FAIL*

Fault Injection Leveraged

Martin Hoffmann

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Motivation

DanceOS Evolution

CoRed → FT-eCos, ... → Evaluation: What could possibly go wrong?!

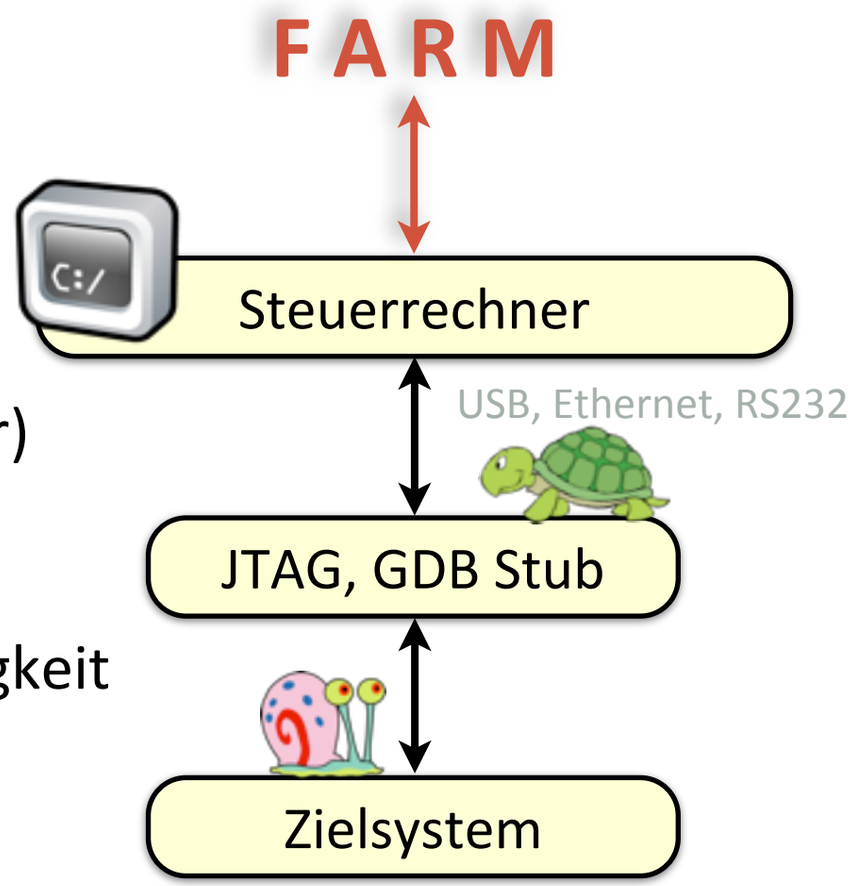
- Dynamische Analyse
  - Propagation bzw. Containment von Fehlern
  - Wirksamkeit von Fehlererkennung / Fehlertoleranz
- Fehlerinjektionsplattform im Rahmen des DanceOS Projektes
  - Verschiedene Architekturen
  - Mächtig, schnell, wiederverwendbar

Agenda

- Grundlagen Fehlerinjektion
- Praktische Probleme
- Lösung FAIL*

Grundlagen zur Fehlerinjektion

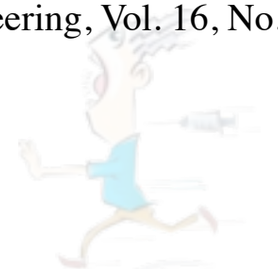
- ***FARM*** Modell[1]
  - **F**ault - **F**ehlerraum (Ort, Zeit, Typ)
  - **A**ctivation - **A**ktivierungsmuster (Eingabeparameter)
  - **R**eadout - E**r**fassung der Ergebnisse
  - **M**easure - **M**aß der Fehleranfälligkeit / Zuverlässigkeit



[1]J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.-C. Laprie, E. Martins, D. Powell, *Fault Injection for Dependability Validation: A Methodology and some Applications*, IEEE Transactions on Software Engineering, Vol. 16, No. 2, February 1990, pp. 166-182

