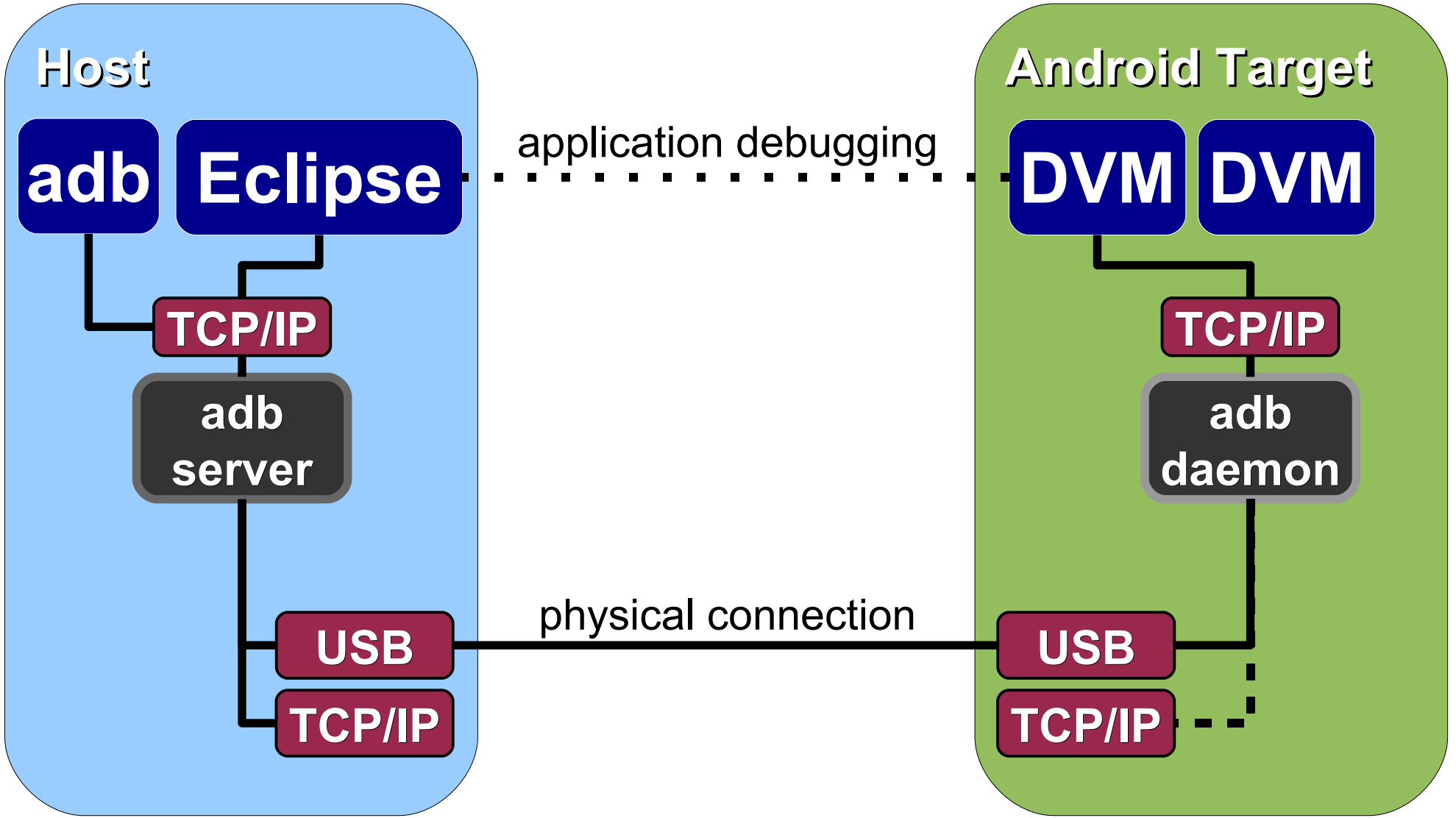
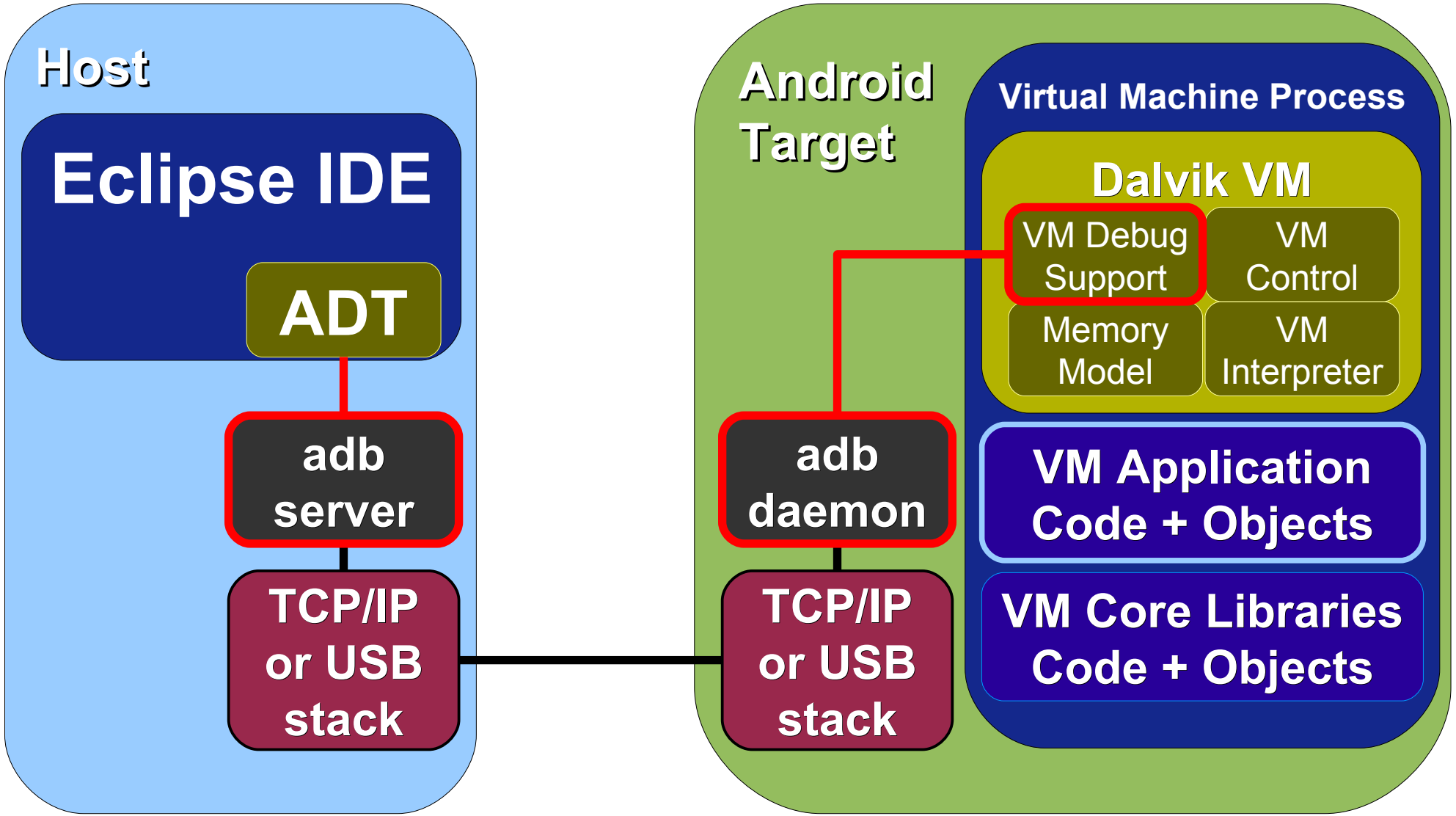
VM Application Debugging via JTAG: Android TRACE32 JTAG Debug Bridge

- **ADB Architecture**
- **Stop-Mode implications for ADB**
- **JTAG Transport**
- **Outlook**

VM Application Debugging with ADB



Application Debugging (Run-Mode)



Architecture

system/core/adb/OVERVIEW.TXT

[...] As a whole, everything works through the following components:

1. The ADB server [...] runs on the host [and] is really one giant multiplexing loop whose purpose is to orchestrate the exchange of data (packets, really) between clients, services and devices.
2. The ADB daemon (adbd) [runs on a] device or emulated system [and connects] to the ADB server (through USB for devices, through TCP for emulators) and provide[s] a few services for clients that run on the host.
3. The ADB command-line client [...] is used to run adb commands from a shell or a script. It first tries to locate the ADB server on the host machine, and will start one automatically if none is found.

