

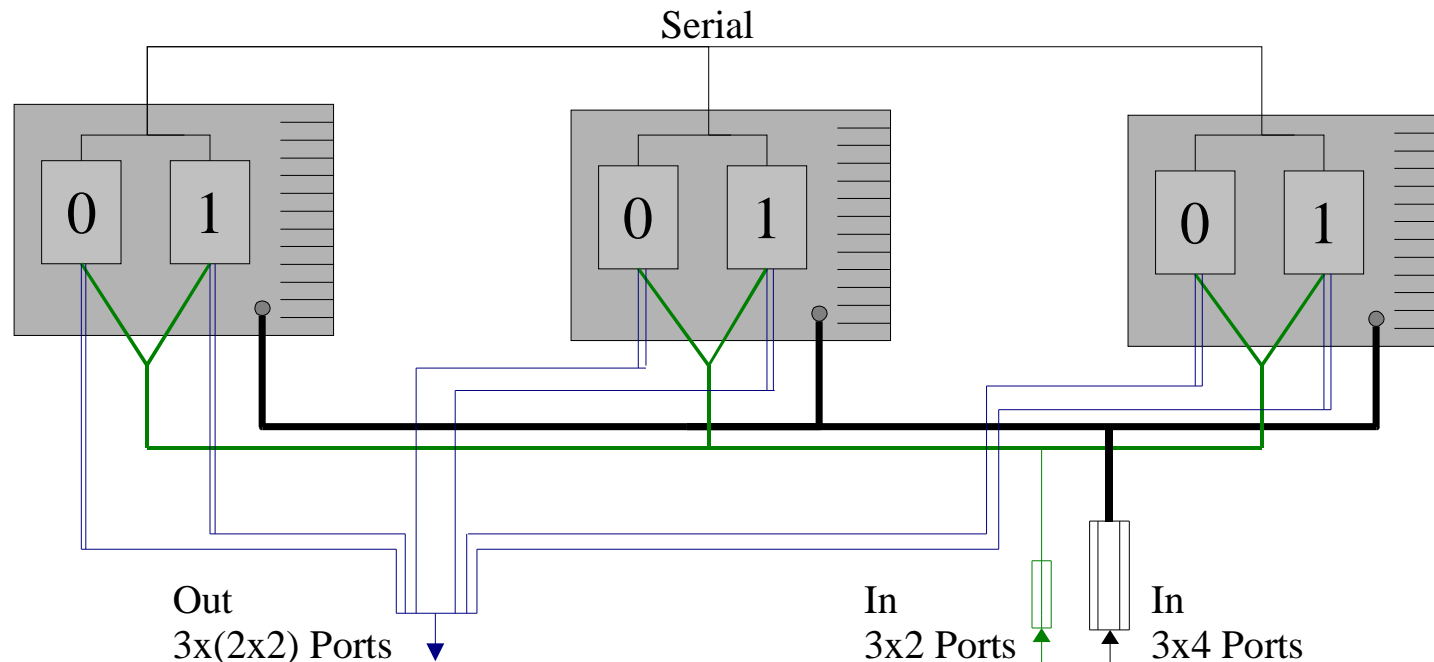
Automatische Testumgebungsgenerierung und Ausführung für Black-Box und Grey-Box-Tests in verteilten, heterogenen Echtzeit-Systemen

**Konrad Betz, Dr. Peter Biechele
beXtec GmbH
Emmendingen**

Inhalt

- Problemstellung:
Typische verteilte Systeme und System Tests
- Anforderungen an Lösung und Begriffsdefinitionen
- Lösung:
Automatisierung der System-Tests
- Vorteile der Methode
- Zusammenfassung und Ausblick

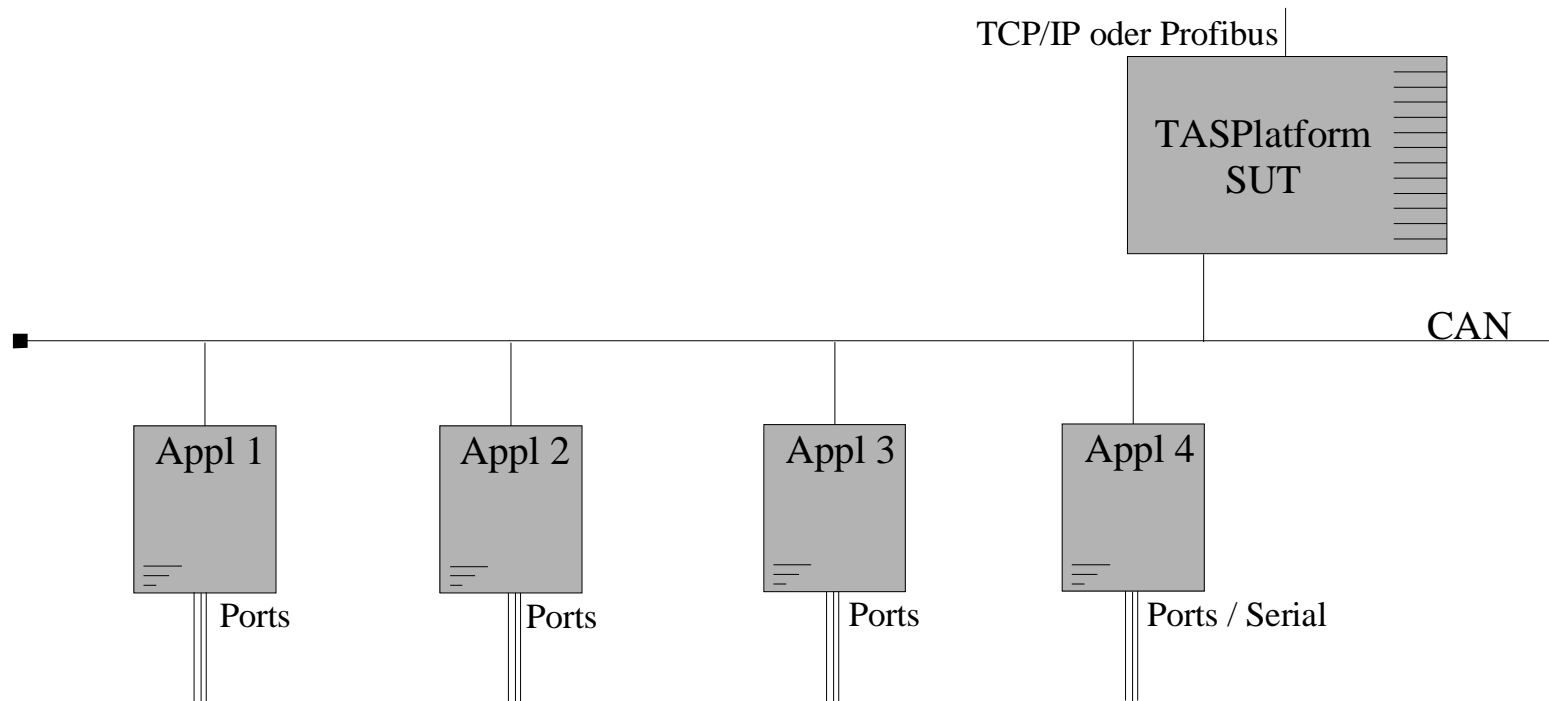
Beispiel SUT



Nötige Testumgebung:

