

# Middleware als Basis für offene und serviceorientierte Testsysteme

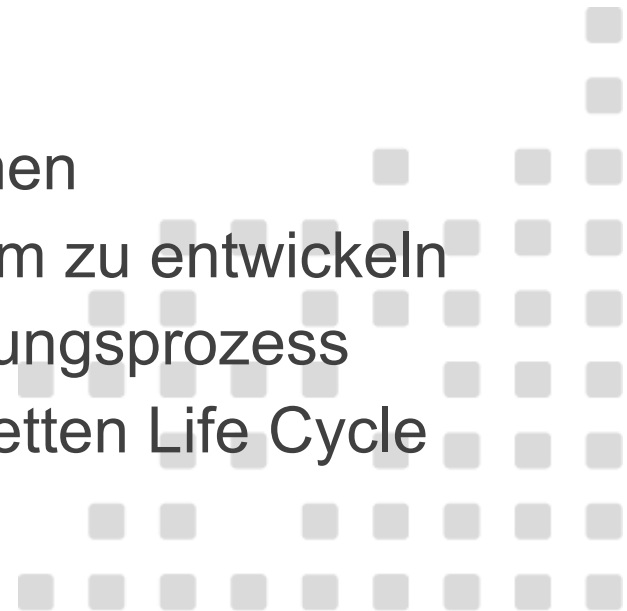


Robert Schachner  
089 961601814  
[rschachner@rst-automation.de](mailto:rschachner@rst-automation.de)

- Wenige Testfirmen beherrschen den Markt
- Testsysteme sind meist proprietär
- Erweiterungen meist nur durch den Hersteller
- Kosten für den Test sind hoch

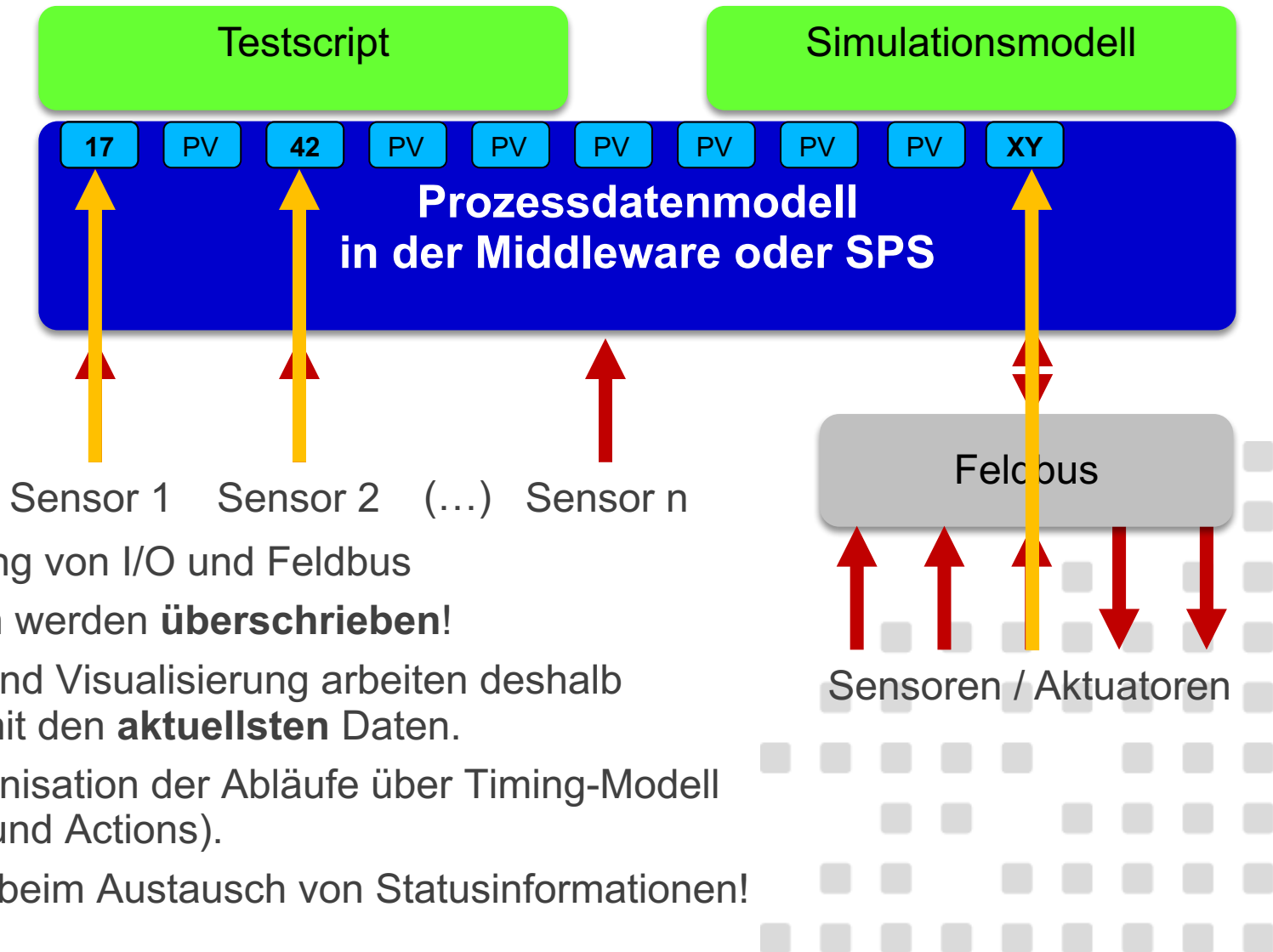
## Der Kunde erwartet aber...

- Offene, leicht erweiterbare Testplattformen
- Die Möglichkeit, Testsysteme gemeinsam zu entwickeln
- Anpassbarkeit an den internen Entwicklungsprozess
- Die Produktbegleitung durch den kompletten Life Cycle

