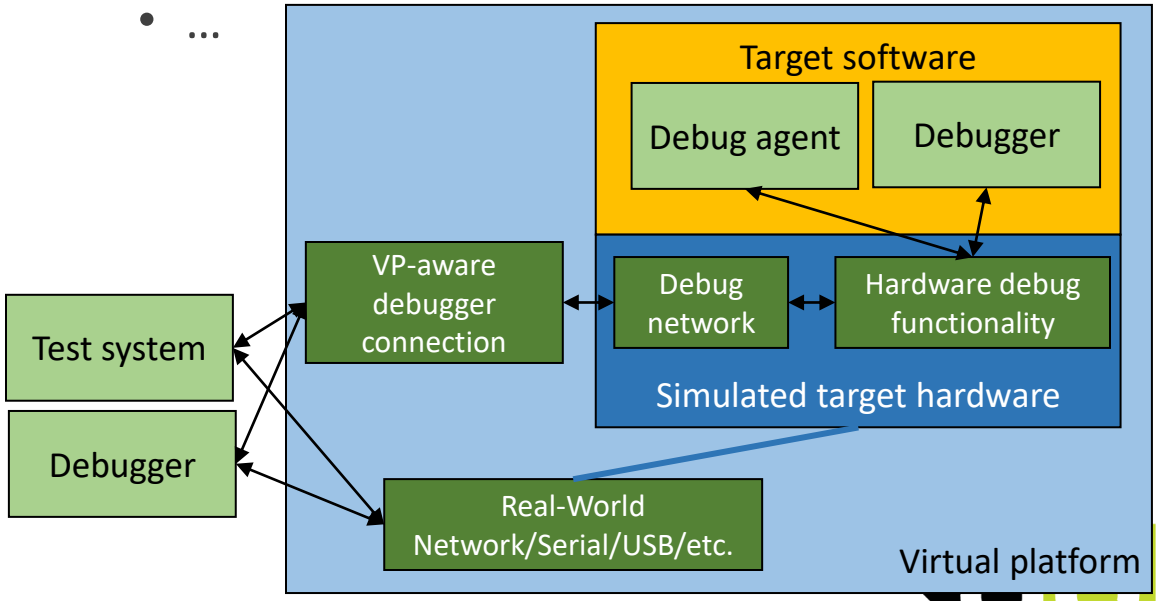


Functional Debugger Validation

Validate **debugger functionality** using the virtual platform

- **Validate the debugger functionality** for a (new) hardware platform
  - Goal: ensure that the debugger works when the hardware arrives
  - On-chip debug functionality can be very complex
- Examples of debug functionality to validate on the VP:
  - Software-facing debug features
  - Tracing configuration and results
  - Debug connection set up
  - Debug network addresses
  - Hardware debug commands
  - …

- Modeling:
  - Hardware debug functionality
  - On-chip debug network
  - Controllers for external connectivity
  - …



Debugger Validation vs Software Debugging

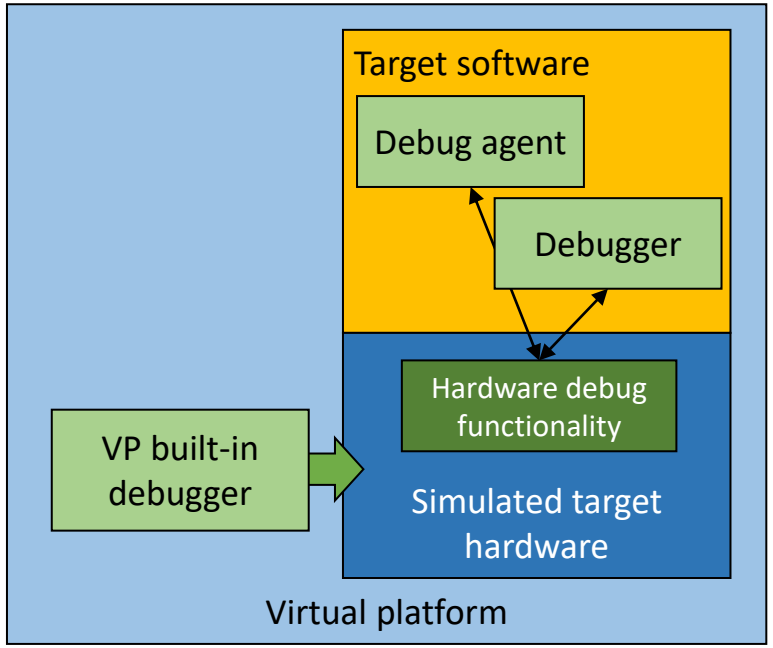
Debug **software** running on the virtual platform