Hochfrequente Signale In/Out
Bus Reader, eventuell Writer

Beispiel SUT



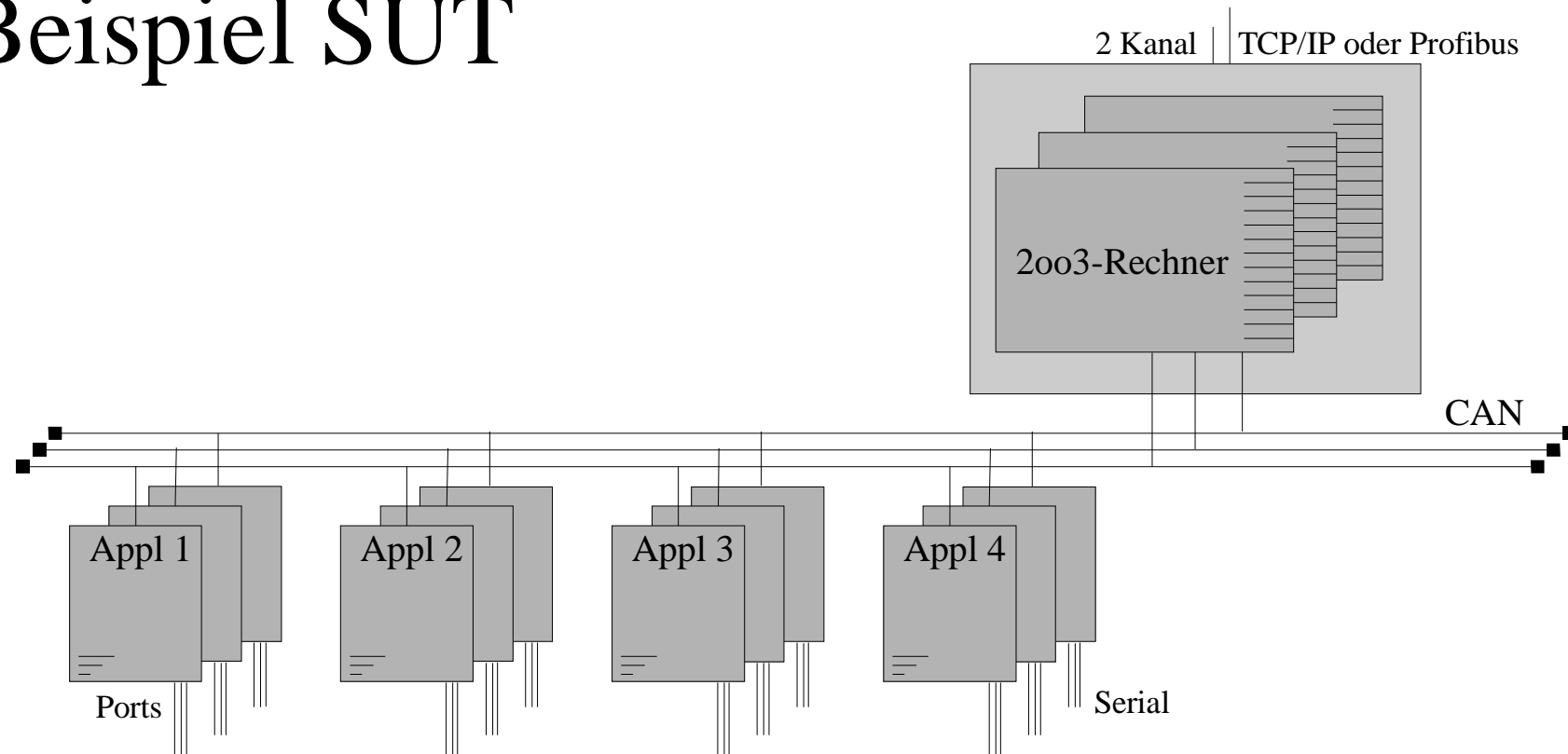
Nötige Testumgebung:

heterogene Sync Medien

System under Test Versions Überprüfungen

aufwändige Test-Umgebung

Beispiel SUT



Nötige Testumgebung:

Echtzeit-Synchronisation der Test-Umgebung
Komplexe Testumgebung, großer Makeprozess
Anfangs- und Endzustand der Testumgebung

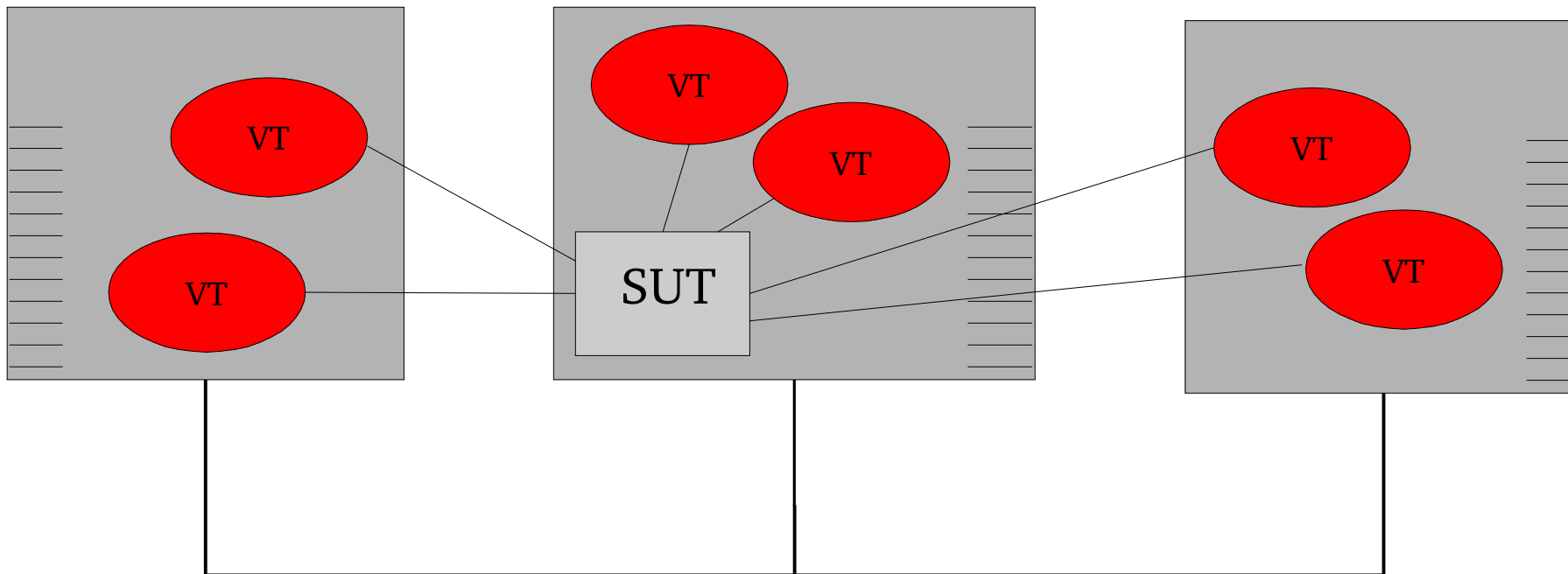
Anforderungen an Methode

- Modulare Tester, die Signale an Ports anlegen können
- Reader/Writer Tester für Bussysteme als Modul
- Heterogene Synchronisationsmöglichkeit der Testumgebung im Mikrosekunden Bereich
- Automatischer System Download und Version Check
- Erstellung (compile) der Testumgebung automatisch
- Definierte Anfangs- und Endzustände der Testumgebung
- Reproduzierbare Tests (System und Testumgebung)
- Bisher: 30-40% der Zeit/Kosten in Erstellung der Testumgebung