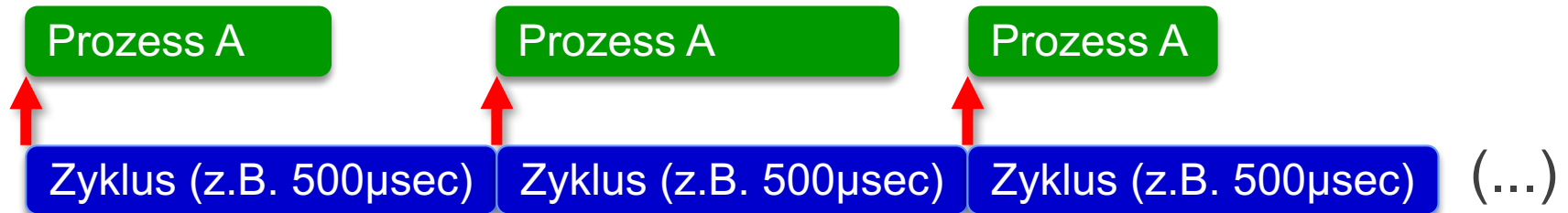


# Time Frame-basierte Kommunikation

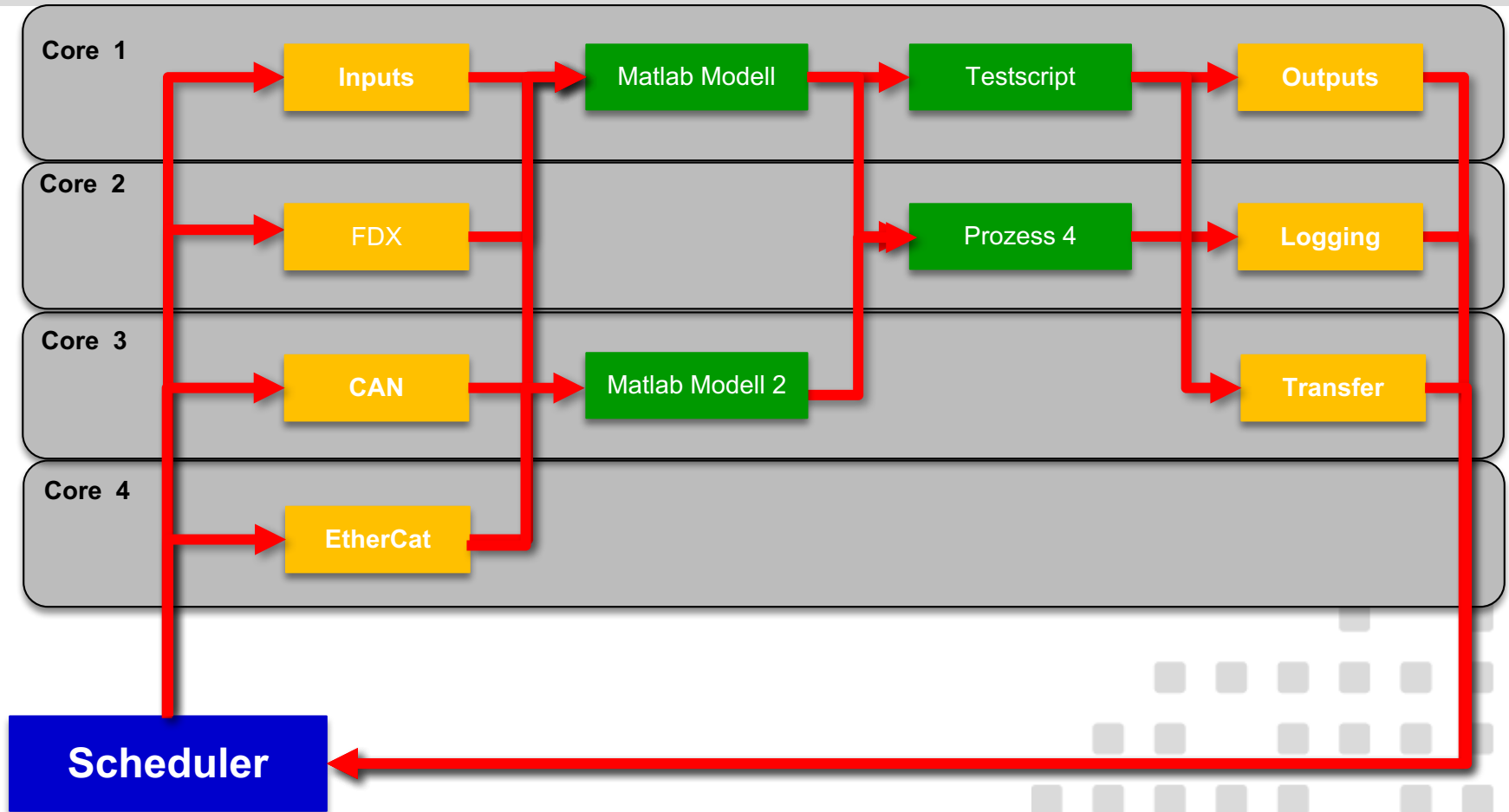




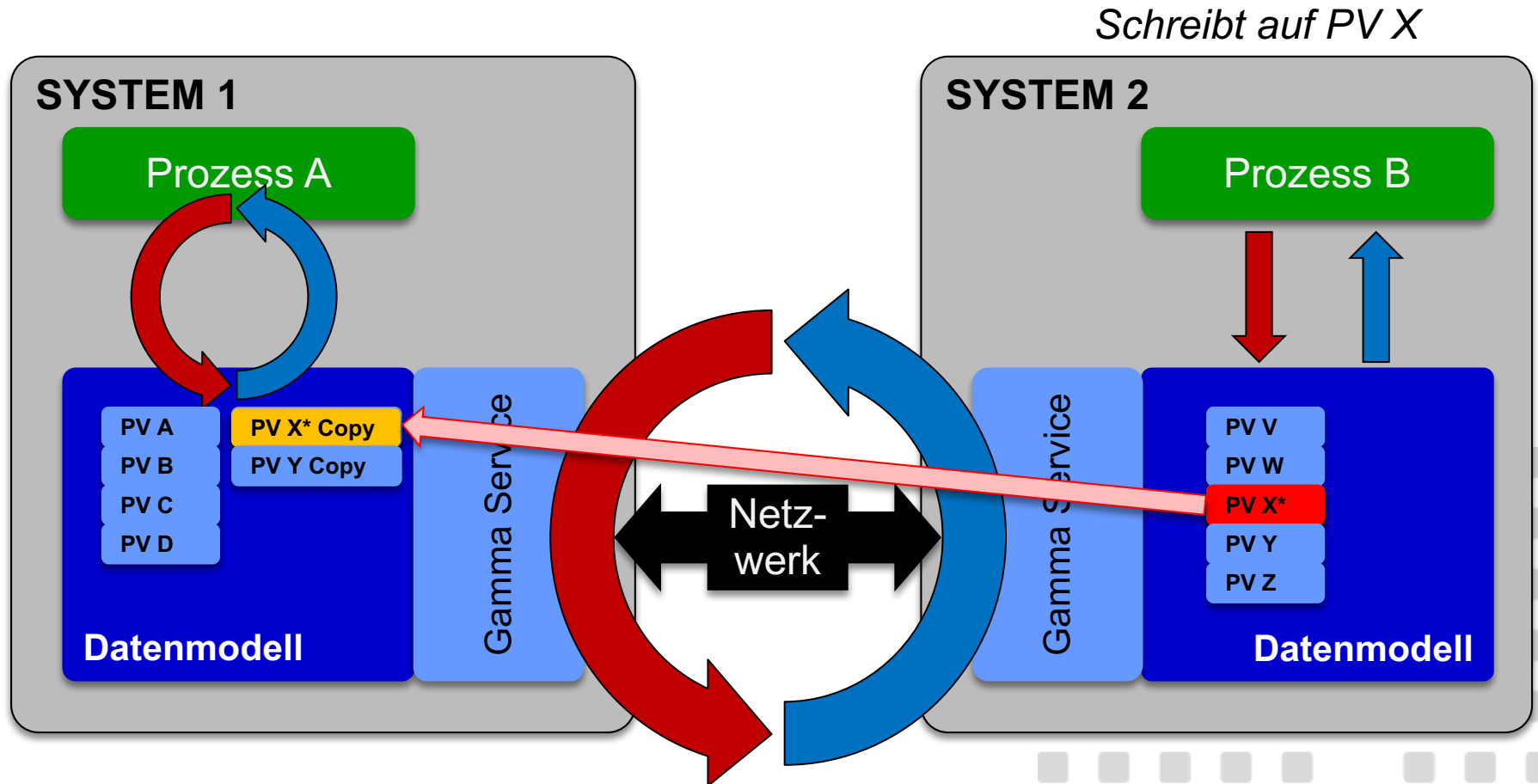
- **Loop:** für zyklisch wiederkehrende Aktionen



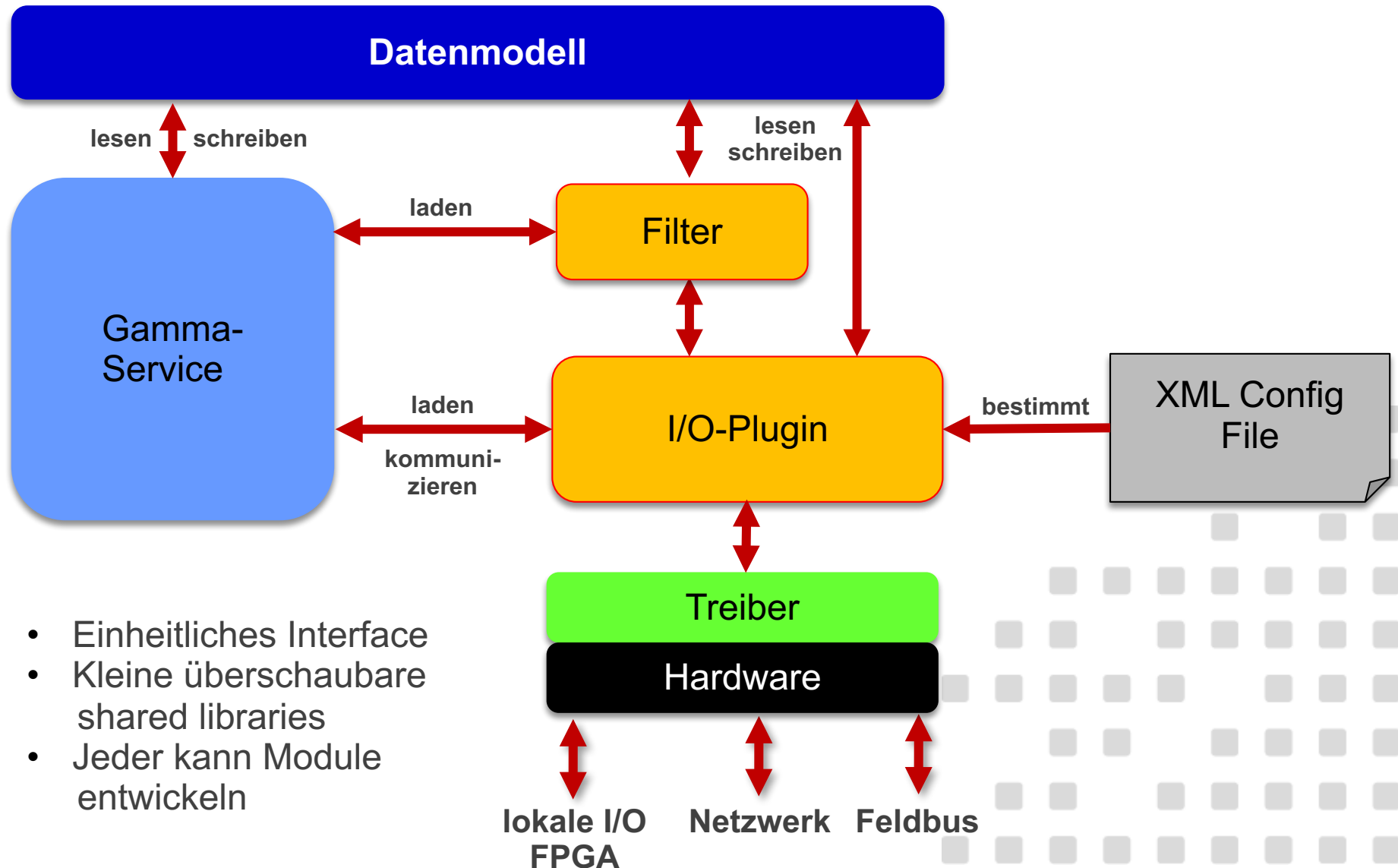
- Die Zyklusdauer ist unabhängig von der Prozesslaufzeit der angestoßenen Funktion
  - Ideal für Aufgaben, die in exakten Abständen aufgerufen werden müssen (z.B. Eingänge prüfen, Logs durchschreiben, usw.)
- 
- **Actions:** zur Integration von Hardwareereignissen der I/O
    - Interrupts können so abstrakt und hardwareunabhängig Softwareaktionen auslösen.



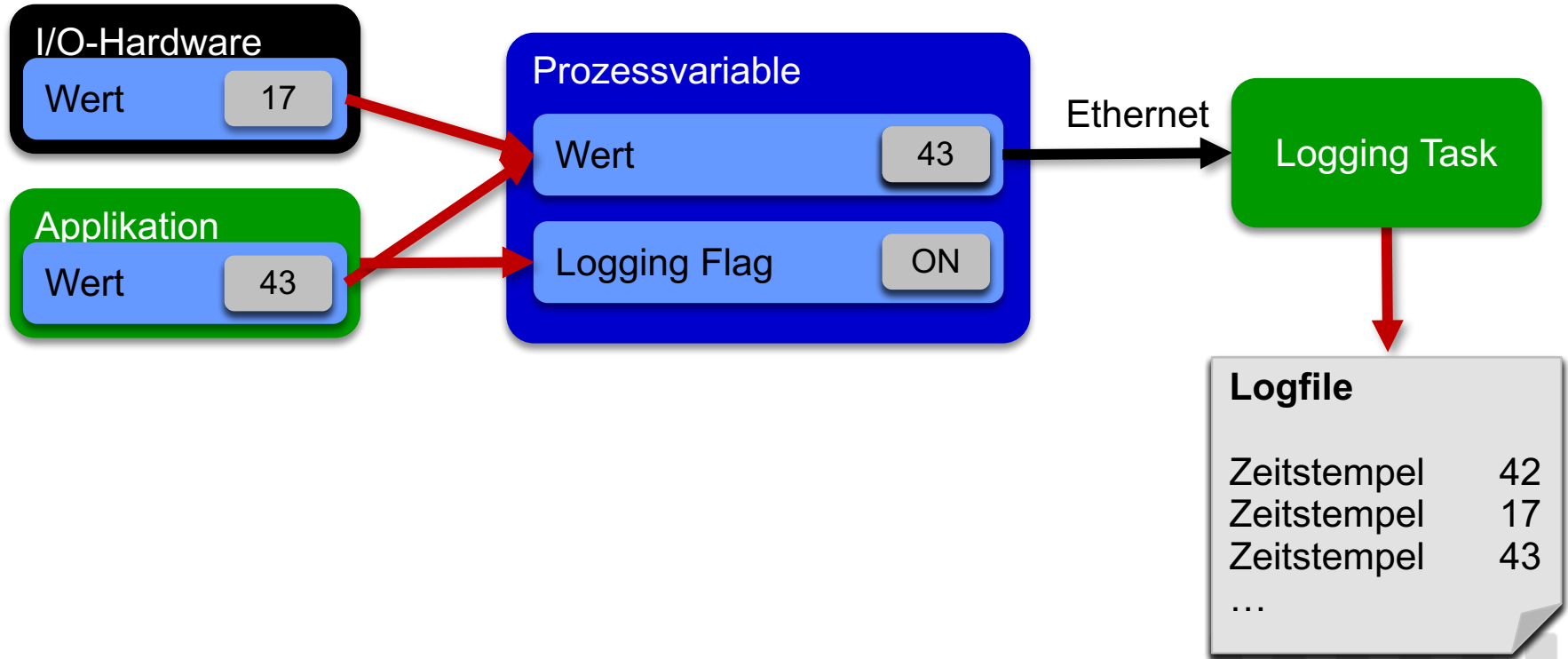
Parallelisierung verkürzt die absolute Laufzeit und vereinfacht die zeitliche Kontrolle der Abläufe.