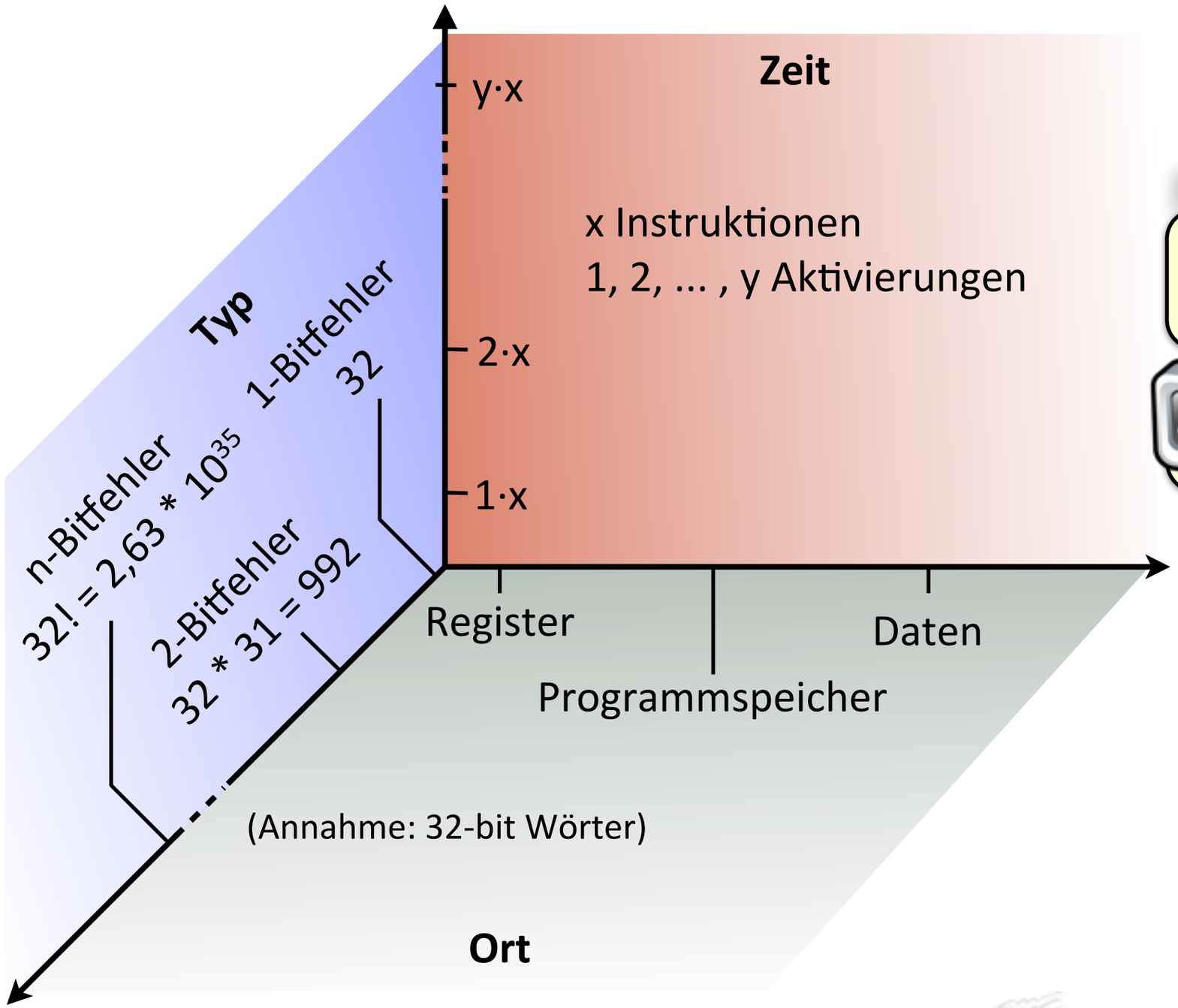
Theorie

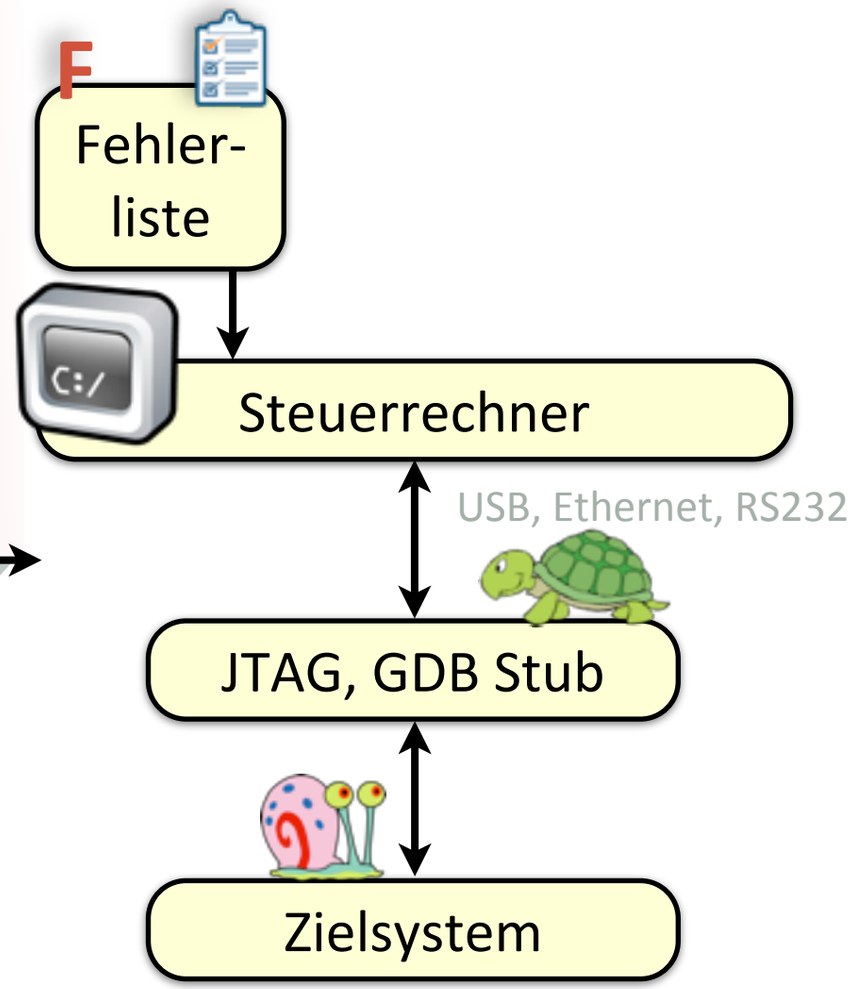
Praxis



FAIL*

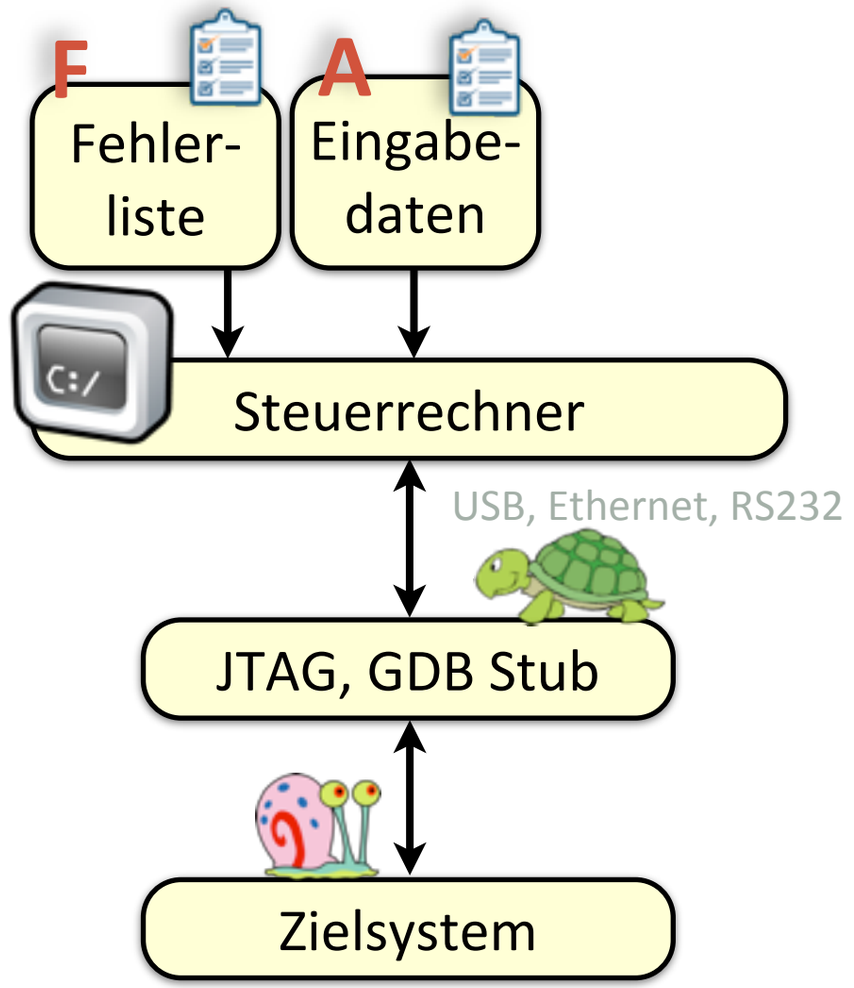
Fehlerraum


Zeit
y·x
2·x
1·x
x Instruktionen
1, 2, ... , y Aktivierungen
Typ
1-Bitfehler
32
n-Bitfehler
32! = 2,63 * 10^35
2-Bitfehler
32 * 31 = 992
Register
Daten
Programmspeicher
(Annahme: 32-bit Wörter)
Ort