Validate **debugger functionality** using the virtual platform

- Debugger validation and software debugging are **superficially** similar
- Optimal virtual platform implementation is usually very different
- Software debugging is best done using virtual platform features
  - Using VP back doors, breakpoints, tracing, instrumentation, …
  - (Most) debug functionality independent of the virtual platform model
- Debugger validation requires modeling hardware debug features
  - Simulating hardware debug features in a VP is often expensive
  - Requires simulation semantics closer to the hardware
  - Difficult to simulate at full speed

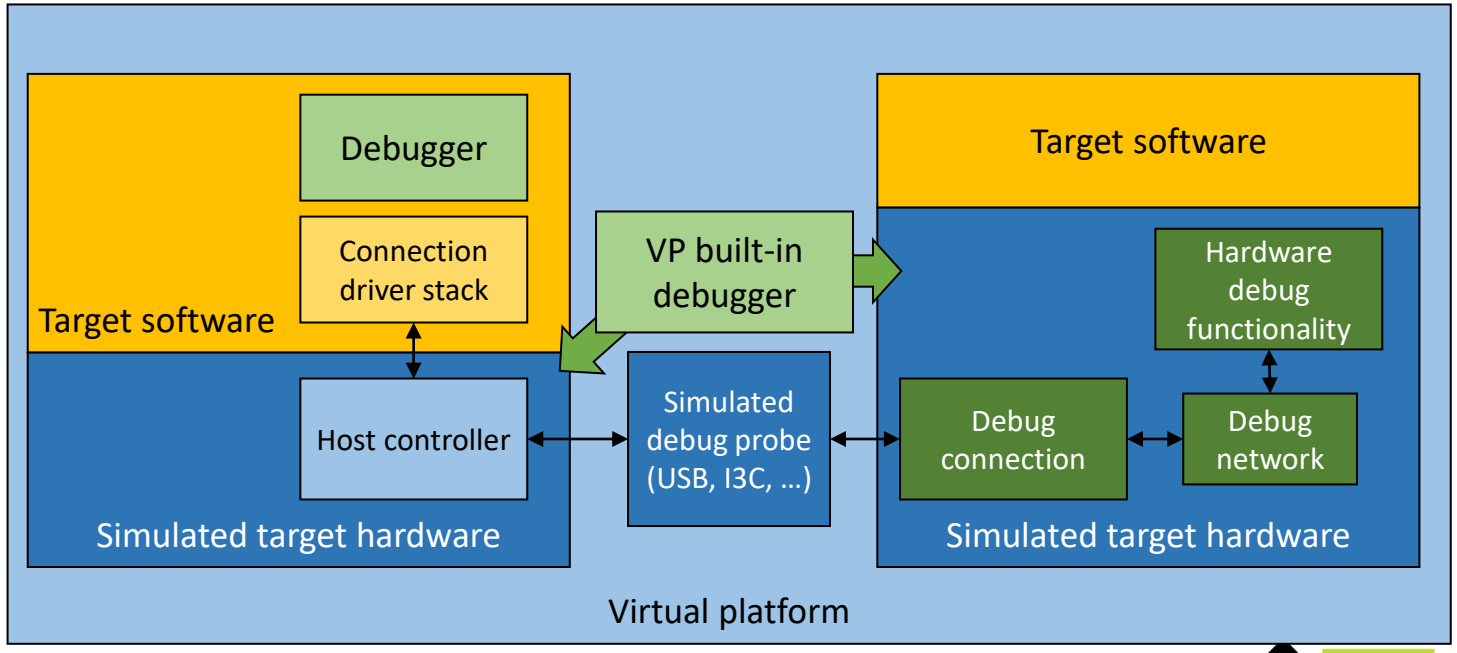
Example: instruction tracing… implementing the semantics of actual tracing hardware is much more complex than just tracing the instructions in the VP

Mixing Up the Layers and Tasks



Debugging how a debugger uses hardware debug features using the VP built-in debugger

Validating the debugger and the supporting connection driver stack by simulating the debug host & debug target



Summary

- “Debug” in a virtual platform context can mean different things
- Important to distinguish between distinct use cases:
  - **Debugging software** running on the virtual platform target system
  - **Validating debugger** implementations and connection to the target system
  - (And **debugging the virtual platform** itself)
- The implementation will vary with the use case(s):
  - Software debug does not require modeling hardware debug features
  - Real-world connectivity
  - Debugger validation requires deep modeling of debug features
  - In some cases, one implementation can solve multiple use cases