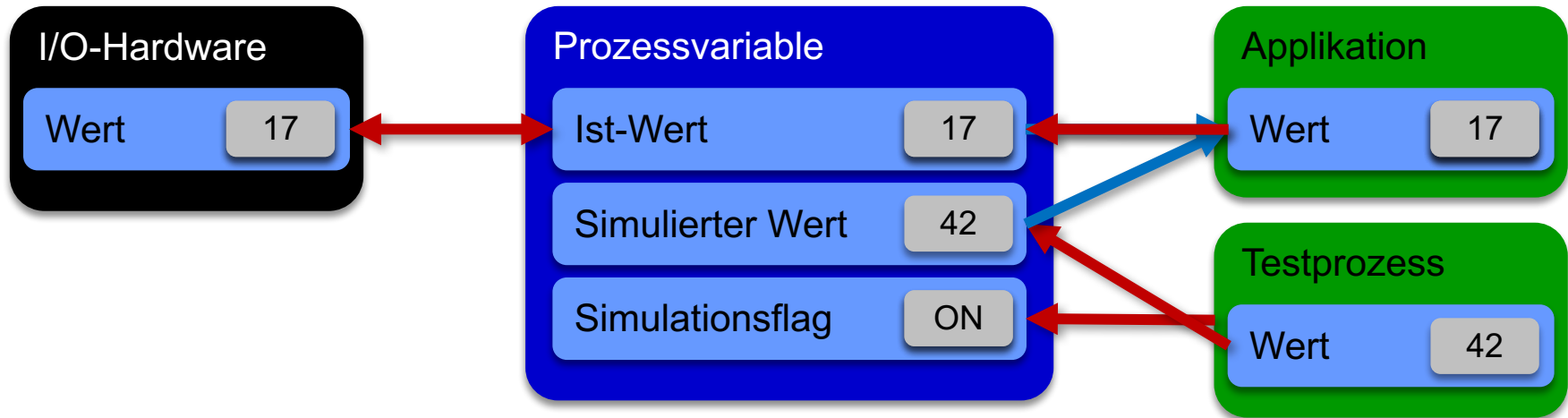
Derzeit Standard sind einfache zyklische Transfers (konfiguriert über XML-Datei).



- Einheitliches Interface
- Kleine überschaubare shared libraries
- Jeder kann Module entwickeln



- Dynamisch während der Laufzeit schaltbar
- Logfiles können über das Netzwerk auf einem externen Rechner angelegt werden.
- **Neu:** Logging des Timings aller Prozesse und Datenmodellfunktionen zur Optimierung der Abläufe

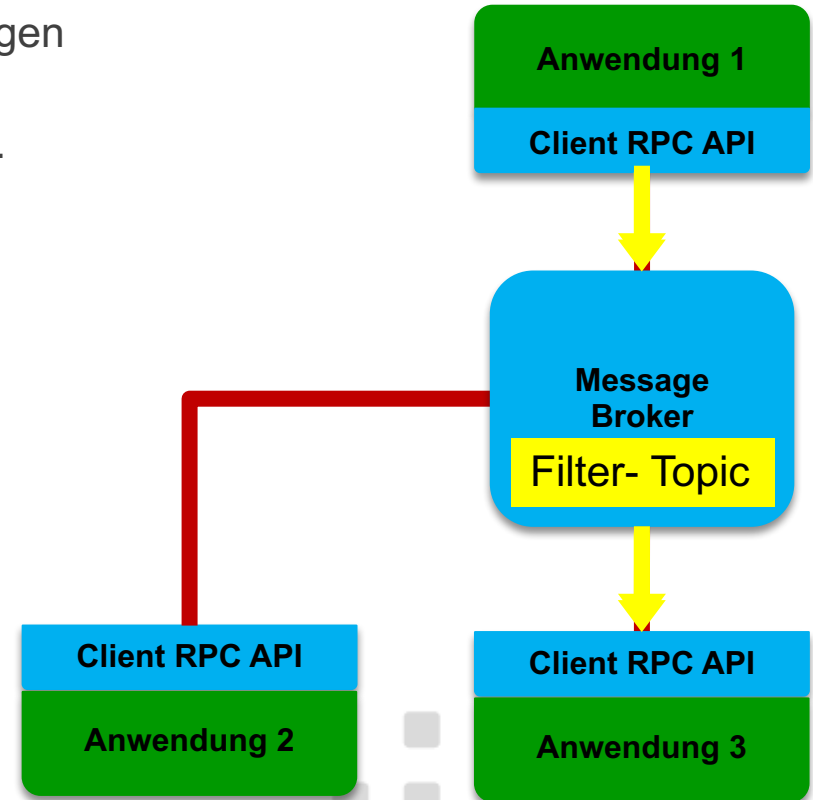


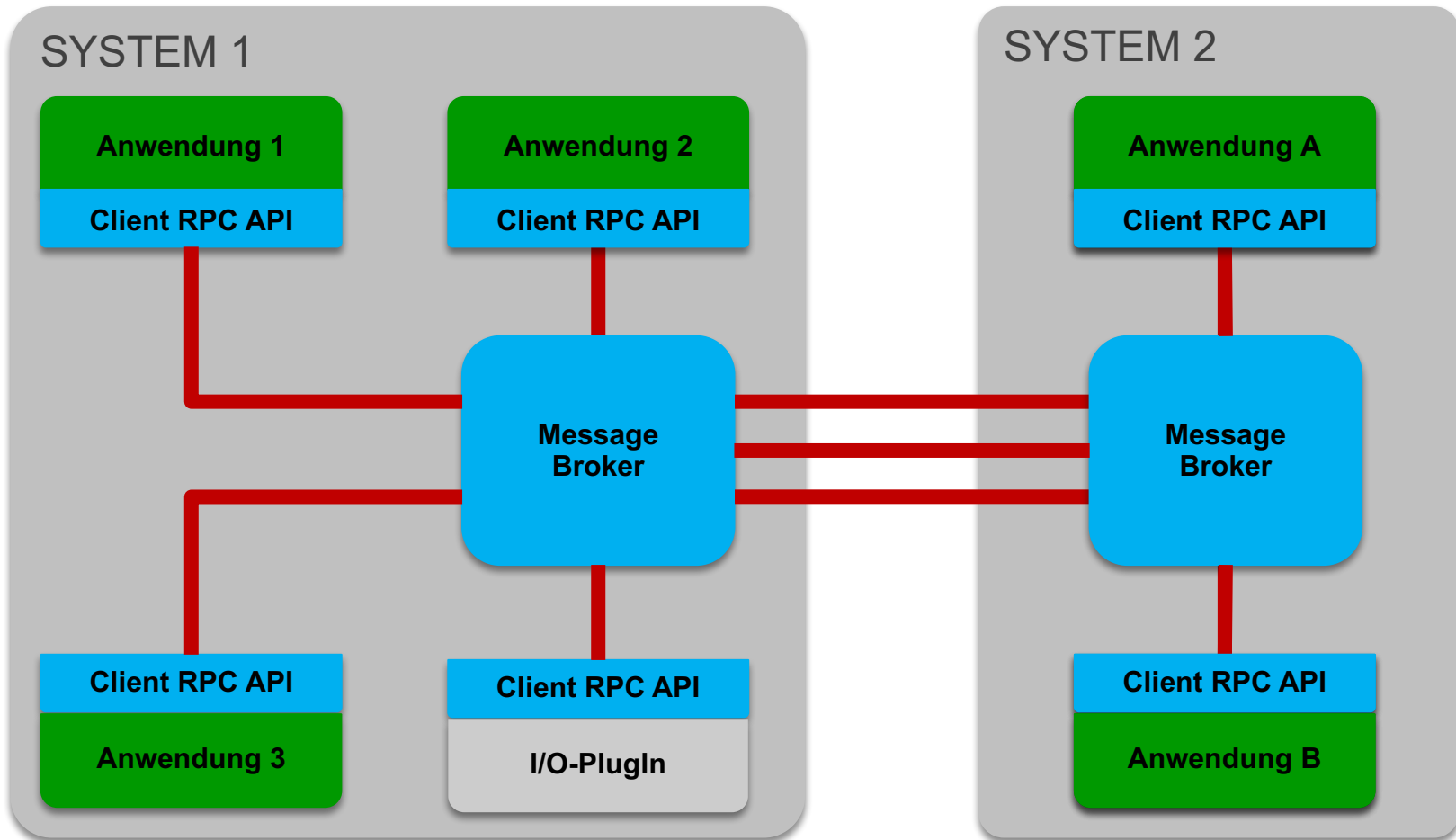
- Die Simulation ist dynamisch während der Laufzeit schaltbar.
- Der Simulationswert ist unabhängig vom aktuellen Ist-Wert.
- Die API stellt für Applikationen einen dedizierten Befehl zum Schreiben auf den Simulationswert bereit.
- Lesezugriffe von Userprozessen erhalten bei aktiviertem Simulationsflag den Simulationswert.
- Normale Schreibvorgänge in das Datenmodell verändern den Ist-Wert.
- Die Hardware liest und schreibt unabhängig vom Sim-Flag *immer* den Ist-Wert.

# Messagebasierte Kommunikation



- Broker verteilen Nachrichten zwischen Anwendungen und Systemen.
- Nachrichten werden meist über Topics organisiert.
- Clients senden Messages an den Broker.
- Clients empfangen Messages vom Broker.
- Daten gehen nicht mehr verloren.
- Für zeitdiskrete Tests zu langsam!





- Lokale und systemübergreifende Verteilung über **einen** Broker
- Systemübergreifender Datenaustausch über Broker – Broker Kommunikation

## ■ Datenkommunikation Punkt-zu-Punkt

- Sender adressieren Nachrichten an einen expliziten Empfänger.