F
Fehler-
liste
C:/
Steuerrechner
USB, Ethernet, RS232
JTAG, GDB Stub
Zielsystem

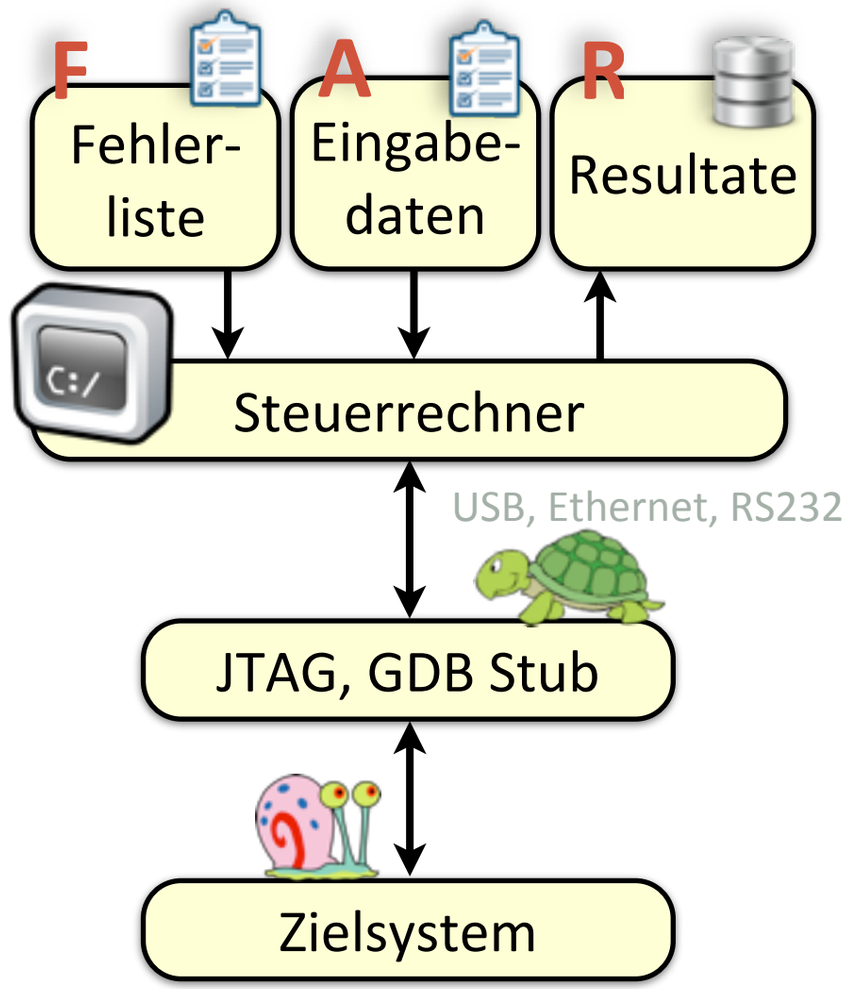
Aktivierungsmuster

- Ausführung des Aufgabensystems
- Setzen von Eingabeparametern
  - Sensordaten
  - Netzwerkpakete
  - Unterbrechungen
- Erfassung eines Referenzlaufs (Golden Run)
- Injektion von Fehlern