Currently, a single 'adb' binary is used for both the server and client. [This] makes distribution and starting the server easier. [...]

Architecture

system/core/adb/protocol.txt

--- a replacement for aproto ----------

When it comes down to it, aproto's primary purpose is to forward various streams between the host computer and client device (in either direction).

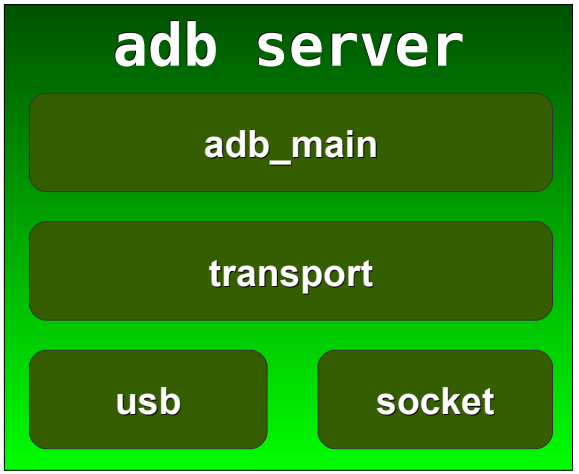
This replacement further simplifies [this concept].

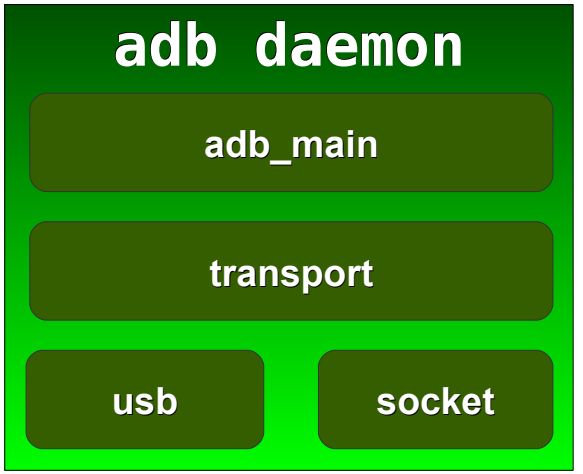
The host side becomes a simple comms bridge with no "UI", which will be used by either commandline or interactive tools to communicate with a device or emulator that is connected to the bridge.

The protocol is designed to be straightforward and well-defined [...].

The protocol discards the layering aproto has and should allow the implementation to be much more robust. [...]

Communication Architecture





Different transport methods active at the same time:

- usb_init()
- local_init()

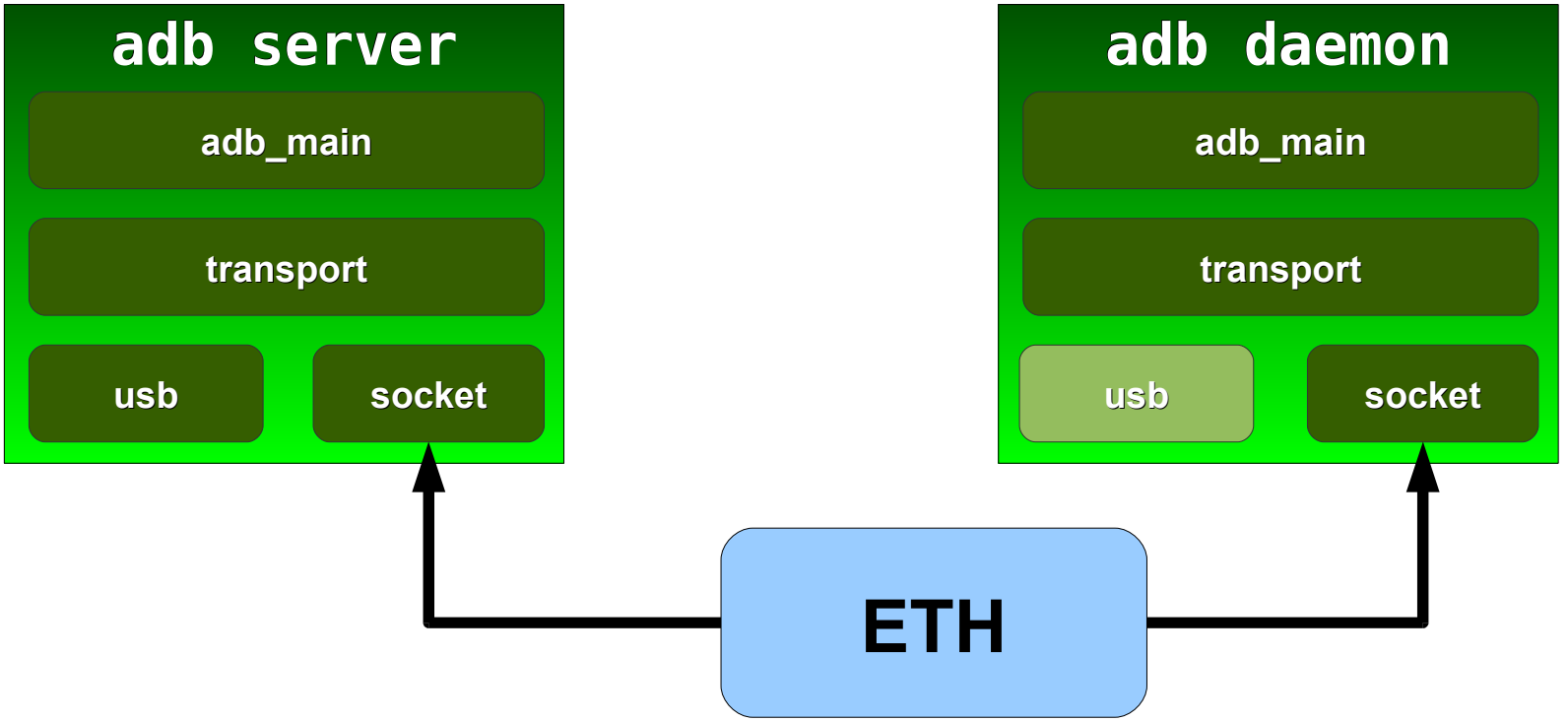
Select transport at startup:

IF property(service.adb.tcp.port)
- listen at tcp:*port* ■

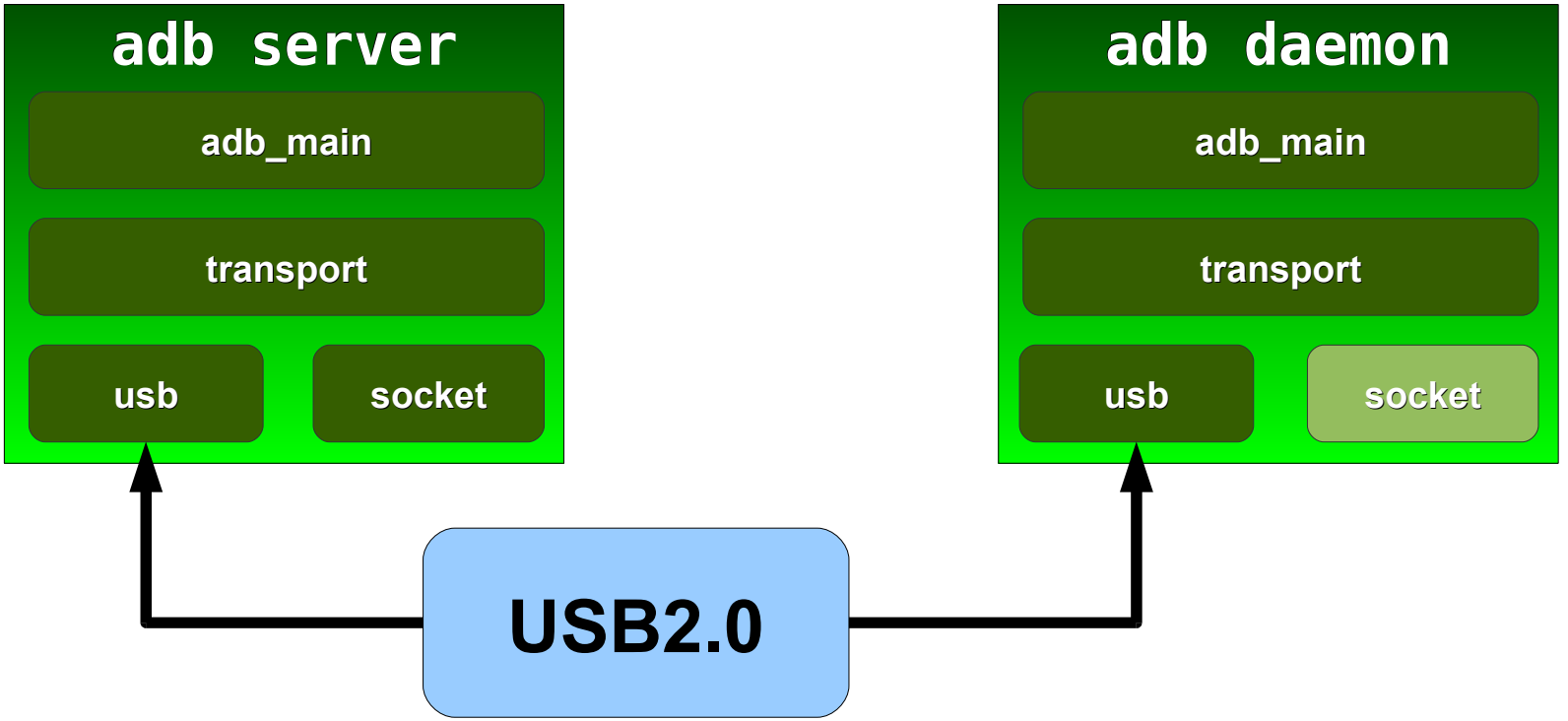
IF access(/dev/android_adb)
- listen at USB gadget device ■