## ■ Funktionsaufrufe durch Remote Procedure Calls

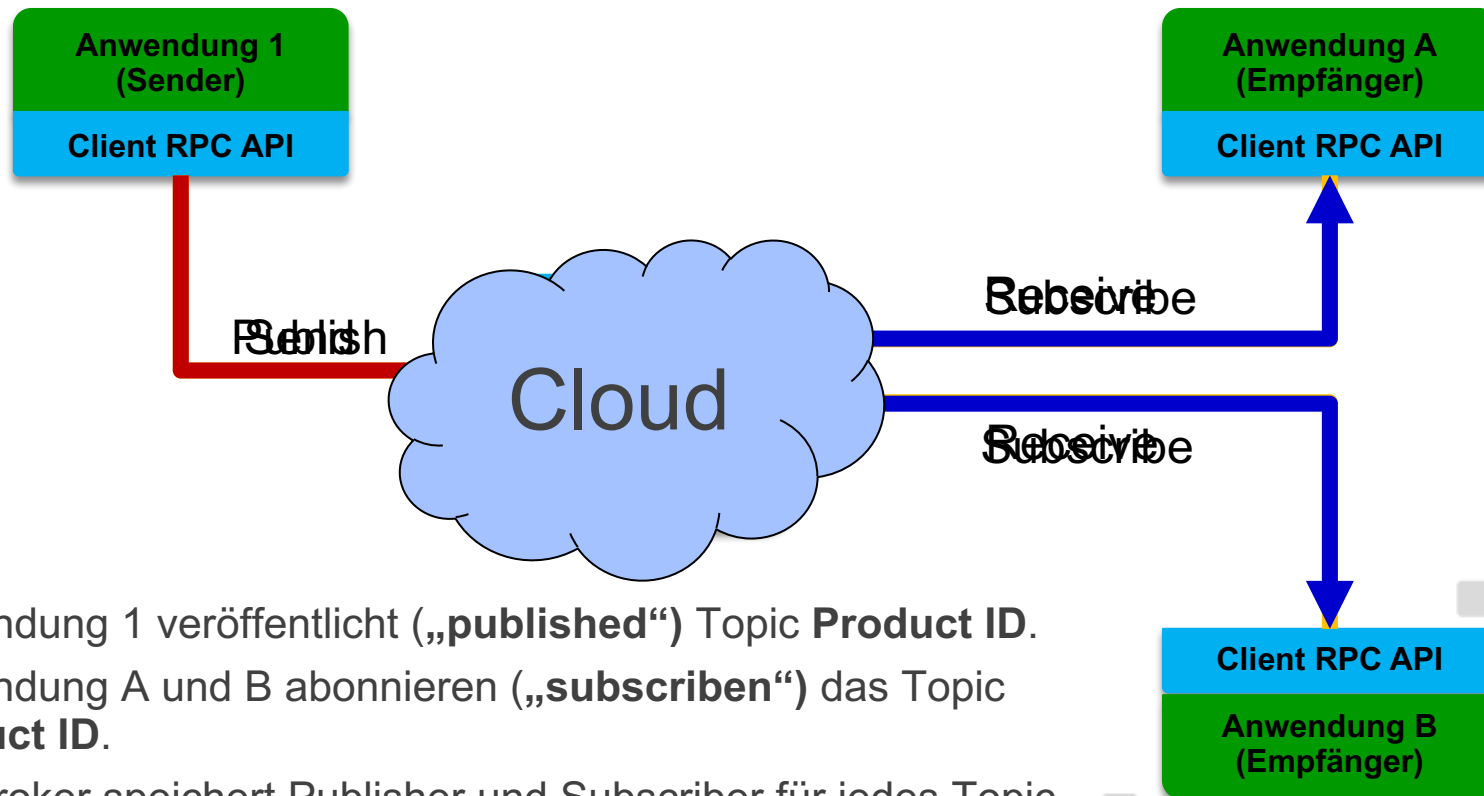
- Aufruf und Verteilung von Kommandos über Prozessgrenzen hinweg.

## ■ Datenkommunikation Publish / Subscribe

- Sender (Publisher) veröffentlichen Informationen unter Topics.
- Empfänger (Subscriber) abonnieren relevante Topics.
- Kein *direkter Bezug* mehr zwischen Sender und Empfänger.
- Wildcards unterstützen das Abonnieren von Topics.

## ■ Last Will / Retain

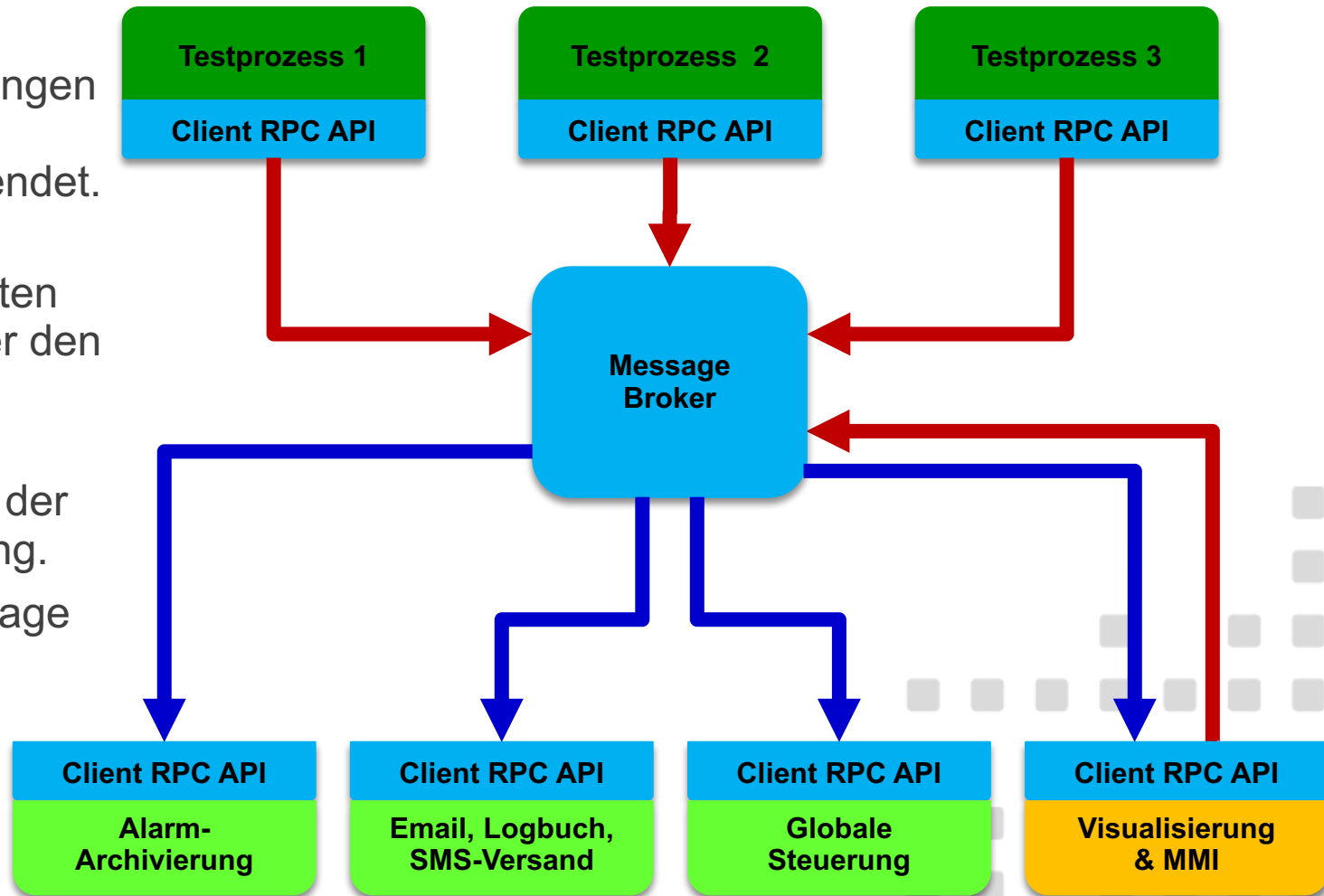
- Ein Publisher hinterlässt beim Broker eine Nachricht, die versendet wird, wenn die Verbindung unterbrochen wird.
- Ein Subscriber erhält vom Broker die letzte Message über das Topic, nachdem die Verbindung abgebrochen und wiederhergestellt wurde.



- Anwendung 1 veröffentlicht („**published**“) Topic **Product ID**.
- Anwendung A und B abonnieren („**subscriben**“) das Topic **Product ID**.
- Der Broker speichert Publisher und Subscriber für jedes Topic.
- Anwendung 1 sendet eine Message unter dem Topic „**Product ID**“, die vom Broker an Anwendung A und B weitergeleitet wird.

**Wichtig: Sender und Empfänger wissen nichts voneinander!**

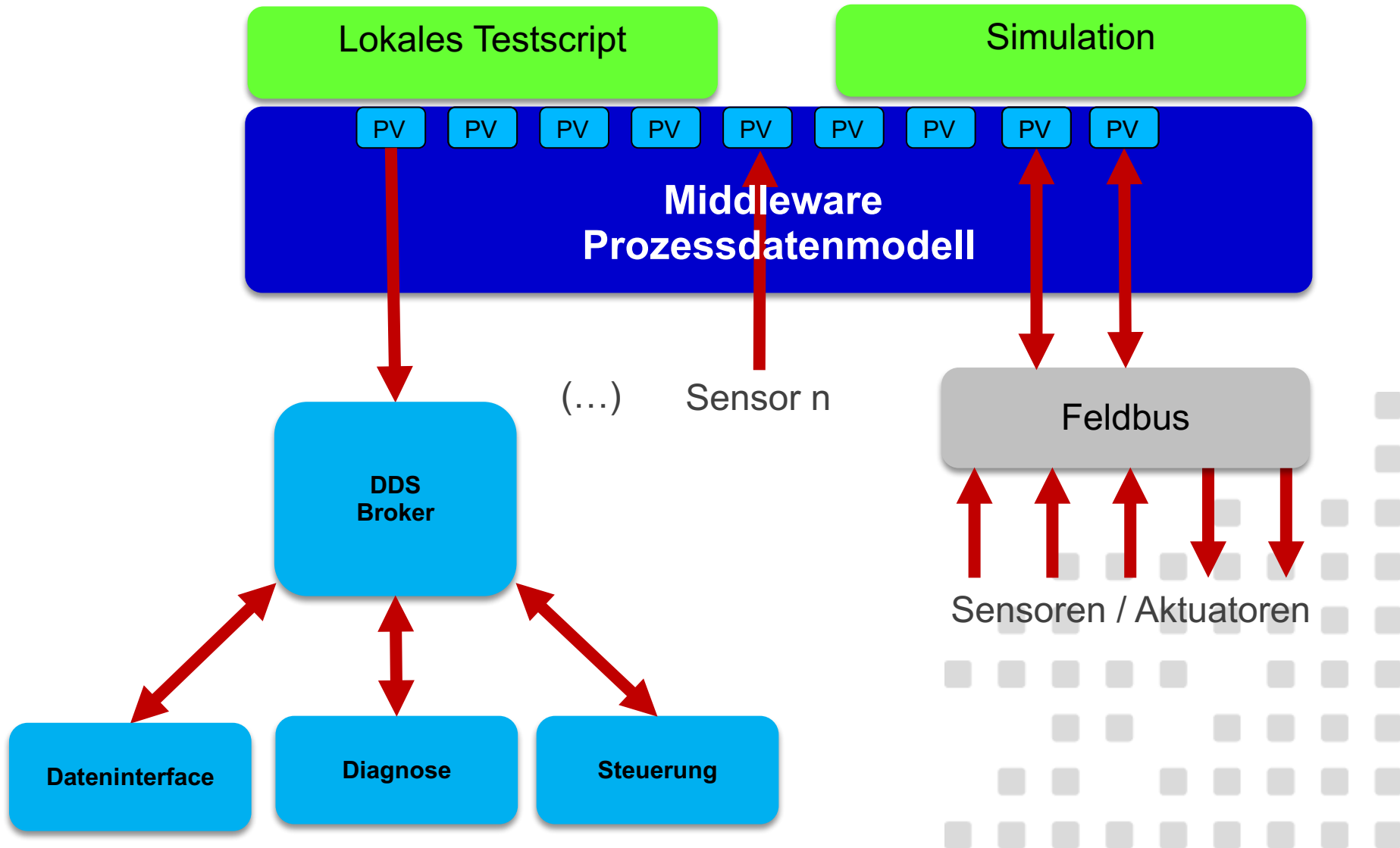
- Alarme & Warnungen werden von Publishern versendet.
- Abonnierte Subscriber erhalten die Meldung über den Broker.
- Die geforderten Trigger sind Teil der Alarmverarbeitung.
- Message Brokerage funktioniert über Systemgrenzen hinweg.