Erfassung der Ergebnisse

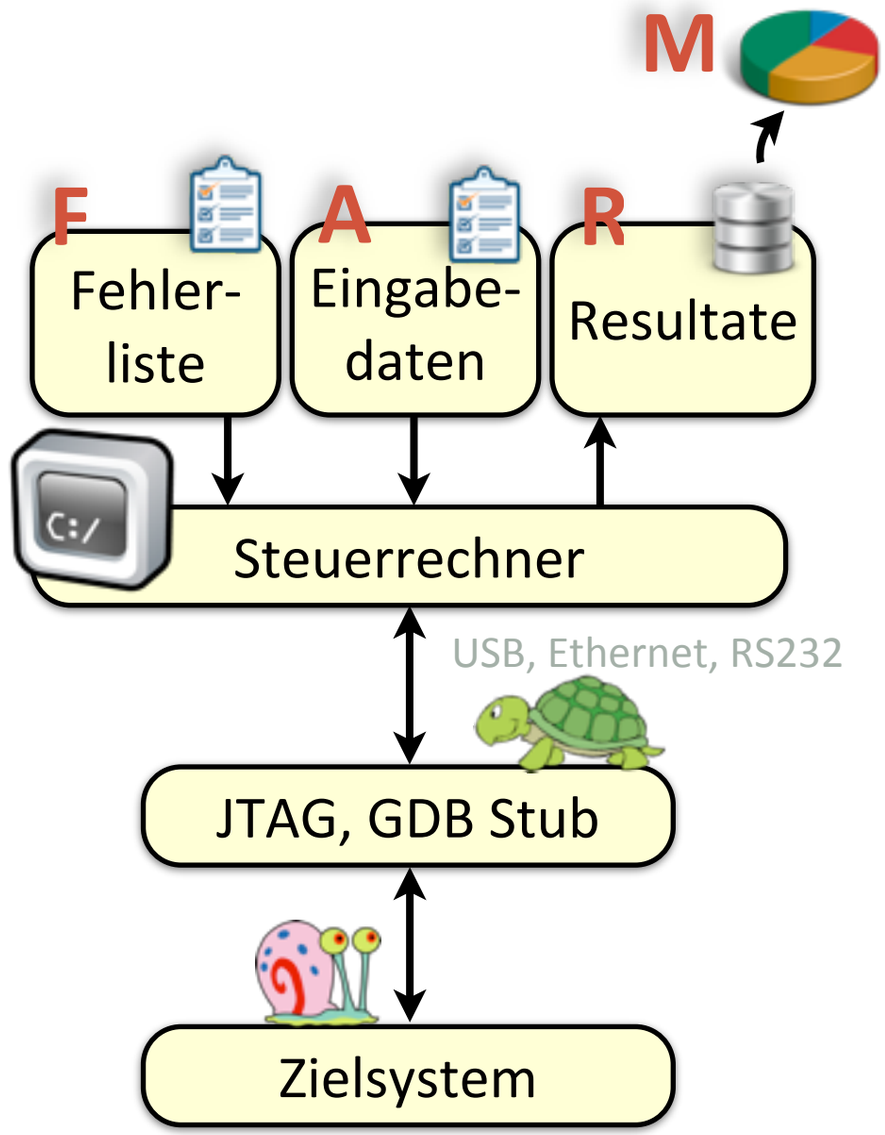
- Abhängig von der Mächtigkeit der Zielplattform
- Beispiele:
  - Fehlerparameter (Ort, Zeit, Typ)
  - Systemkontext (Register-, Speicherauszug)
  - Ausführungszeit
  - Rückgabewerte (Rechenergebnisse, etc.)
  - Fehlererkennungsmechanismen