Begriffsdefinitionen für System Tests

- **System under Test (SUT)**
- Test-Umgebung --> **Virtueller Tester (VT)**
"Eine Instanz, welche die Instrumentierung/Stimulation des SUT übernimmt" (Diagnose-VT / Stimulator-VT)
- Schnittstellen VT - SUT beliebig (meist Msg. basiert)
- Moskitos (SUT spezifische Hilfsprogramme)
- Echt verteilte Synchronisation, dadurch keine Agenten / Supervisor notwendig
- **Synchronisationsmedium** ist beliebig, auch inhomogen (Portleitungen, CAN, TCP/IP, UDP/IP, etc.)

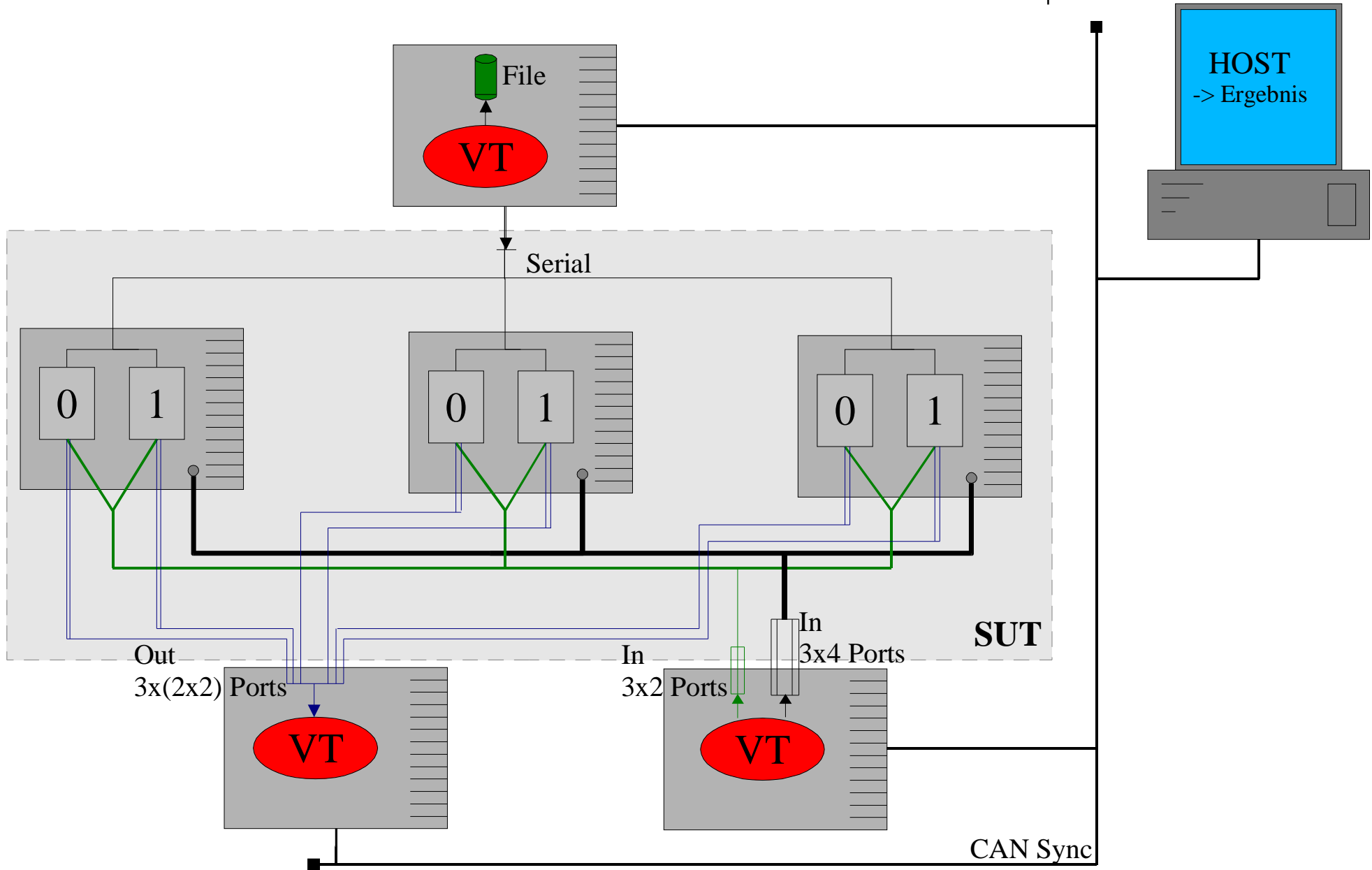
Struktur von System Tests der beXtec GmbH