**■** : nur Publisher  
(sendet Daten)

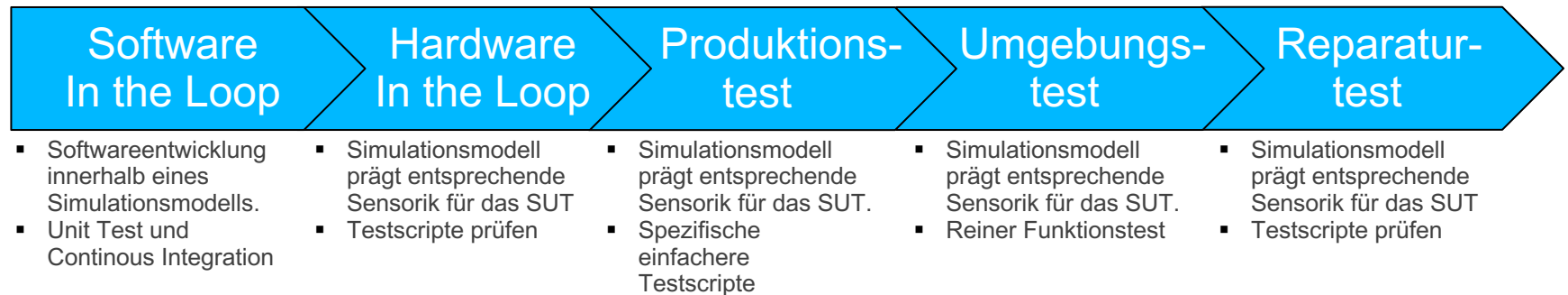
**■** : nur Subscriber  
(empfängt Daten)

**■** : Publisher / Subscriber  
(sendet und empfängt)



# Testen entlang des gesamten Life Cycles

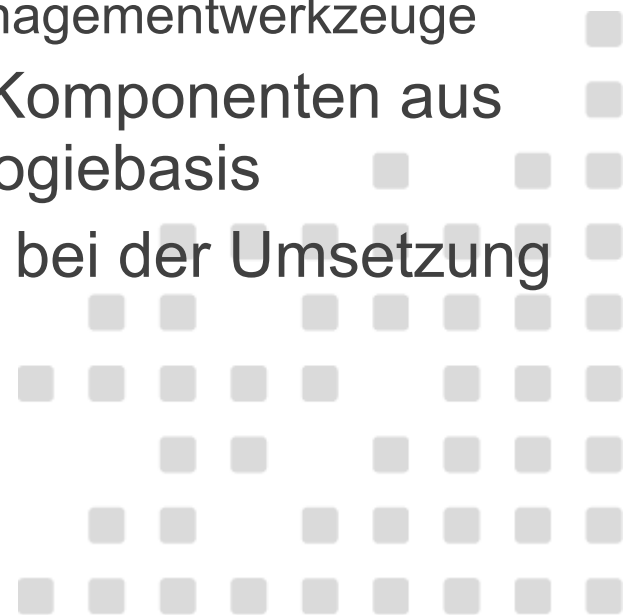




- Die Middlewareplattform begleitet das Produkt durch den gesamten Life Cycle.
- Das Simulationsmodell entsteht bei der Entwicklung und bleibt über den gesamten Zyklus unverändert.
- Die Testverfahren werden jeweils angepasst.

**Der Entwicklungs- und Testzyklus wird mit jedem neuen Produkt optimiert und den Prozessen angepasst!**

- Zusammenarbeit innovativer Firmen in Technik und Marketing
- Einheitliche Middlewaretestplattform als technologische Integrationsplattform für:
  - Hardwareplattformen (x86, PowerPC und ARM)
  - I/O-Funktionen, Feldbusse und Spezielle Messtechnik
  - Diverse Test-, Simulations- und Qualitätsmanagementwerkzeuge
- Freie Konfiguration durch Auswahl von Komponenten aus einer immer größer werdenden Technologiebasis
- Enge Zusammenarbeit mit dem Kunden bei der Umsetzung seiner Life Cycle-Prozesse





Mitglieder Embedded4You e.V.



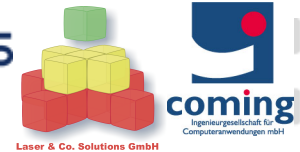
## Test



## Mechanics



## Embedded Systems & Automation



## Research & Education



## Safety & Security



[www.embedded4you.com](http://www.embedded4you.com)

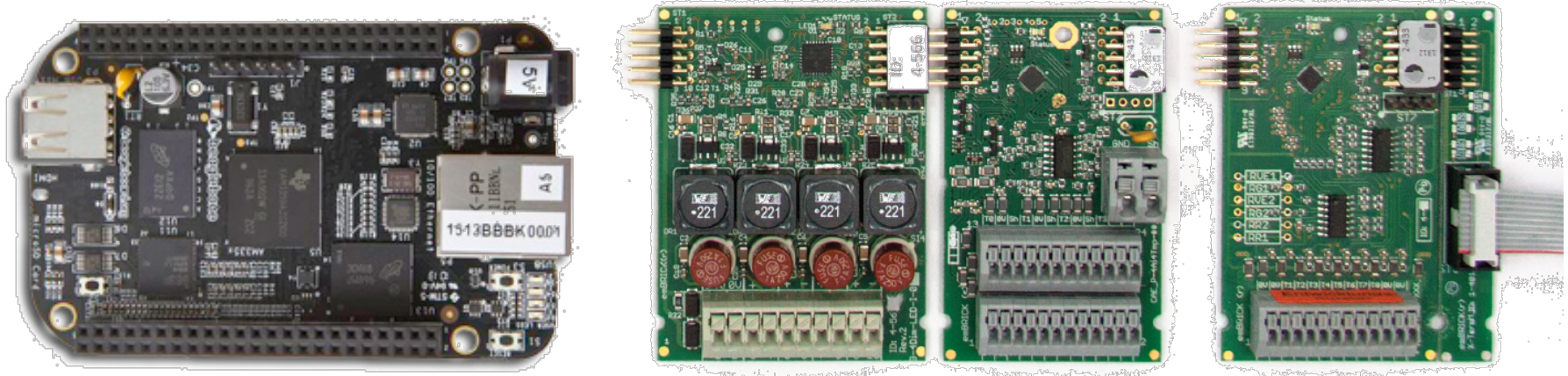


**Passende  
Hardwareplattformen  
für jeden Einsatzfall!**





# emBRICK als I/O-Basis für Serienmaschinen



Beagle Bone Black mit Open Source I/O-Konzept



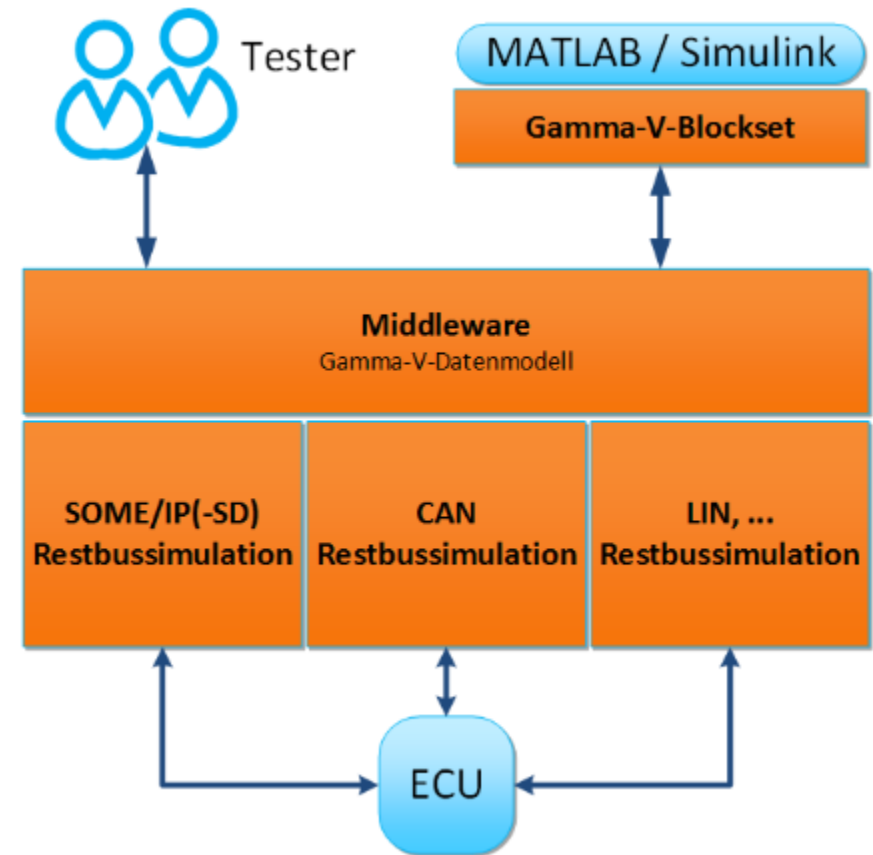
## CAN-SOME/IP-Gateway (nach AUTOSAR 4.2)



Gateway der Firma iSyst

ARM Quadcore Testgerät:

- Visualisierung
- CAN-Bus
- Ethernet (SOME/IP)





Mini-HIL Prüfstand iSyTester der Firma iSyst

- ARM Core-basierte Rechnerplattform mit eingeschränkter Funktionalität (CAN, EtherCAT, Ethernet, emBRICK)



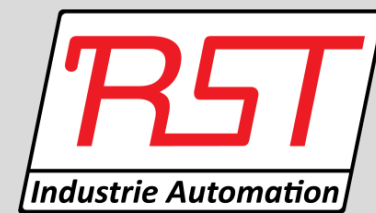
# Entwicklungs- / Produktionsprüfstand



x86 Multicore-PC  
mit PCI und PCIe



# Bremen Test Center

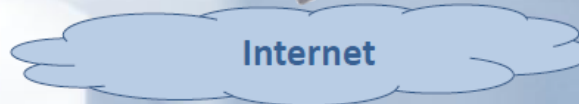


BRETECE  
BREMEN TEST CENTER



x86 Multicore-PC mit PCI und PCIe als Flugsimulator





Testsysteme mit FastWAN standortübergreifend vernetzt!