Maß

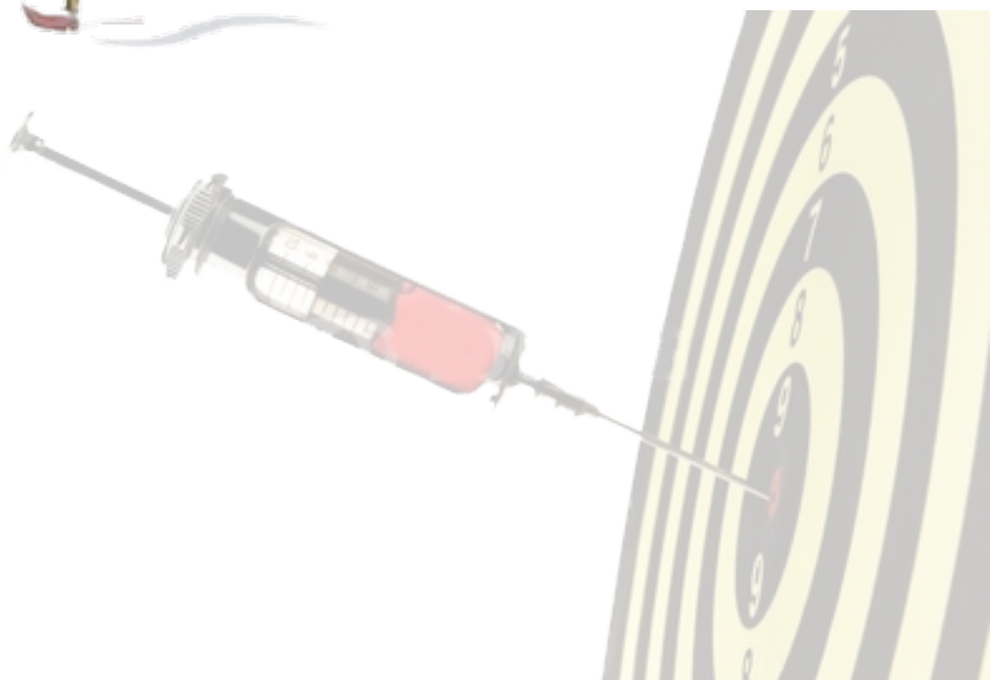
- Interpretation der Ergebnisse
  - Fehlererkennungsrate
  - Maskierungsrate
    - Erholungszeit
- Nicht beschränkt auf Fehlerinjektion