DEFAULT
- listen at tcp:5037

Case A: Using Sockets



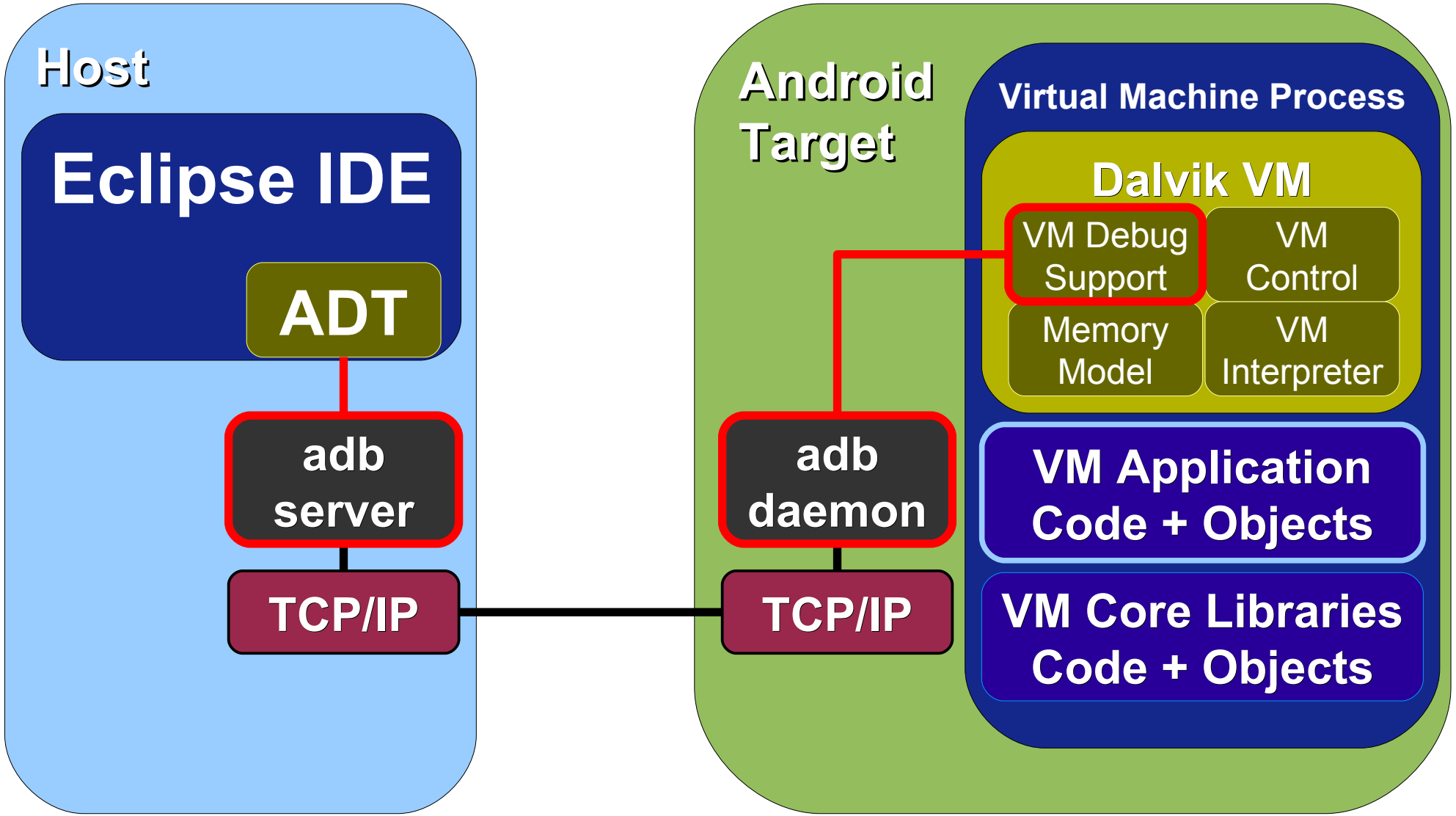
Case B: Using USB



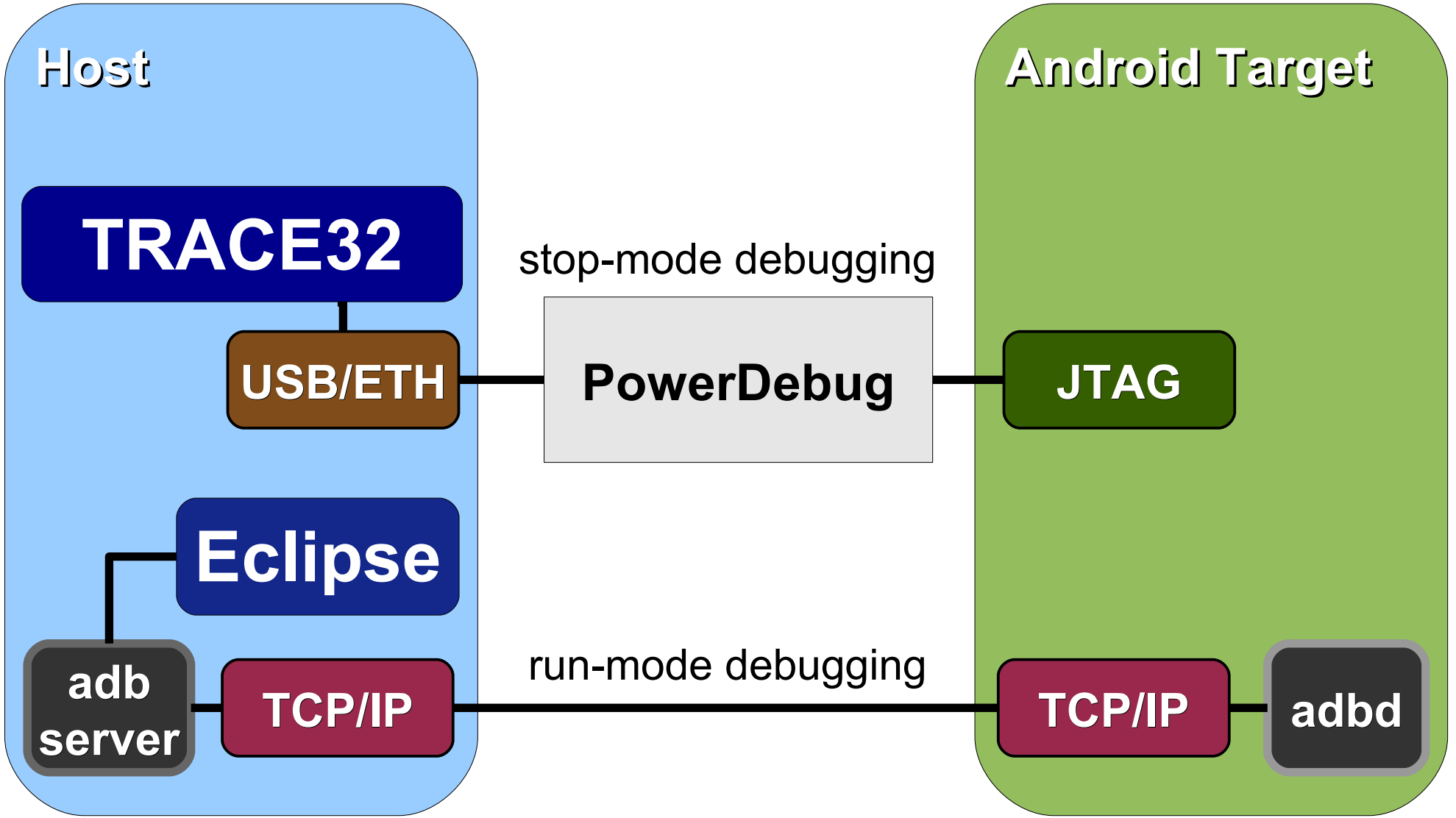
VM Application Debugging via JTAG: Android TRACE32 JTAG Debug Bridge

- ADB Architecture
- **Stop-Mode implications for ADB**
- JTAG Transport
- Outlook

Application Debugging via TCP/IP



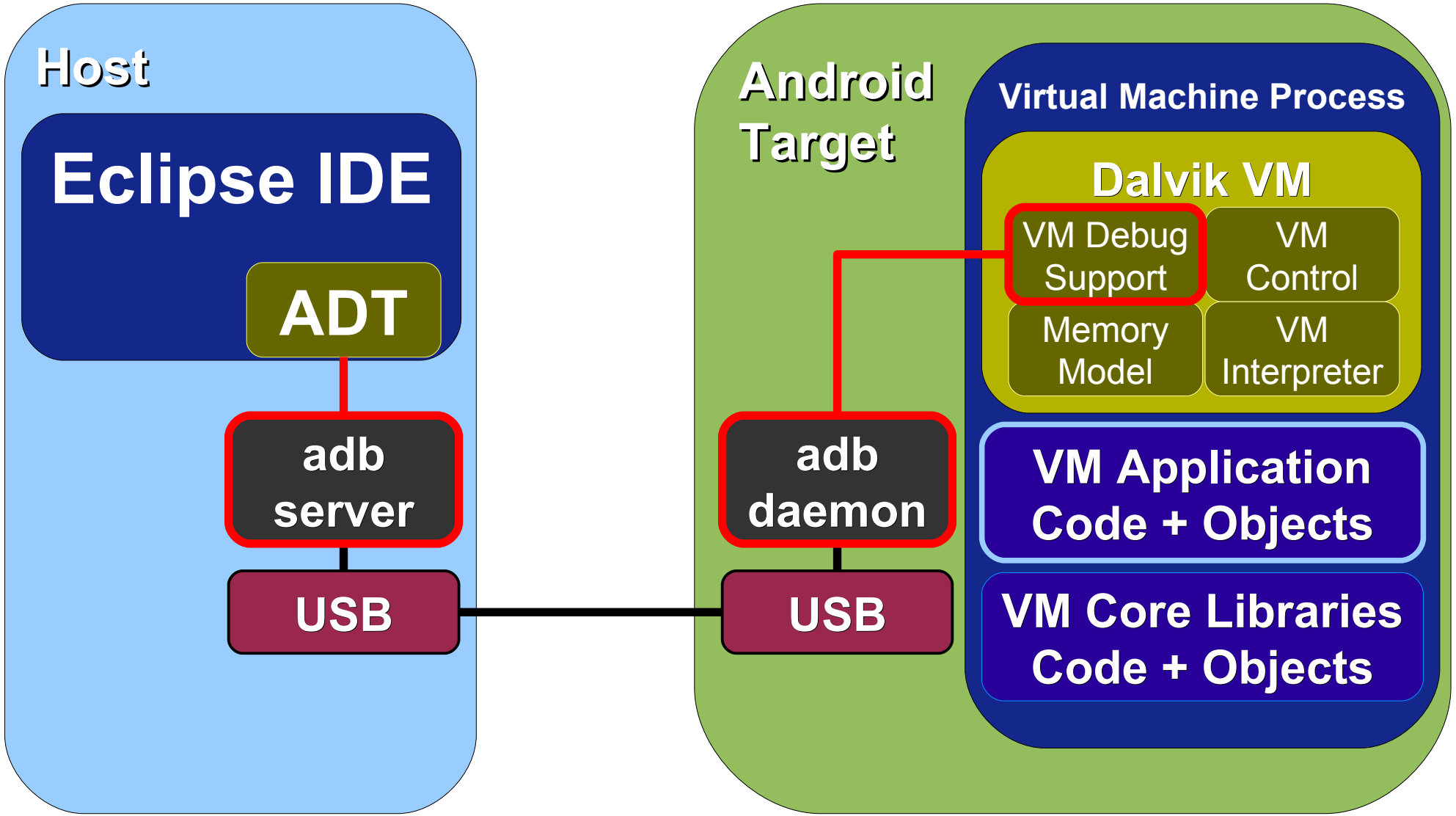
App Debug via TCP/IP (Run-Mode) + JTAG Debug (Stop-Mode)