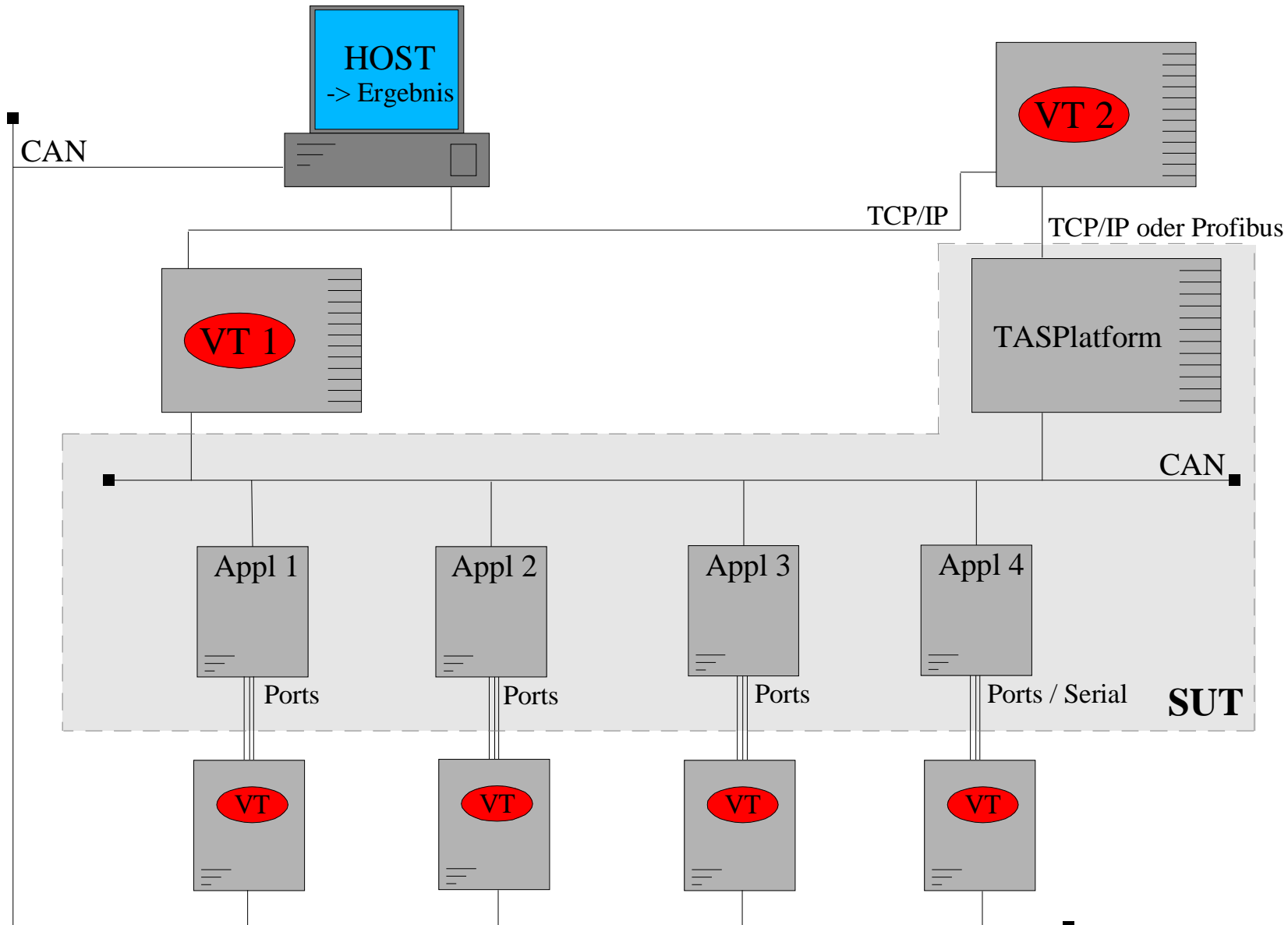


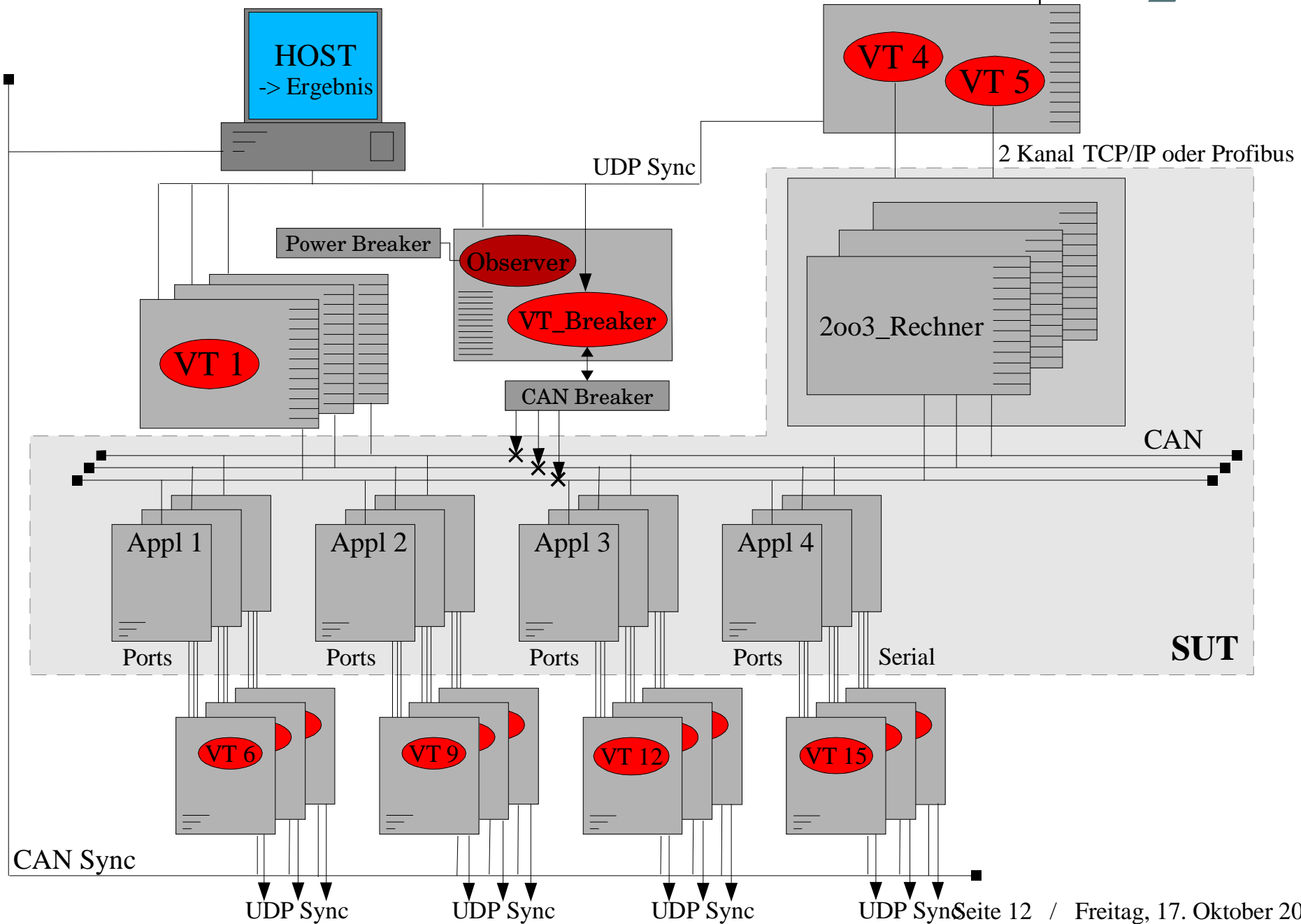
z.B. einen CAN-Bus als Synchronisationsmedium