# Passende Werkzeuge oder Werkzeugkombinationen



## Demo in Python:

```
# import libraries
import gaapi
import gacommon

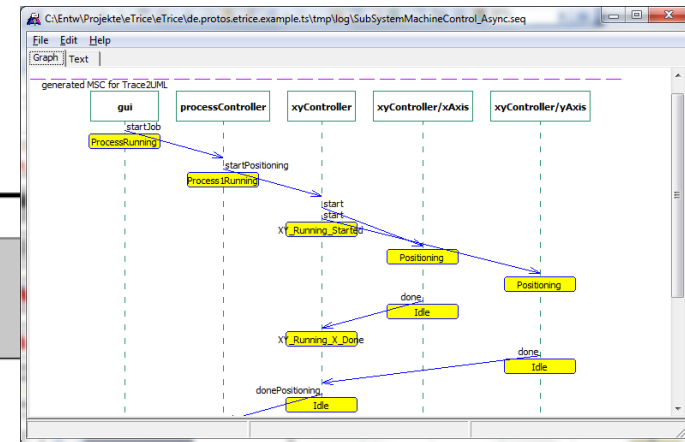
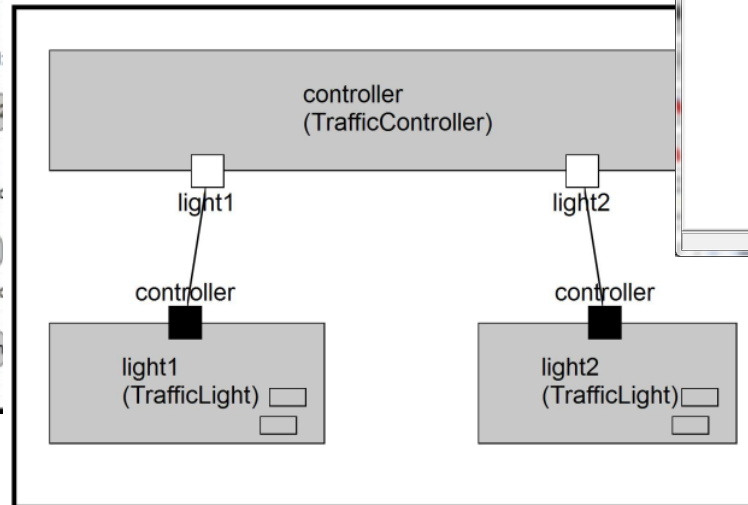
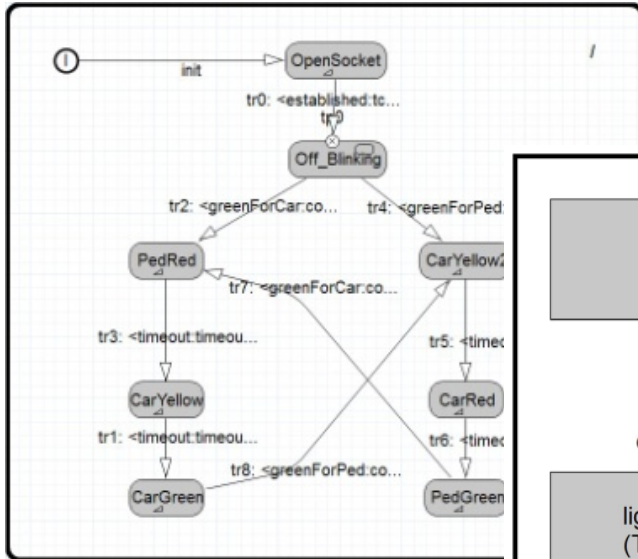
# attach to gamma middleware
gacommon.attachToService()

# attach process variable
ExamplePV = gaapi.PV( ":Node.Array.Value" )

# read process variable
print "Opened variable with value %s" % ExamplePV.value
# write process variable
ExamplePV.value = 123
print "Changed variable value to %s" % ExamplePV.value
```



# Modellbasierte Entwicklung: Beispiel Eclipse eTrice



ntation of the FSM \*/  
e ifitem, int evt, Obj  
VT\_SHIFT\*evt;



[www.eclipse.org/eTrice](http://www.eclipse.org/eTrice)

```
boolean skip_entry = false;  
  
if (!handleSystemEvent(ifitem, evt, generic_data)) {  
    switch (this.state) {  
        case STATE_OpenSocket:  
            switch(trigger) {  
                case TRIG_tcpCtrl_established:  
                    {  
                        chain = CHAIN_TRANS_tr0_FROM_OpenSo.  
                        catching_state = STATE_TOP;  
                    }  
                    break;  
            }  
            break;  
        case STATE_CarYellow:
```



# ITE, CCDL, TOP und TRM



changes to renamed 2 | Test Frame Details | host: simhawk2/TRM4 stabil - TRM-Client Entwicklung [TRM v4.2.2, TRM-API v4.0.3.1]

Frame Import/Export Queue Window Help

Test Execution Procedure Edit

Attached File Browser Execution Flow Interval: 150 [ms]

11 12 13

wait 100... wait 500 [...] wait 500 [...] wait 500 [...] wait 1500 [...] wait 1500 [ms] wait 500... wait...

within 0 [ms] .. 2500 [ms]: expect HLSS2.OD\_POBOpen\_V => 1

set HLSS2.OD\_POBOpen\_V to 1

set HLSS2.OD\_POBOpen\_V to 0

set HLSS2.OD\_POBOpen\_V to 0

set HLSS2.OD\_POBOpen\_V to 0

set HLSS2.OD\_POBOpen\_V to 0

Log Messages Attached File Preview Parameter Browser for procedure develop

Schedule Messages

Build Messages

Run Messages

Messages from the last execution run of this test frame.

@@Exitstatus= 0@@

01/22 12:44:20.939 [INFO |scheduler] Starting run, frame rate:300.03 Hz, frame length:3333 Usec, run up

01/22 12:44:20.942 [WARNING |scheduler] Task /hlss.root/RTW/null\_rtw/null\_rtw assigned to inactive cp

01/22 12:44:20.944 [WARNING |scheduler] Task /hlss.root/Script/.TestRunner/.TestRunner assigned to c

01/22 12:44:20.949 [WARNING |scheduler] Task logtxin assigned to cpu 5 while we have 1 cpu's availab

01/22 12:44:20.951 [WARNING |scheduler] Task logtxout assigned to cpu 4 while we have 1 cpu's availab

01/22 12:45:23.060 [INFO |.TestRunner\_1] Successful end of CCDL program

Show full text

Test Frame Information Attributes

Core Data

Name: changes to renamed 2

Test Frame Id: 10142

Test Case Id: -

Current State: EXECUTED

Created at: 05.12.11 09:09:20

Modified at: 22.01.14 11:44:14

Modified by: wittner

Locked by: wittner

Execution/Evaluation

Test Frame result data from the last execution run of this test frame.

Scheduled: successfully | 22.01.14 11:44:16 - 22.01.14 11:45:23

Test\_Frame\_10142.ccd

```
wait 1000 [ms]
set HLSS2.OD_POBOpen_V to 1
wait 500 [ms]
set HLSS2.OD_POBOpen_V to 0
wait 500 [ms]
set HLSS2.OD_POBOpen_V to 1
wait 500 [ms]
set HLSS2.OD_POBOpen_V to 0
wait 1500 [ms]
}

// No change
Test Step 12: {
    within 0 [ms] .. 1000 [ms]:
        expect HLSS2.OD_POBOpen_V => 1

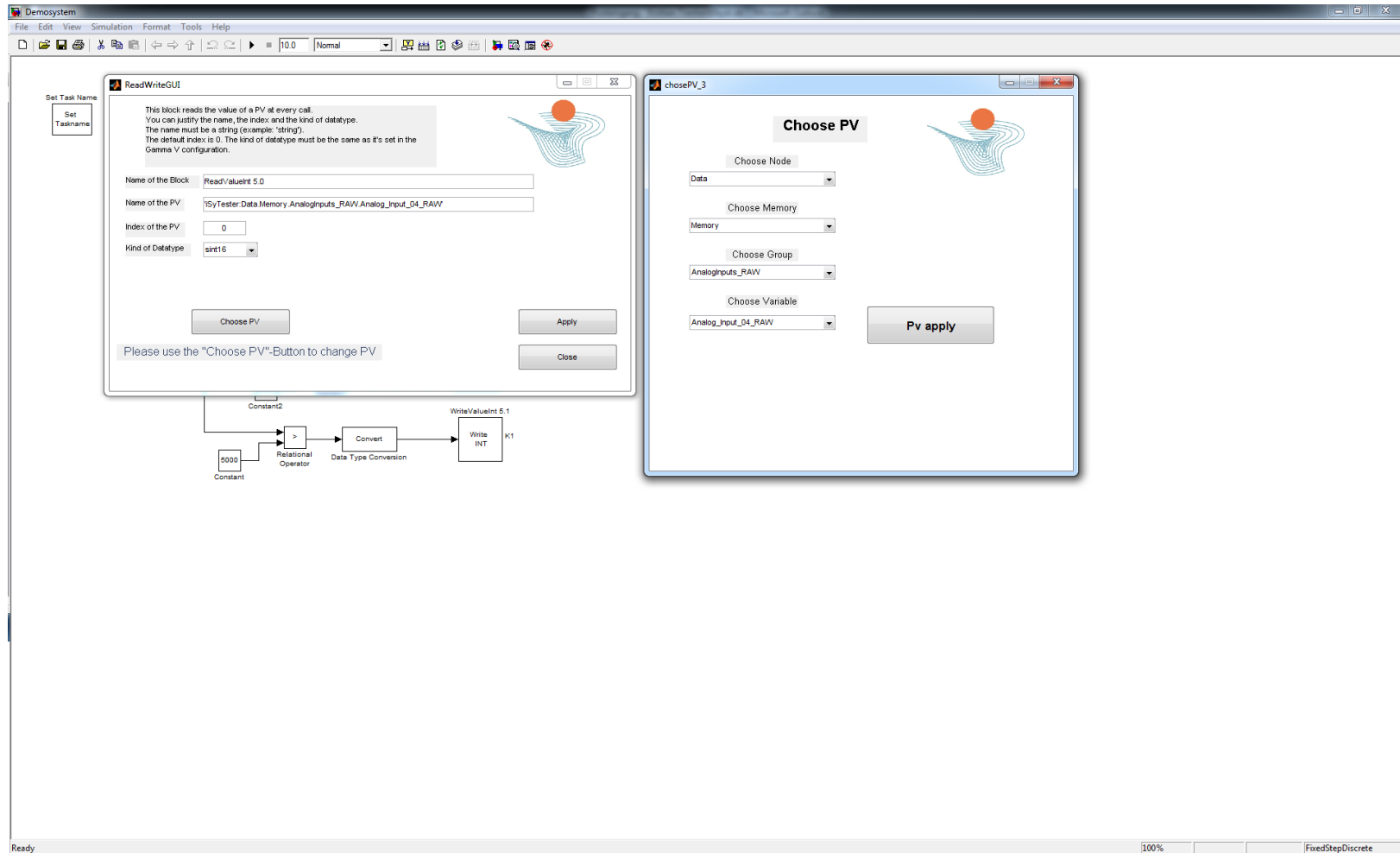
    wait 1500 [ms]
}
```