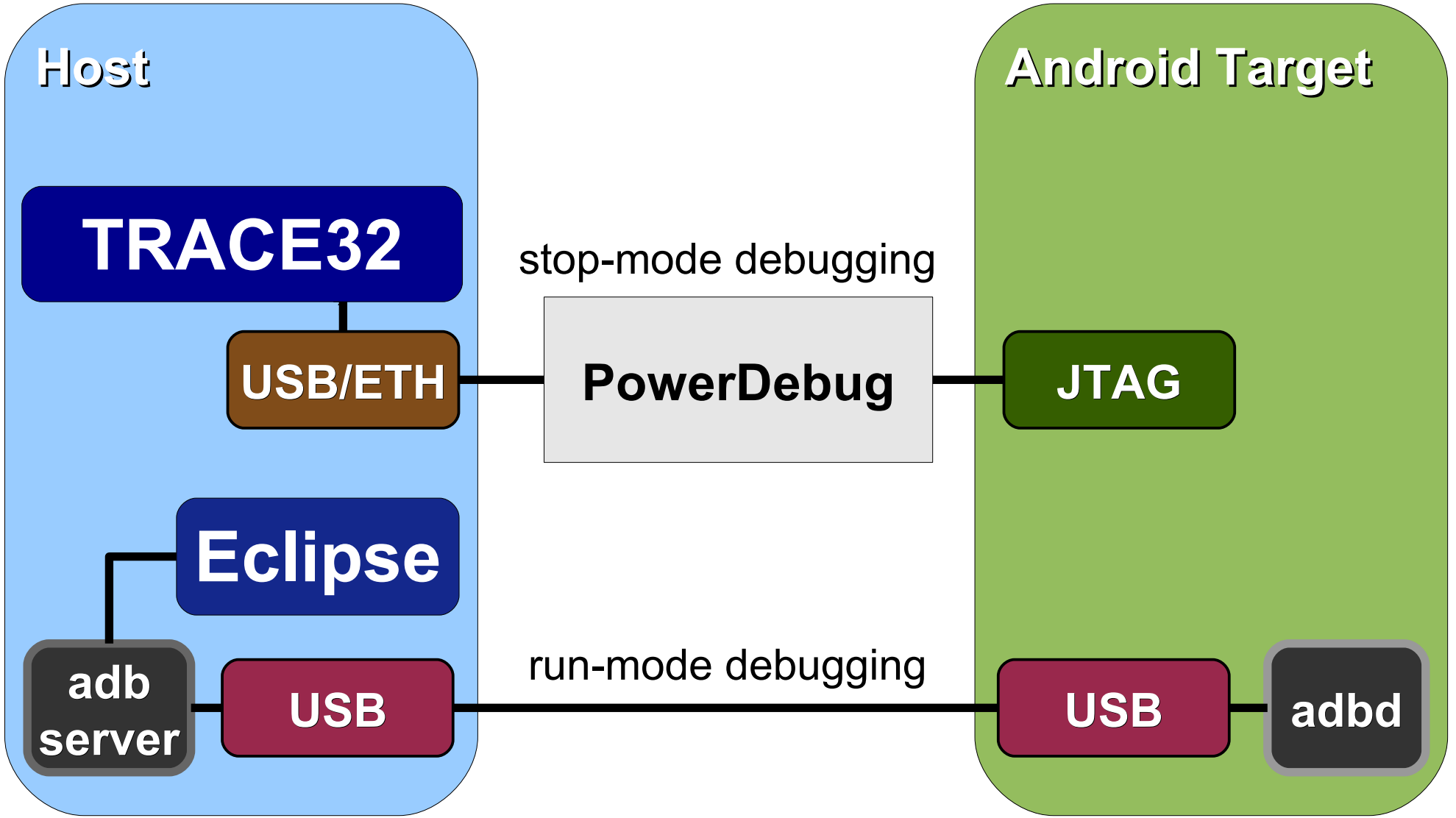
Stop-Mode Implications for TCP/IP

- IP was designed for robustness
- Host does not detect device presence
- Communication can be resumed
- **Eclipse can time out**
  but often an ADT reconnect is possible

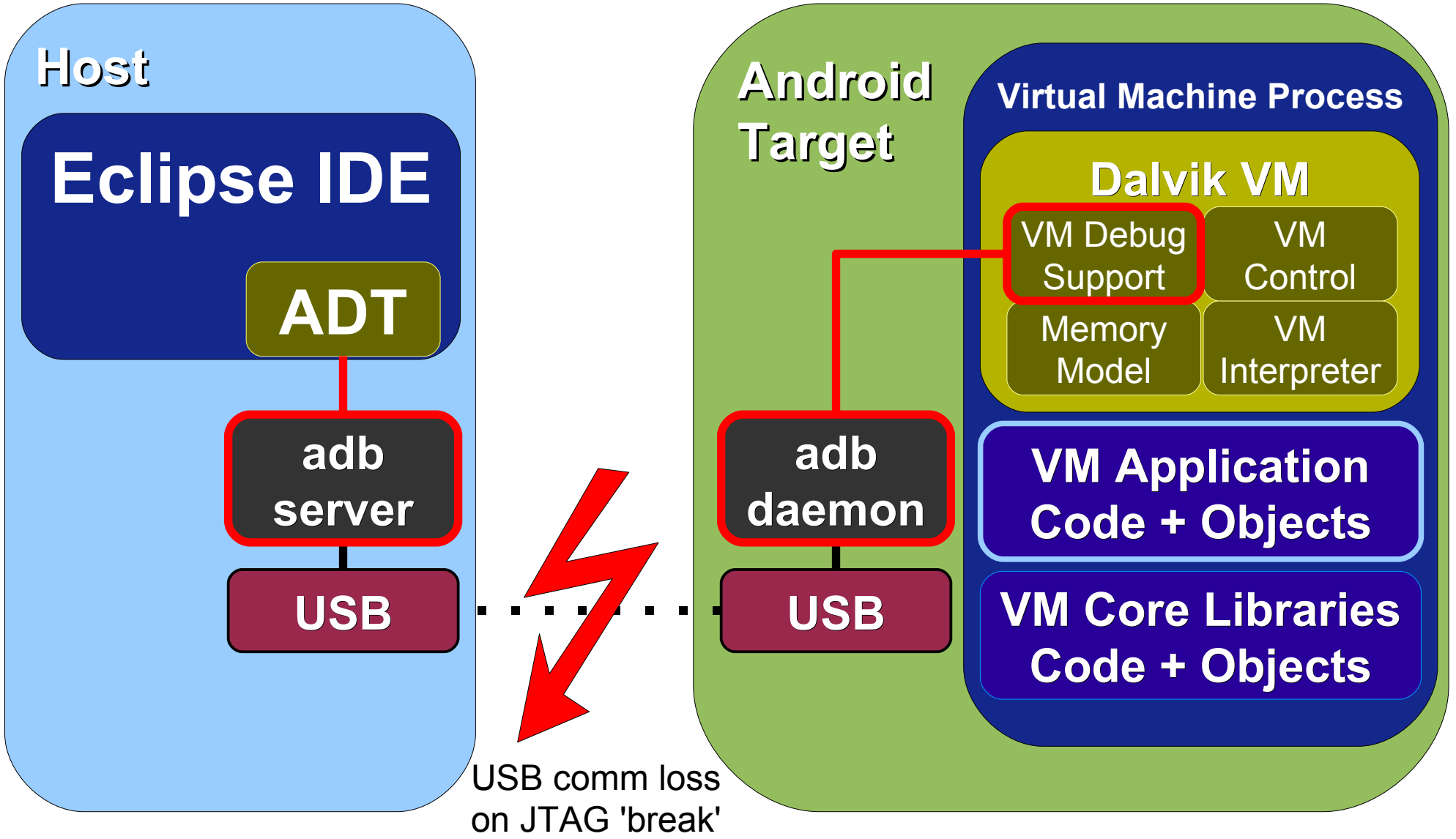
Application Debugging via USB



App Debug via USB (Run-Mode) + JTAG Debug (Stop-Mode)