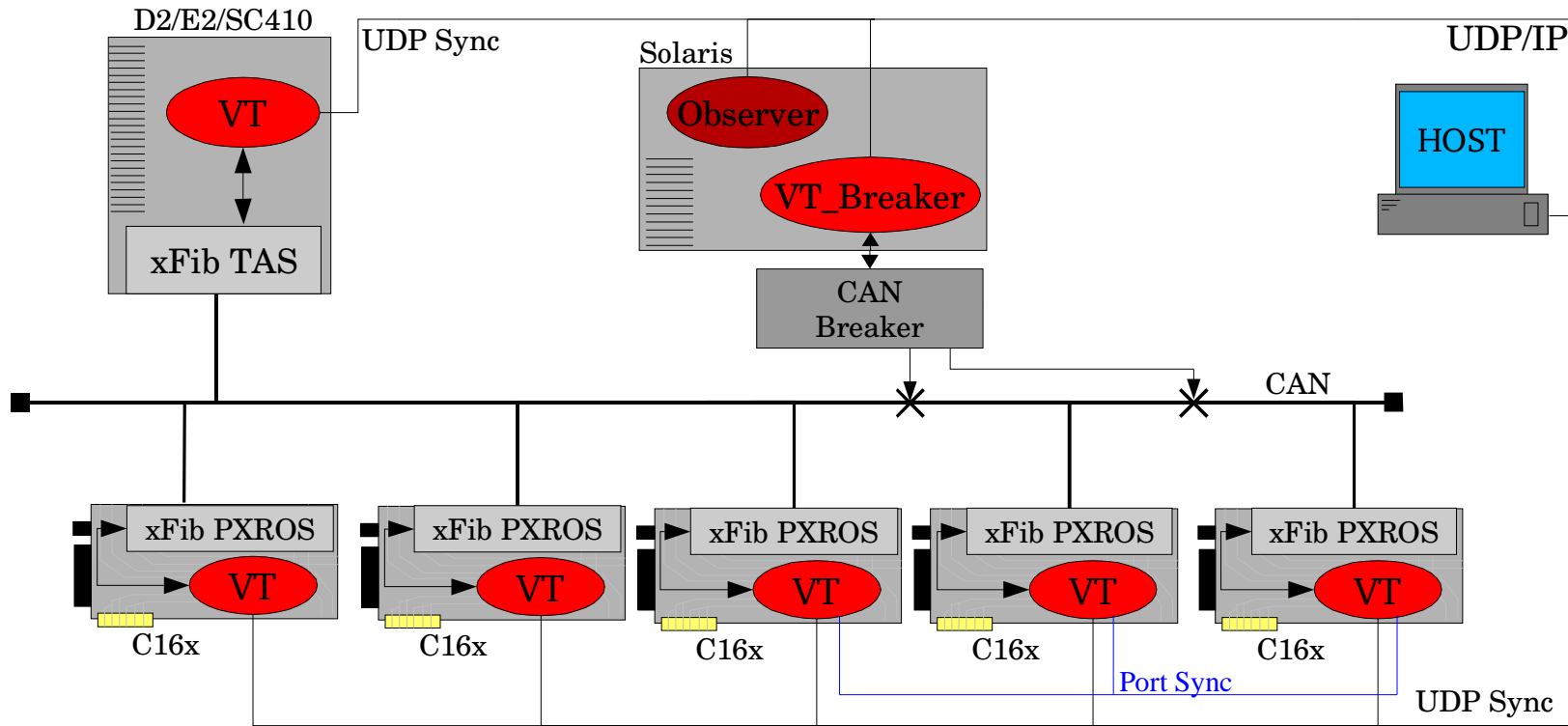
Lösung

- Strukturierung der System Tests:
 - SUT
 - Test-Umgebung
 - Test-Skript (Test-Szenario, Testfälle, Test-Kriterium)
- Alle drei Teile getrennt definierbar und wartbar, aber alles fixiert und kontrollierbar