wittner [O]E[C] Queue Mode: RIG: off Simulator

```
Test Step 1, Timeout 99 [s]: {
  // Action: Set lever position to 2
  set CTRL.LeverPosition to 2
  // Check for motor speed of system
  // - Set a trigger variable if the event occurred
  set trigger T1 when CTRL.MotorSpeed >= 1000 [rpm]           (RQMT:0815-1)
  // When system is in state "ready for this test":
  // Set failure condition: Manipulate sensor
  when T1:
    // Manipulate sensor value
    set CTRL.MotorSpeed to offset 110 [rpm]                   (RQMT:4711-1)
  // Check if system detects the failure condition
  within T1 .. T1 & 100 [ms]: {
    expect CTRL.BreakState          => ENGAGED                (RQMT:4711-1)
    expect CTRL.FailureWarning      => 1                      (RQMT:4711-1)
  }
}
```

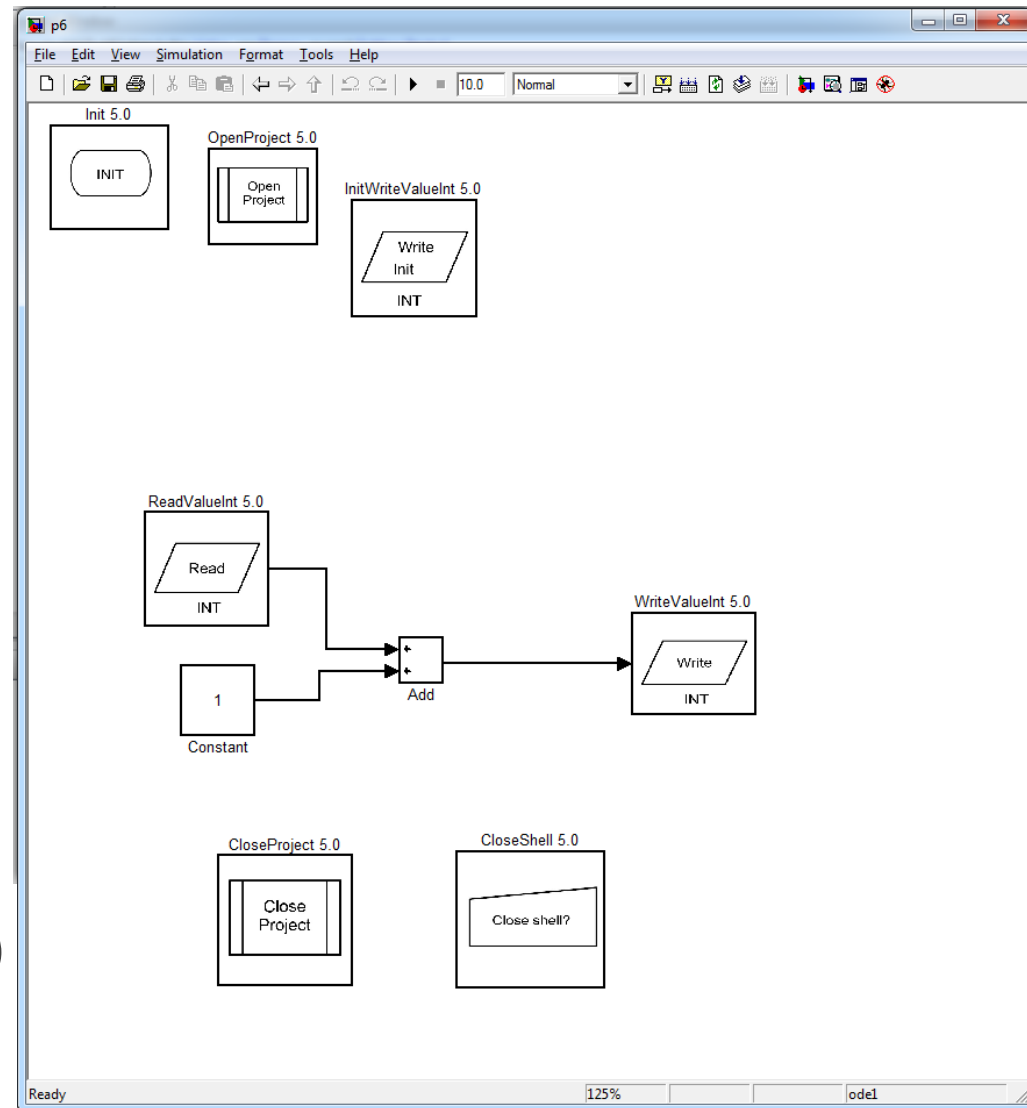


# Matlab Simulink Steuerung / Simulation



Alternativ können auch vorkompilierte Modelle von Airbus verarbeitet werden.

- Einbindung der Middleware anhand von S-Functions
- S-Function Builder Block
- Lauffähig direkt in Matlab Simulink, unter Windows und Linux
- Multicorefähig über Middleware-Scheduler
- Einfache Integration durch Templates
- Einfache Übernahme von Modellen (z.B. von DSpace)