JTAG 'Stop' ►►► loss of USB communication



Stop-Mode Implications for USB

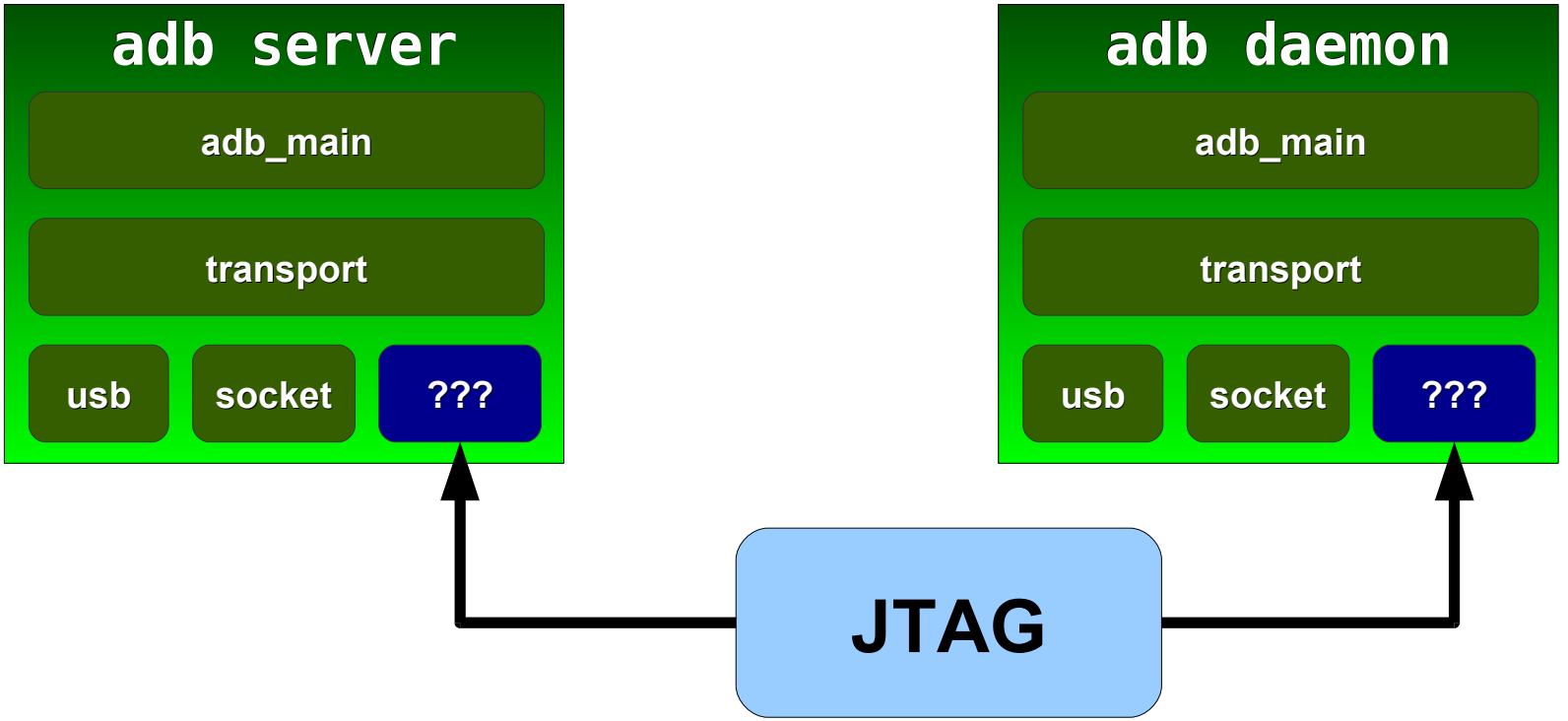
- Host periodically polls USB devices
- Host detects device presence
- ADB 'kicks' disconnected devices
- Host reconnect possible only with USB plug-in event
- No on-the-fly device reconnect mechanism provided
- Multiple reconnects per second can be dangerous
- **USB Communication Breakdown**

VM Application Debugging via JTAG: Android TRACE32 JTAG Debug Bridge

- ADB Architecture
- Stop-Mode implications for ADB
- ▶ JTAG Transport
- Outlook

LAUTERBACH DEVELOPMENT TOOLS

Introduce Case C: Using JTAG



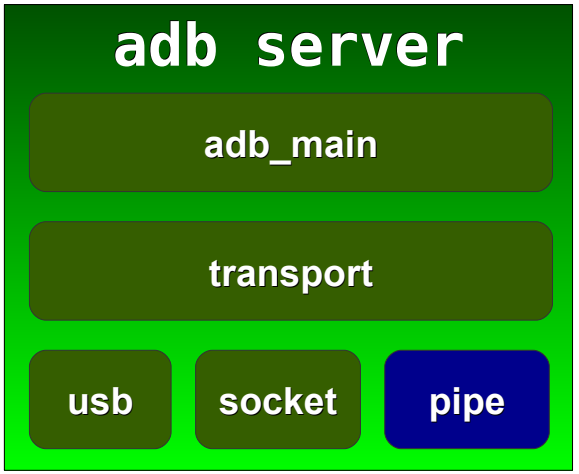
Is this possible?

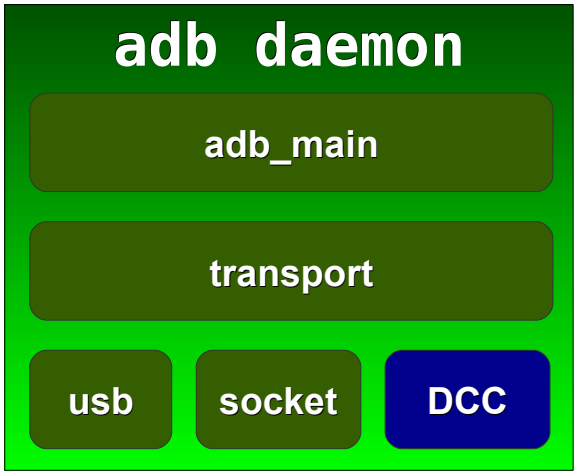
LAUTERBACH DEVELOPMENT TOOLS

“TRACE32 JTAG Bridge” Ideas:

- ARM9|10|11 has “Debug Communication Channel” - a DCC register (pair) that debugger can r/w via JTAG
- TRACE32 “Fast Data eXchange” (FDX) can use DCC
- TRACE32 FDX client can send/receive data via “TRACE32 RemoteAPI” or via “named pipes”
- Try to keep changes limited to system/core/adb
- Must be possible on Linux + Windows hosts
- Must work on SMP systems
- Use as working code reference

Communication Architecture





On the host:

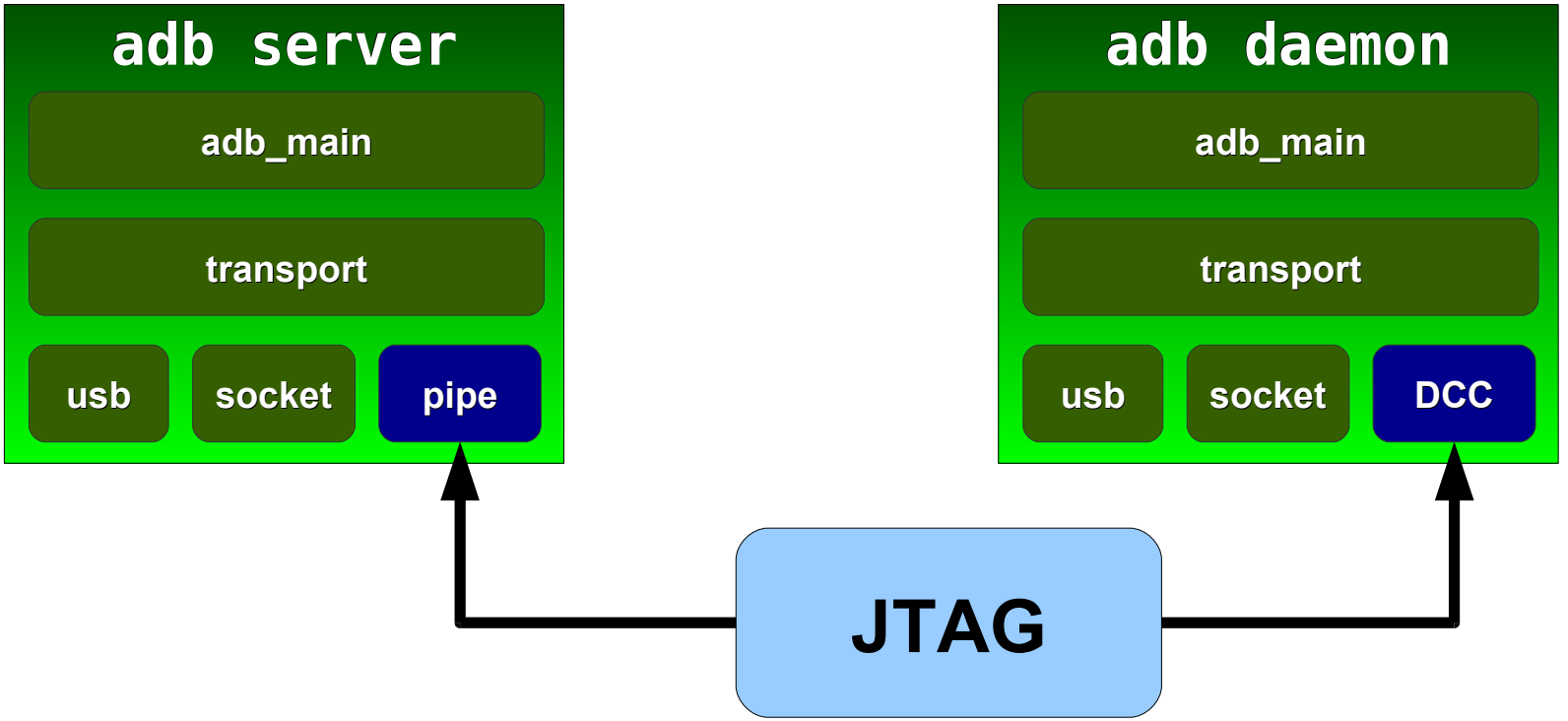
- TRACE32 FDX using JTAG (named pipe) transport

On the target:

- TRACE32 FDX using JTAG (ARM DCC) transport

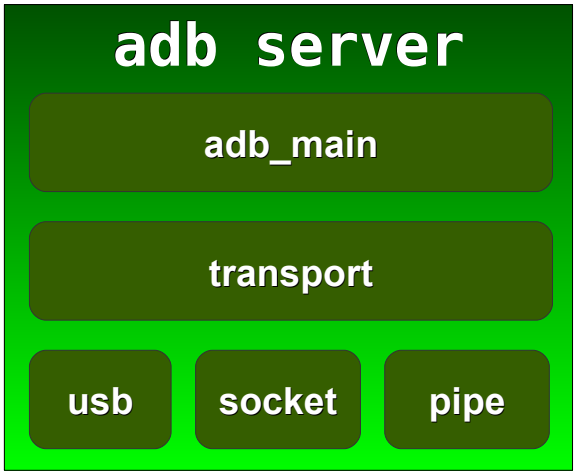
LAUTERBACH DEVELOPMENT TOOLS

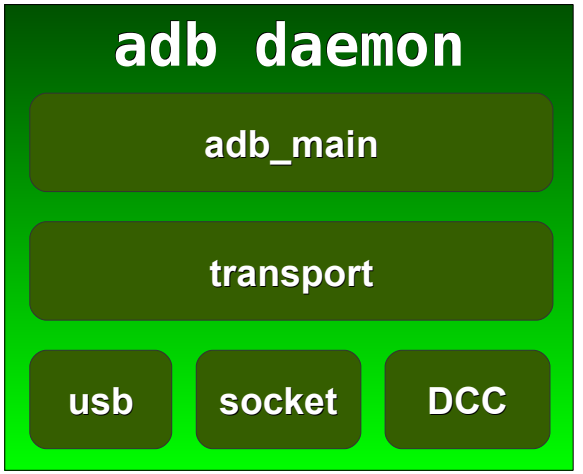
Introduce Case C: Using JTAG



This is possible!

Communication Architecture – New Version





Different transport methods active at the same time:

- ► jtag_init()
- ► usb_init()
- ► local_init()

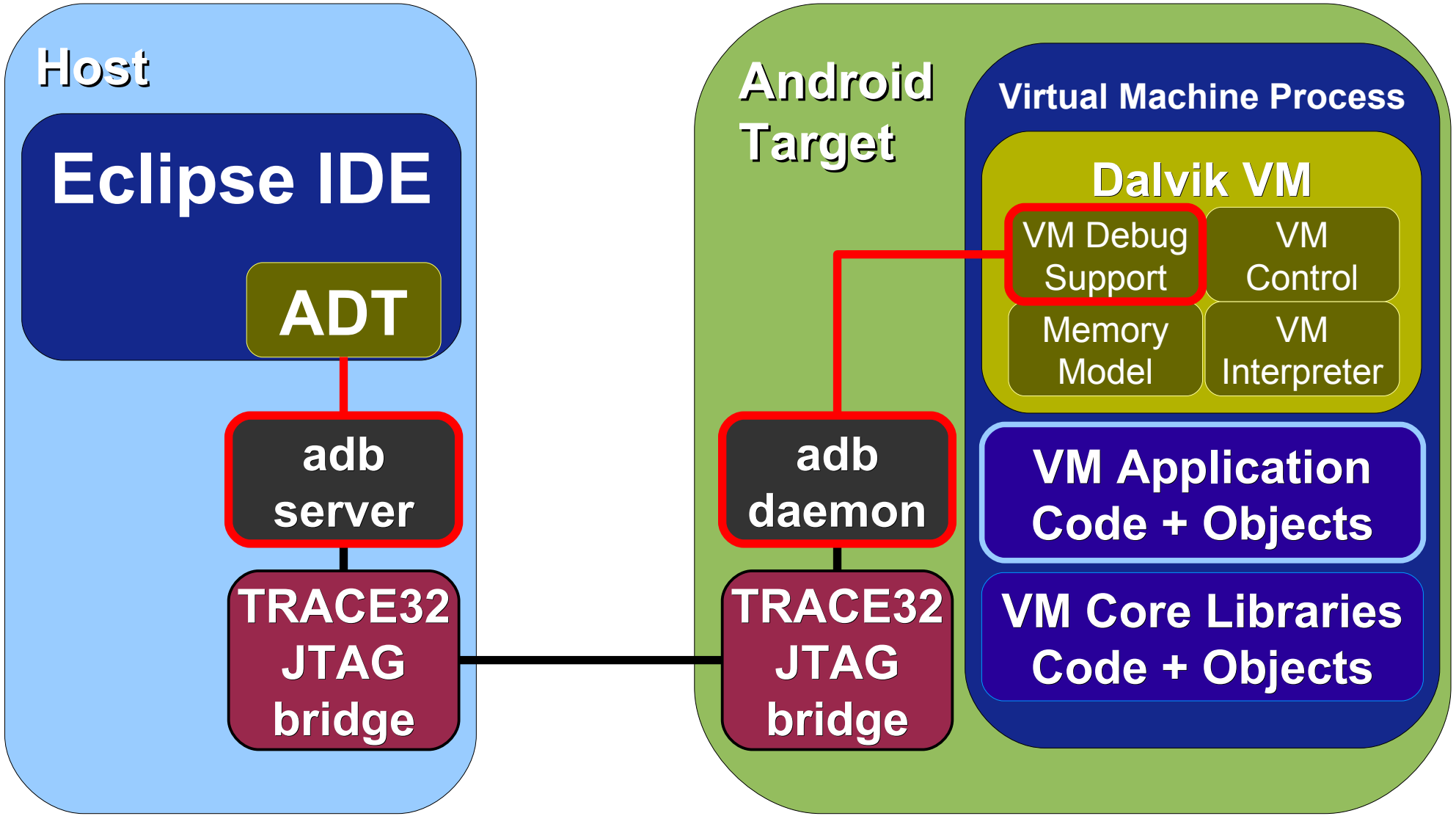
IF property(service.adb.jtag) > 0
► listen at DCC register ■