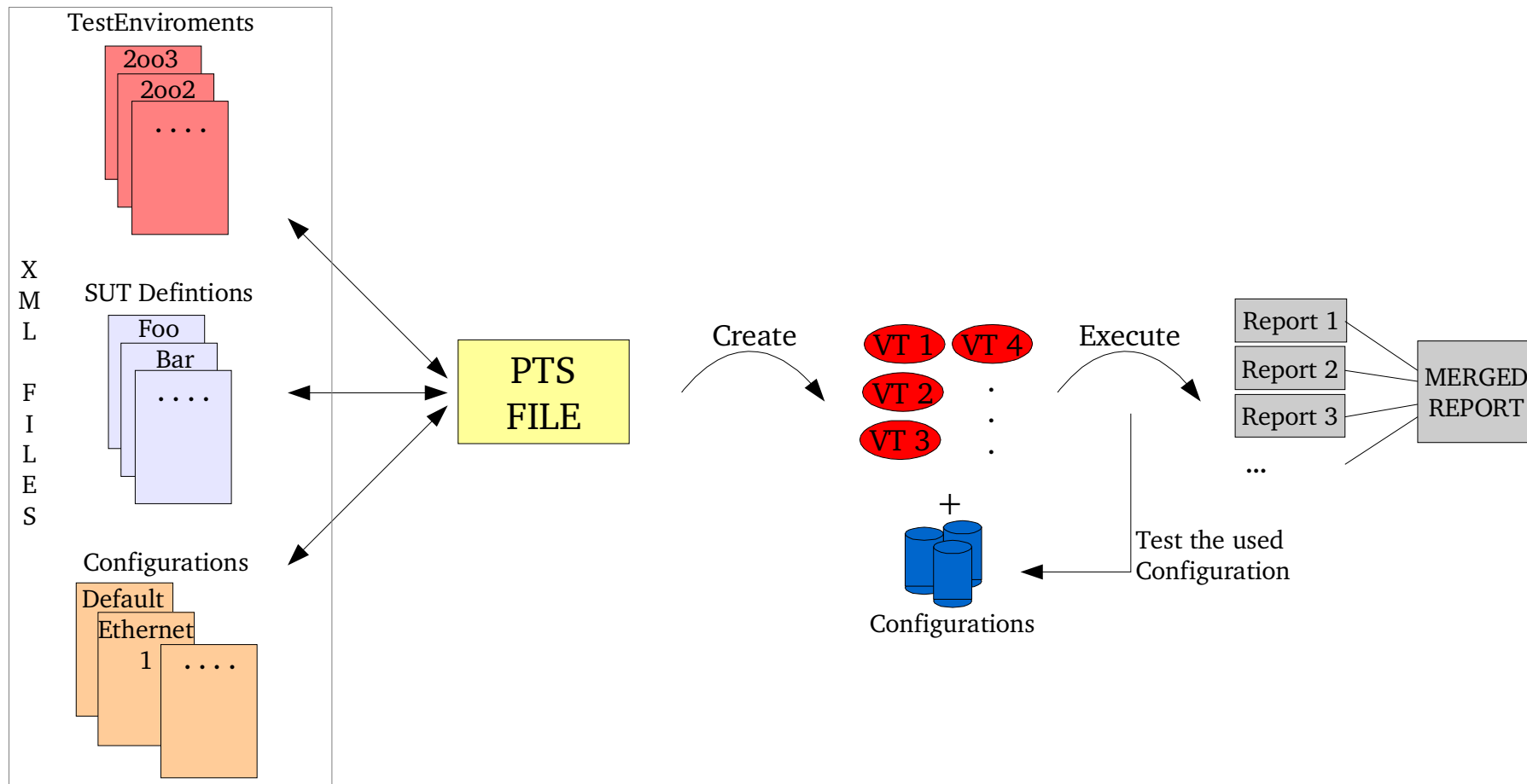






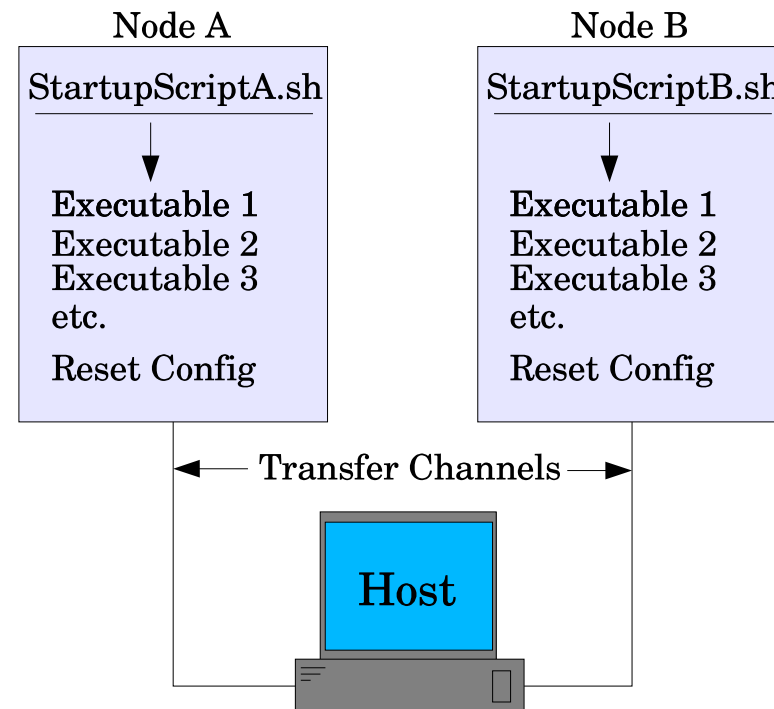


Test Automatisierung: Test Ablauf



SUT Definition

- Optional
- XML File
- Besteht aus: (pro Node)
 - Files
 - Startup Skript
 - Konfigurations-Files
 - Log-Files



```
HEADER "TestClassAndID", $SUT VERSION, "$Revision: 1.4$"
```

```
--XML
--XML <!-- XML Tool Definitions -->
--XML
```

```
-- ----- Includes / Defines / etc. -----
#include "rendezvous_struct.h"
#include "rendezvous.h"

INSTANCE Sender, Receiver:
#include "interface_OCS.h"

  -- Messages used in this script
  MESSAGE vt_msg_t : ocs_msg1, ocs_msg2, ocs_msg3

  INCLUDE "OCS_Macros.pts"
END INSTANCE

INSTANCE Sender2:
#include "interface_OCS.h"

  -- Messages used in this script
  MESSAGE vt_msg_t : ocs_msgSpecial

  INCLUDE "OCS_Macros2.pts"
END INSTANCE
```

```
-- ----- Initialization -----
INITIALIZATION
COMMENT Initialization Block
END INITIALIZATION
```

```
-- ----- Termination -----
TERMINATION
COMMENT Termination Block
END TERMINATION
```

```
-- ----- Scenarios -----
SCENARIO ExampleScenario
COMMENT Starting Test Script
```

```
INSTANCE Sender:
COMMENT Initialize Msg Queues to Helper
CALL initOCS_Queuees()

VAR ocs_msg1.ocs_msg.data, INIT="This is a Test !"
SEND( ocs_msg1, ocs_ch)
END INSTANCE
```

```
INSTANCE Sender2:
COMMENT Initialize Msg Queues to Helper
CALL initOCS_Queuees()

VAR ocs_msg1.ocs_msg.data, INIT="This is a Test !"
SEND( ocs_msg1, ocs_ch)
END INSTANCE
```

```
INSTANCE Receiver:
COMMENT Initialize Msg Queues to Helper
CALL initOCS_Queuees()

  -- Receiving infos from OCS
  DEF_MESSAGE ocs_msg2, EV={
&      ocs_msg => {data=>"DOWN - Virtual Channel is DOWN"}
&      }
  WAITTIL(MATCHED(ocs_msg2), WTIME > 10000)
END INSTANCE
```

```
END SCENARIO
```

Test Automatisierung: Test Umgebungs Generierung 1

```

HEADER "Test Plan for Test: TLinux", "1.0", "1.0"
--XML
--XML <useArchitecture>OneLinuxWithOneVTAndObserver</useArchitecture>
-- <useArchitecture>OneTASWithOneVTAndObserver</useArchitecture>
--XML
-- <useSystemUnderTest>TAS_Single_Dummy</useSystemUnderTest>
--XML
--XML <observer use="true" reset="false"/>
--XML
--XML <timeout-startup unit="sec">20</timeout-startup>
--XML <timeout-testCompletion unit="sec" resetOnError="false">100</timeout-
testCompletion>
--XML
--XML <map>
--XML <instance>
--XML <name>Pan</name>
--XML <identifier>pan</identifier>
--XML <resetVT beforeTest="false" afterTest="false"/>
--XML </instance>
--XML </map>

--
----- Includes / Defines / etc. -----
--
###include "rendezvous_struct.h"
###include "rendezvous.h"

```

```

--
----- Initialization -----
--
INITIALIZATION
  COMMENT Initialization Block
END INITIALIZATION

--
----- Termination -----
--
TERMINATION
  COMMENT Termination Block
END TERMINATION

--
----- Scenarios -----
--
SCENARIO OneTesterAndObserver

  COMMENT Starting Test Script

  INSTANCE Pan:

    COMMENT Pan
    PAUSE(50)
    -- Here we can put our own Testscript

  END INSTANCE

END SCENARIO

```

Test Automatisierung: Test Umgebungs Generierung 2

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE architecture SYSTEM "../xml/Architecture.dtd">

<architecture>
  <name>Ein Linux-PC's mit einem virtuellen Tester und Observer</name>
  <physical>
    <!-- The host node, which contains also the Observer -->
    <node id="host" machine="pentium" isHost="true" pingable="true">
      <name>Host</name>
      <virtualTester id="VirtTester" isObserver="true"
        context="process" mode="standalone">
        <name>VirtTester</name>
        <priority>5</priority>
        <stacksize>1024</stacksize>
        <observerPort>11333</observerPort>
      </virtualTester>
    </node>

    <!-- Node with one VT -->
    <node id="saturn" machine="pentium" isHost="false" pingable="true">
      <name>Saturn</name>
      <resetConfiguration>
        <switchNumber>1</switchNumber>
        <portNumber>1</portNumber>
        <signalLength unit="ms">500</signalLength>
      </resetConfiguration>
      <virtualTester id="pan" isObserver="false" context="process" mode="standalone">
        <name>Saturn</name>
        <priority>5</priority>
        <stacksize>1024</stacksize>
      </virtualTester>
    </node>
```

```
<!-- Connections of the nodes -->
<connection id="milkyway" network="ethernet">
  <name>Milky Way</name>
  <bc-address>192.168.0.255</bc-address>

  <connectionNode node="saturn">
    <address>192.168.0.2</address>
  </connectionNode>
  <connectionNode node="host">
    <address>192.168.0.1</address>
  </connectionNode>
</connection>

</physical>

<logical>

  <!-- Point to Point Channels -->
  <p2p-channel id="p2pChannel" protocol="udp" intern="false" link="milkyway">
    <name>P2P Channel</name>

    <p2p-vt virtualTester="pan">
      <port>4000</port>
    </p2p-vt>
    <p2p-vt virtualTester="VirtTester">
      <port>3000</port>
    </p2p-vt>
  </p2p-channel>
```

Test Automatisierung: Test Umgebungs Generierung 3

```
<!-- Transfer Channels -->
<tf-channel id="host_transfer" protocol="ssh" node="host" link="milkyway">
  <name>Transfer Kanal zum Host</name>
  <directory>/tmp/ST_${USER}</directory>
  <user>${USER}</user>
</tf-channel>

<tf-channel id="saturn_transfer" protocol="ssh" node="saturn" link="milkyway">
  <name>Transfer Kanal zum Saturn</name>
  <directory>/tmp/SystemTestExeSaturn_${USER}</directory>
  <user>${USER}</user>
</tf-channel>

<!-- Rendezvous Init: always necessary for a defined startup -->
<rendezvous id="init">
  <name>Rendezvous im Mondschein</name>