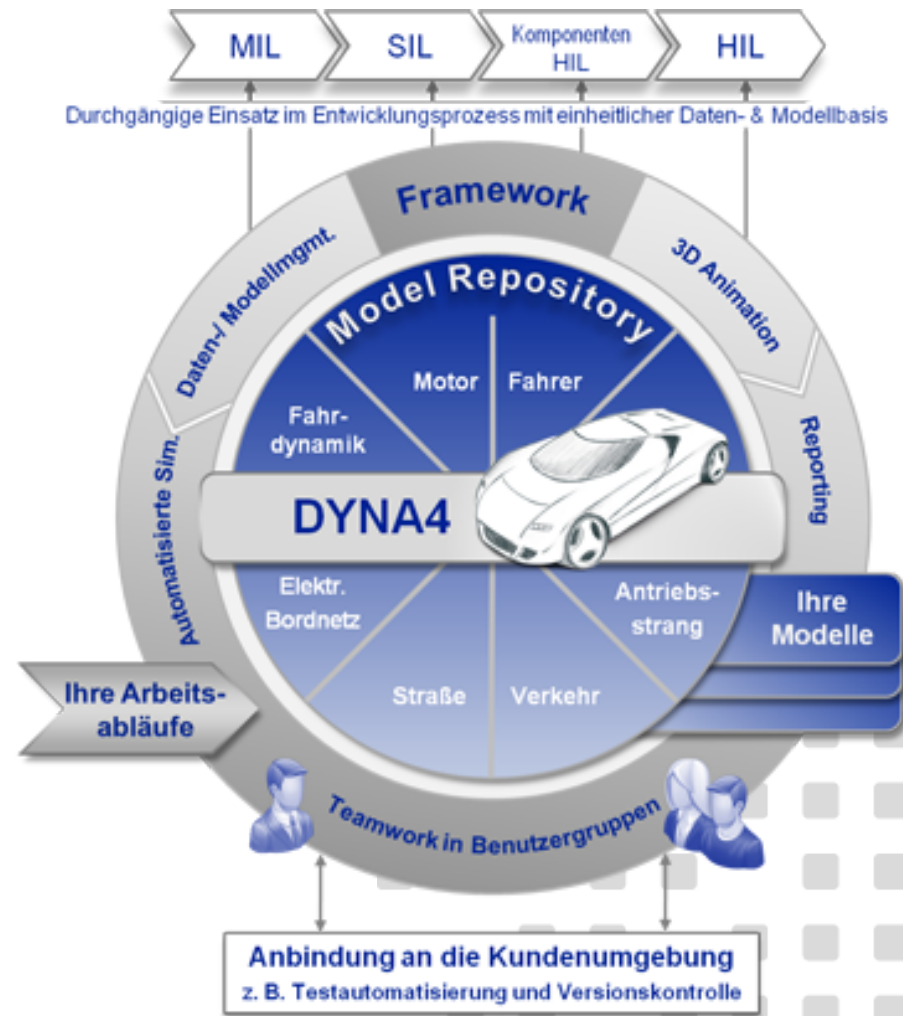


# DYNA4 Framework



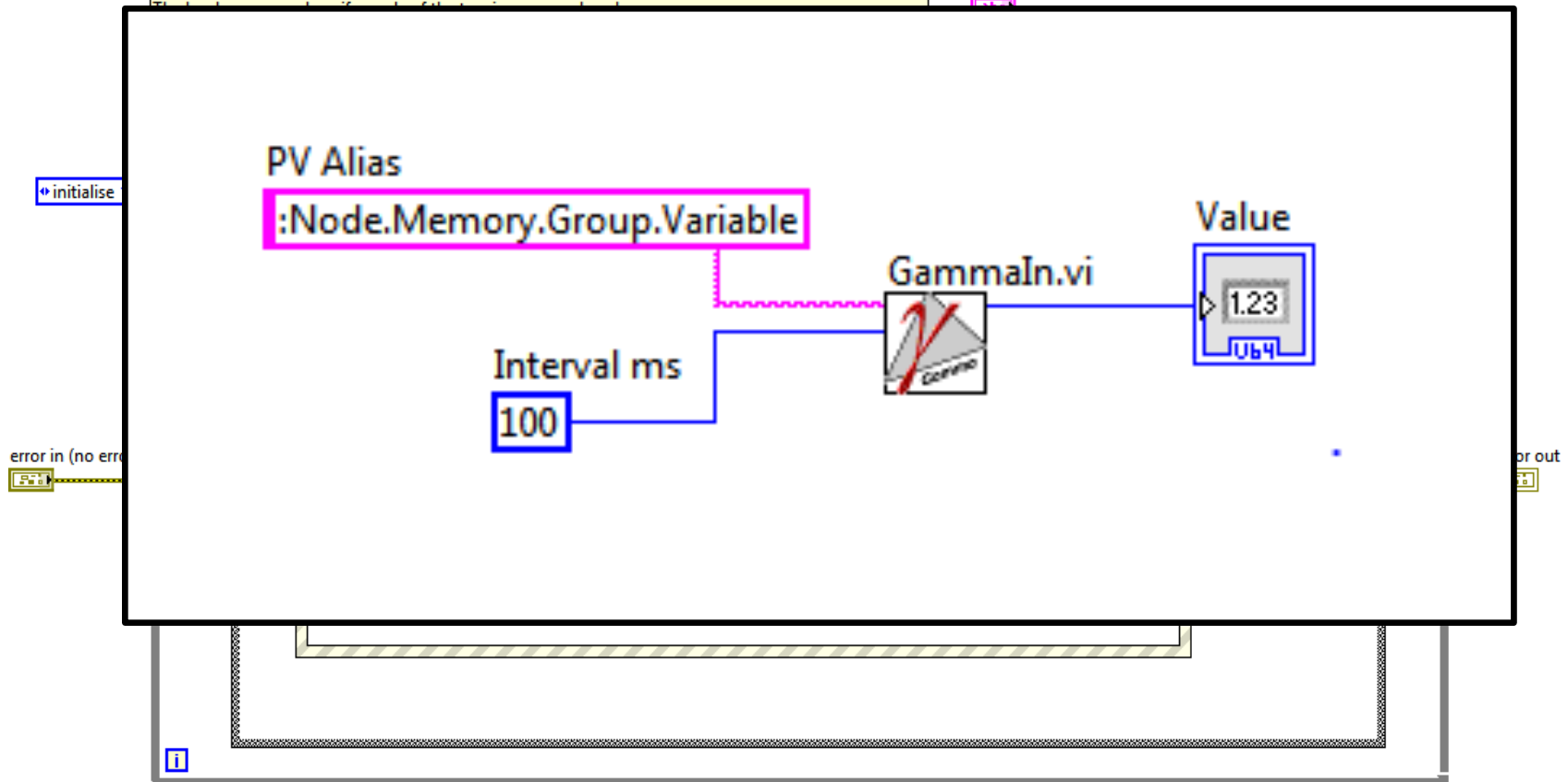
Modulares Simulationsframework  
mit offener Modellbibliothek und  
prozessorientierter Toolumgebung

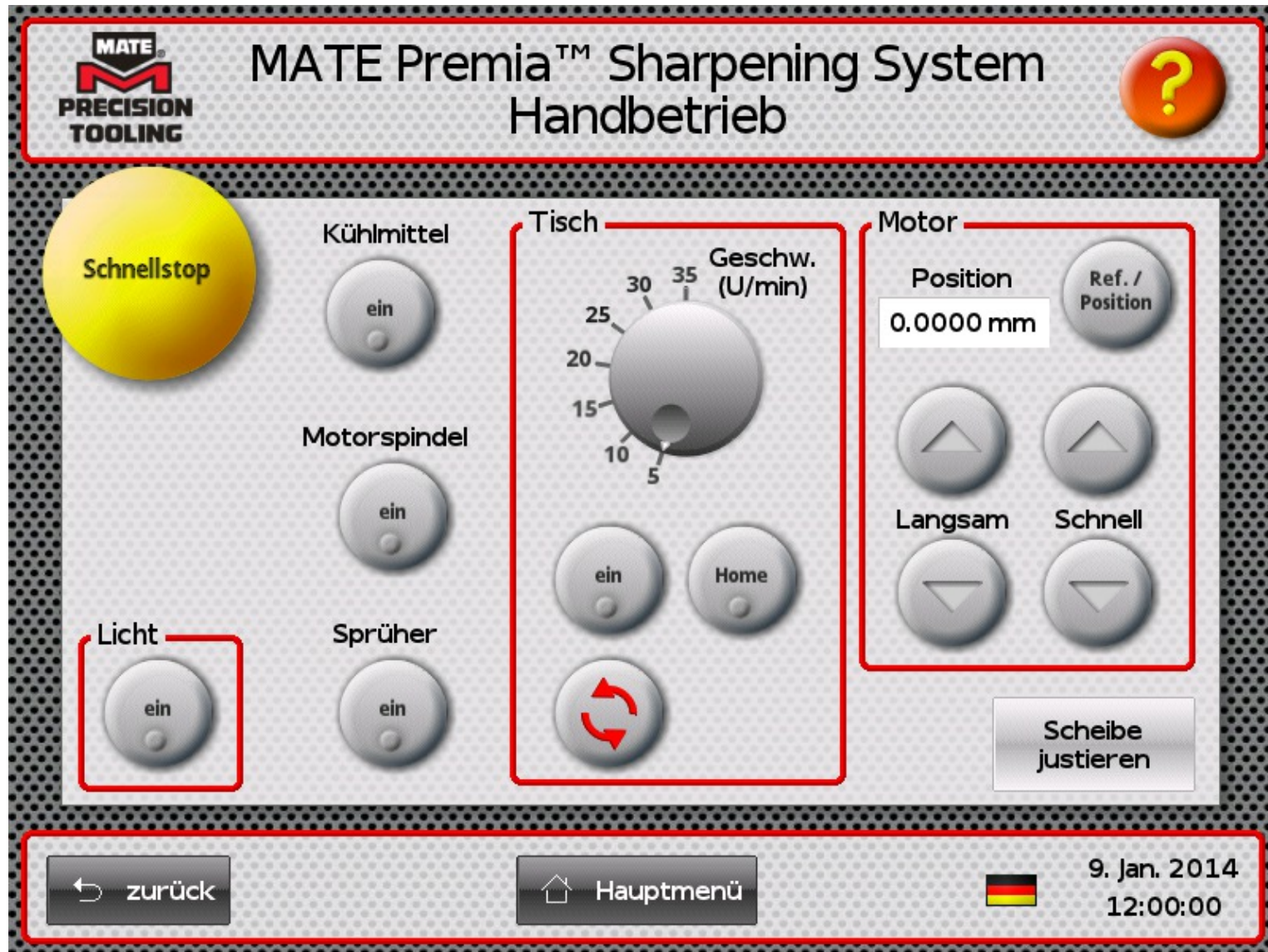
- Durchgängiges Management
- Unterstützung der Zusammenarbeit in Teams
- Tools für automatisierte Simulation
- Flexible Anpassung an Arbeitsprozesse
- Schnittstellen zur Infrastruktur, Testautomatisierung, Versionskontrolle, Reporting



This example is one method of saving and restoring the status of a hierarchy tree in a 1D Boolean array.

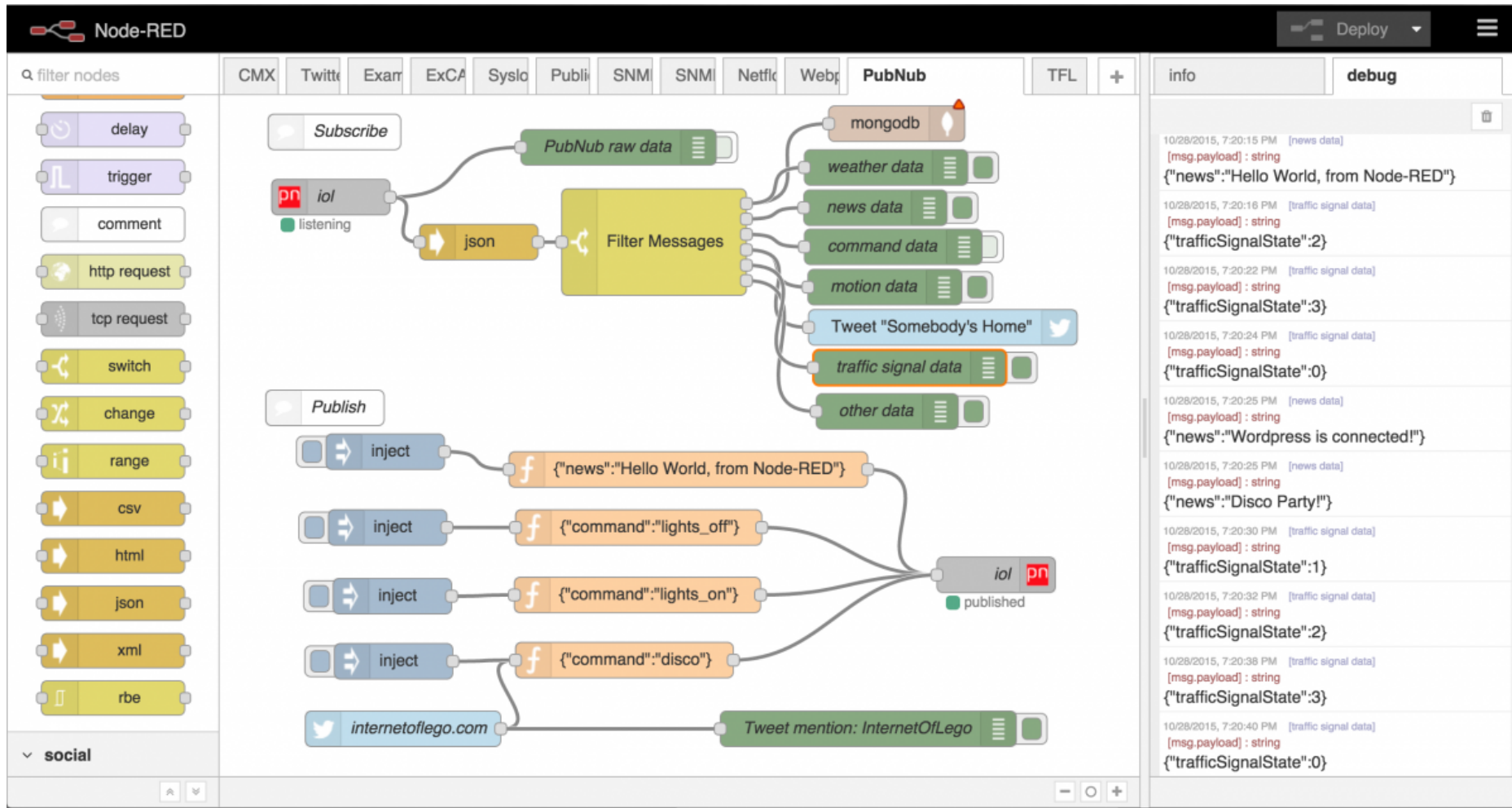
Tree







# Rapid Prototyping: Node Red von IBM



# **Middlewaretechnologien: Ein weiterer Schritt in der Evolution von Testgeräten!**