IF property(service.adb.tcp.port)
► listen at tcp:*port* ■

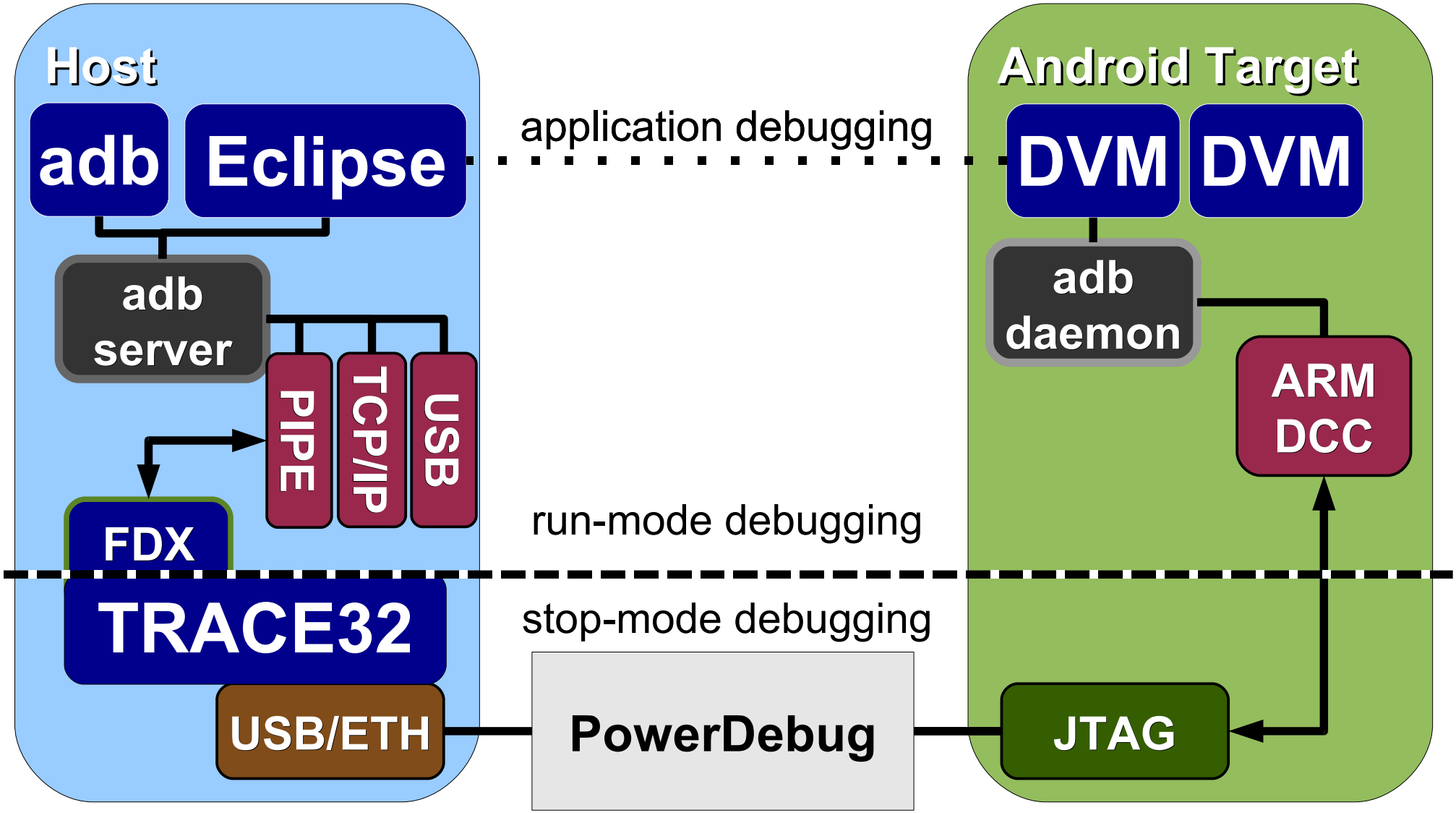
IF access(/dev/android_adb)
► listen at USB gadget device ■

DEFAULT ► listen at tcp:5037

Assisted Run-Mode Debugging via JTAG bridge



Application Debugging via TRACE32 JTAG Bridge



TRACE32 JTAG Bridge – Initial Implementation

- git patch for adb server (host) and adbd (target)
- adds JTAG “transport” to adb (Run-Mode), tested on FroYo
- TRACE32 FDX in ARM9|10|11 DCC4D mode for adbd data
- TRACE32 FDX in “Named Pipe” mode for adb server data
- tested on Linux + Windows hosts and on SMP target
- on/off switch for adbd JTAG bridge mode (setprop, adbd restart)
- not very fast, but only alternative if ETH or USB are not available
- working reference code + open source: customers and 3rd parties can adapt the patch, e.g. for non-ARM target platforms

TRACE32 JTAG Bridge Patch – Timeline

- 2011-04-08 – initial Linux version (tested on MEP6410)
- 2011-05-24 – reworked pipe system to support Windows
- 2011-06-10 – SMP support (tested on PandaBoard)

Available Today!

VM Application Debugging via JTAG: Android TRACE32 JTAG Debug Bridge

- ADB Architecture
- Stop-Mode implications for ADB
- JTAG Transport
- Outlook

LAUTERBACH DEVELOPMENT TOOLS

TRACE32 JTAG Bridge – What Can Be Done?

- introduce Linux Kernel “JTAG Communication Device”
  - encapsulate target-specific communciation code, different platforms expose the same JTAG interface
  - could provide multiple JTAG communication channels for Terminal, System Trace, ADB and Customer Use
- test on more target platforms
- make adbd transport channel easily runtime-switchable
- git-push code changes to Android and Linux Kernel repos to improve JTAG debugging + transport awareness
- profile TRACE32 FDX pipe connection, maybe add TCP/IP mode

TRACE32 JTAG Bridge – Who Can Do It?

- introduce Linux Kernel “JTAG Communication Device”
  - encapsulate target-specific communciation code, different platforms expose the same JTAG interface
  - could provide multiple JTAG communication channels for Terminal, System Trace, ADB and Customer Use
- test on more target platforms
- make adbd transport channel easily runtime-switchable
- git-push code changes to Android and Linux Kernel repos to improve JTAG debugging + transport awareness
  ►►► everyone can do this (Customer, 3rd Party, You!)
- profile TRACE32 FDX pipe connection, maybe add TCP/IP mode
  ►►► only possible for Lauterbach

Thank You!

Hagen Patzke

hagen.patzke@lauterbach.com

http://www.lauterbach.com/vmandroid.html