  <rendezvousVirtualTester virtualTester="pan">
    <communication overChannel="p2pChannel"/>
  </rendezvousVirtualTester>

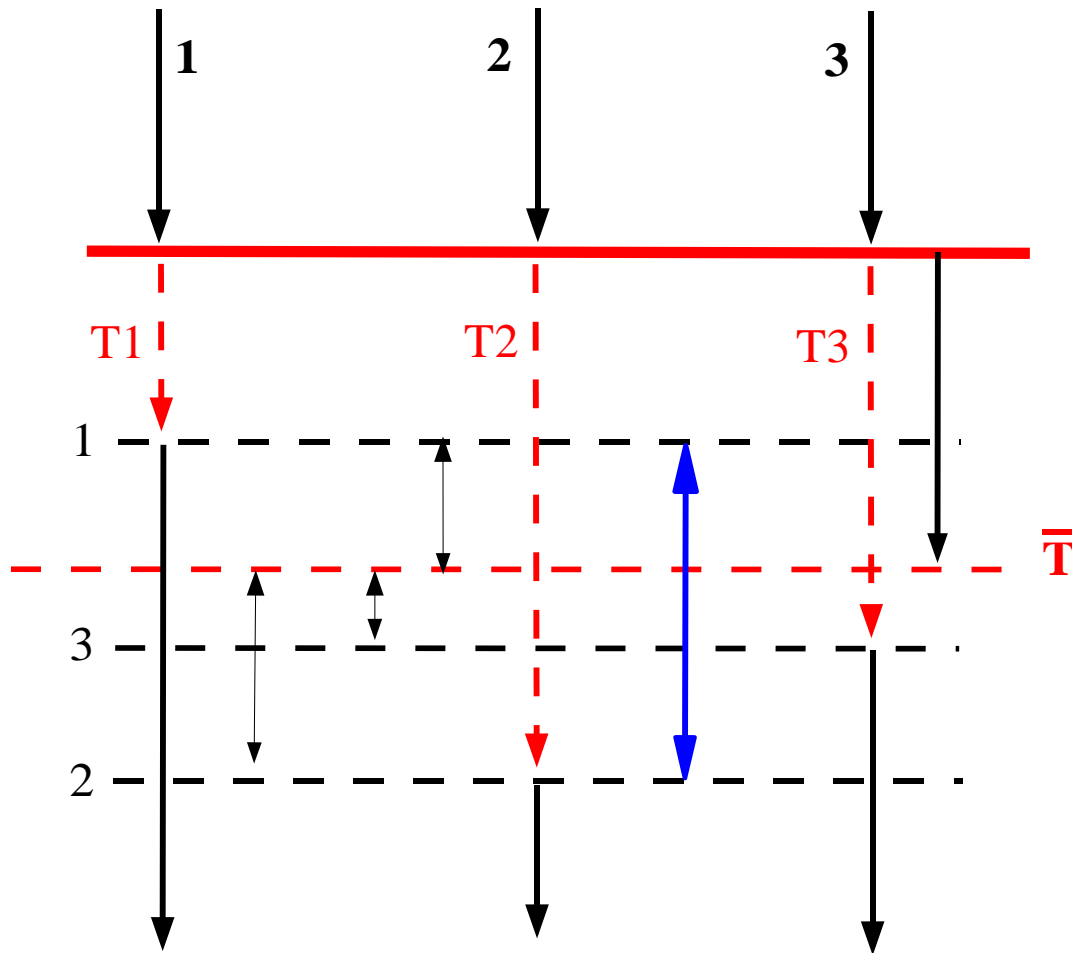
  <rendezvousVirtualTester virtualTester="VirtTester">
    <communication overChannel="p2pChannel"/>
  </rendezvousVirtualTester>
</rendezvous>

<!-- Put your own Rendezvous here ... -->

</logical>

</architecture>
```

Realtime Synchronisation bei ST -> Jitter



Jitter :=
 Maximaler Zeit-Abstand
 zwischen zwei
 synchronisierten,
 loslaufenden Prozessen !

Bsp.-Daten für Jitter:

- Port Sync (Handshake): ca. < 10 μ s
- CAN Sync (xFibuss): ca. < 100 μ s
- TCP/IP Sync: ca. < 1-2 ms

--> Frequenz von Rendezvous im System Test mit dieser Frequenz nicht möglich !!

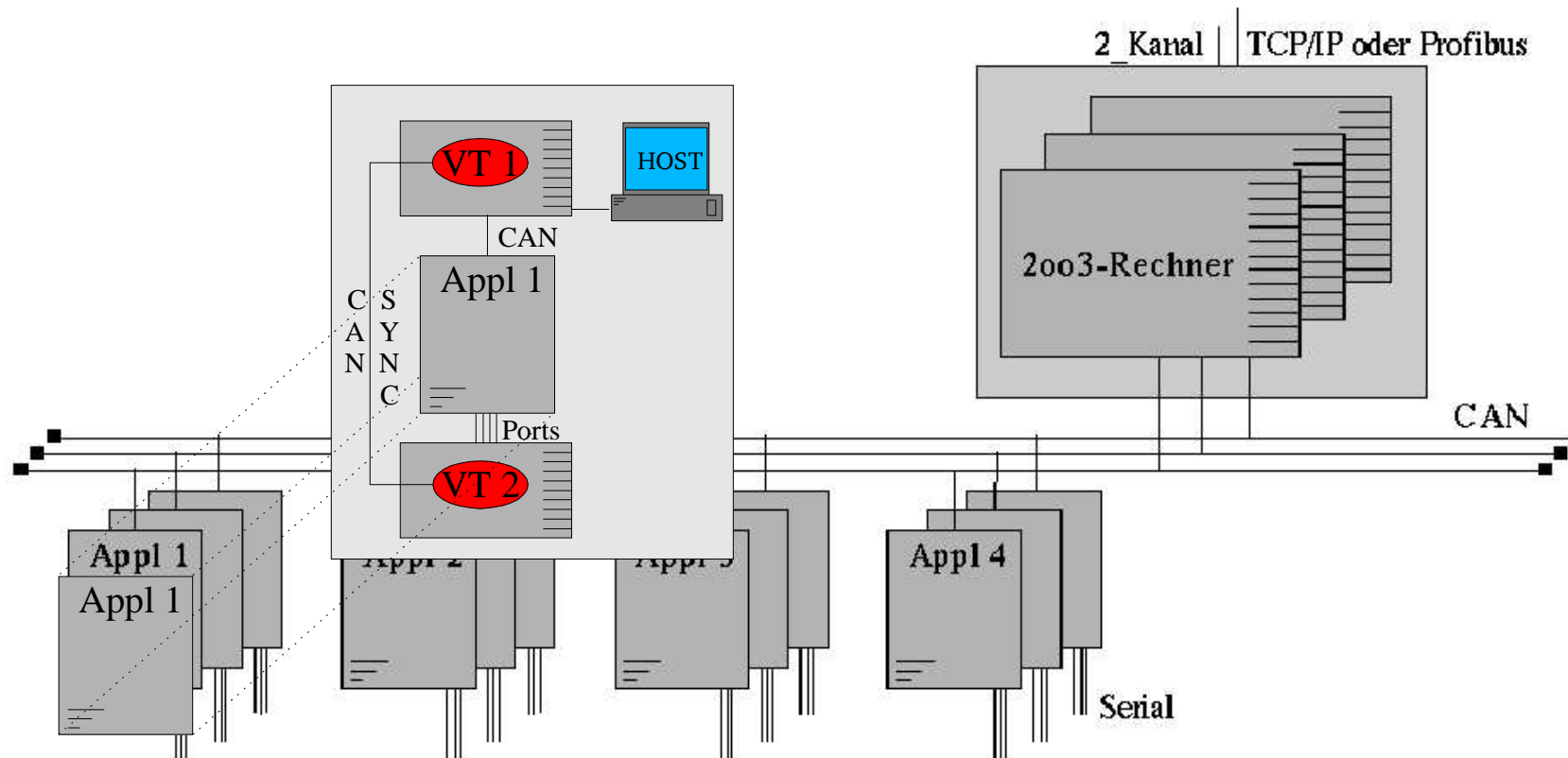
Definierter Zustand der Testumgebung

- Anfang:
überwachte Synchronisation aller VTs
- Ende:
überwachte Synchronisation aller VTs
- Reset von Nodes vor und nach Tests automatisch möglich
- Timeout Überwachung des gesamten Tests

Vorteile: Synchronisation

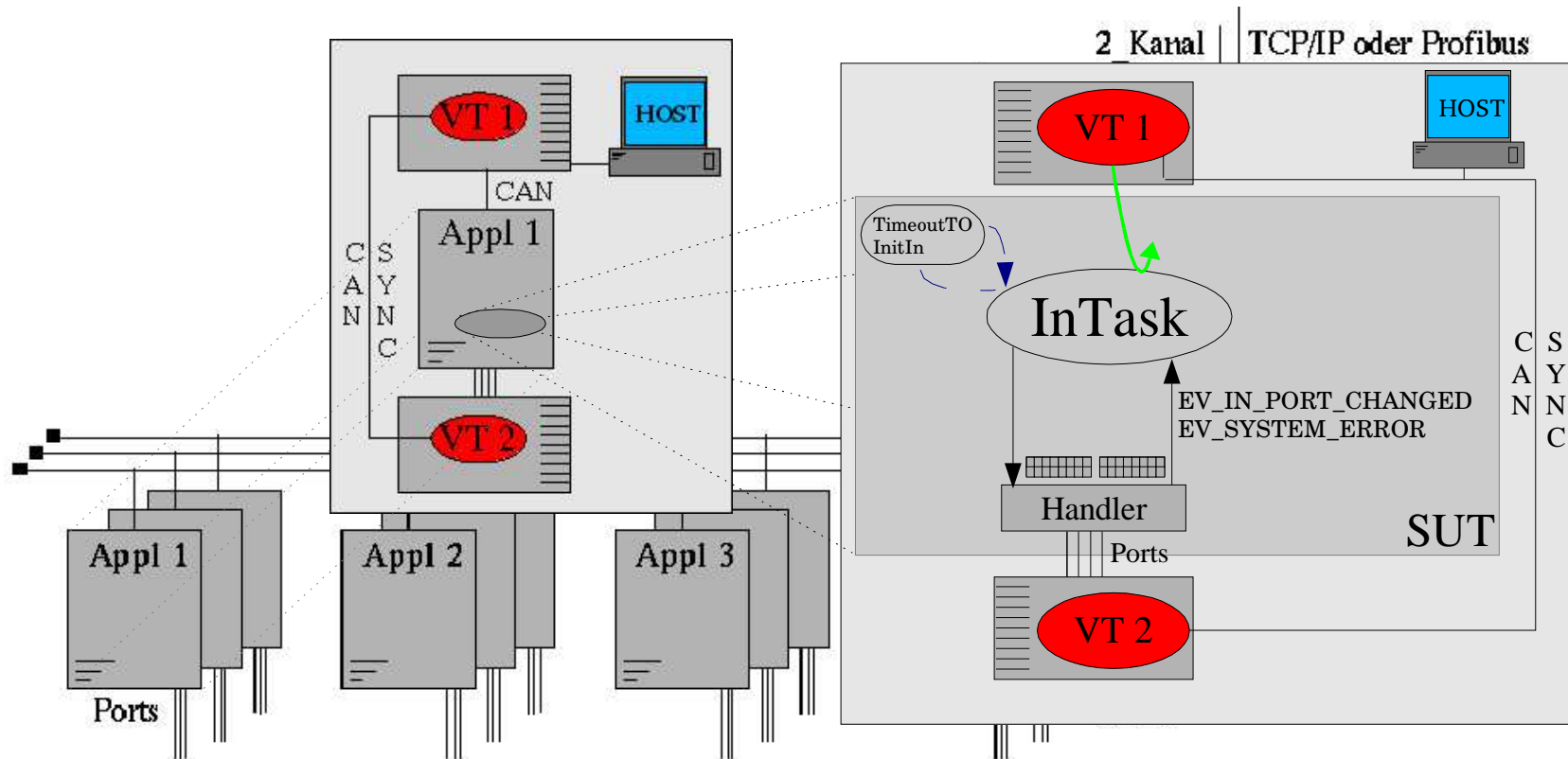
- Echt verteilte Synchronisation mit Echtzeitfähigkeit (Mikrosekunden)
- Heterogene Synchronisation:
im Testszenario und in einem Rendezvous (Gateways)
- Testskripte unabhängig von Rendezvous Physik
- Beliebige Sync Gruppen definierbar

Vorteile: Durchgängigkeit



z.B. Diagnose

Vorteile: Durchgängigkeit



z.B. Entwicklertests

Vorteile: Wiederverwendung

- Kapselung der Testumgebungsfunktionalität in VTs
- Wiederverwendung in vielen Projekten
- Schnittstellen zum VT konstant:
Wiederverwendung der Ansteuerung der VTs in
Test-Skripten
- Lesbarkeit der Testskripte auch für Validierer
- VTs wiederverwenden von Entwicklertests bis zu
Verifikations/Validationstests

Vorteile: Arbeitsteilung

- Kein Spezialwissen über Tests bei einzelnen Personen
- Test HW klar definiert und sie wird überprüft zur Testlaufzeit: inkrementelles Vorgehen mgl.
- SUT / Test-Umgebung / Test-Skript Makros und Testfall sind auf unterschiedliche Personen einfachst abzubilden

Vorteile: Details

- Parameter-Variationen im Test-Skript, d.h. Testdatenraum automatisiert ablaufen
- SUT Konfigurationsparameter automatisch Durchlaufen
- Verschiedene SUTs (z.B. HW) automatisiert ablaufen lassen
- Allgemeine SUT Logfile Behandlung in Testskripten

==> mehr Tests und somit erhöhte Testtiefe

Ausblick

- Expertensystem oder/und GUI für Testumgebungs bzw. SUT Definition
- Coverageunterstützung integrieren (gcov, etc.)
- Erweitertes Testmanagement: z.B. andocken an Requirementsmanagement bzw. Test-Spezifikation
- Verbesserung der Reports, der Test-Skript -Sprache und der teilweisen Echtzeitunfähigkeit des RTRT-Tools.

DANKE



Gerne können wir bei uns in Emmendingen auch
Vorführungen organisieren !