  → Integrationstest

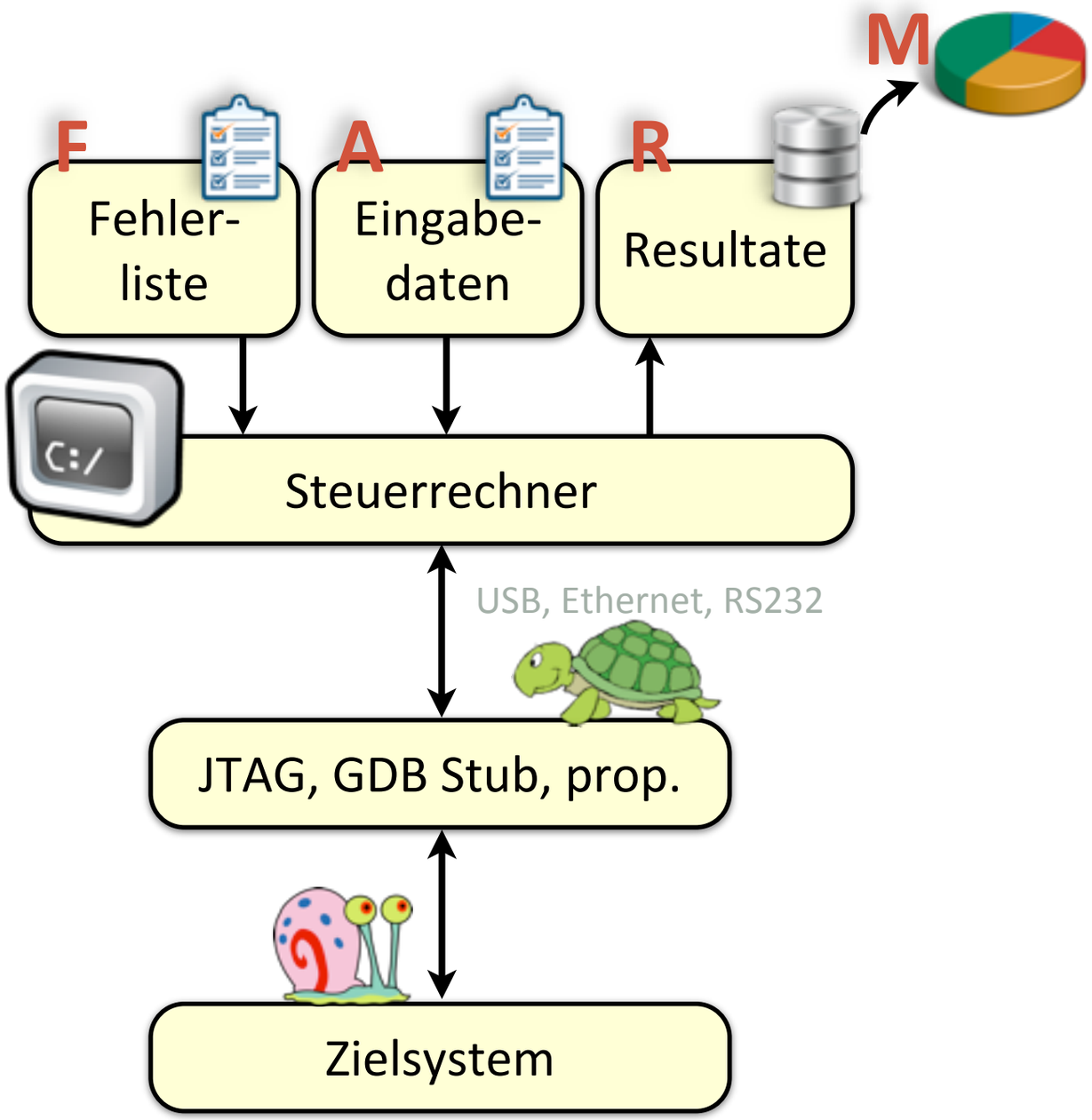


Agenda

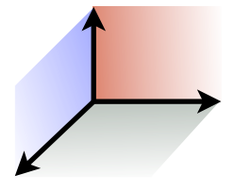
- Grundlagen Fehlerinjektion
- Praktische Probleme
- Lösung FAIL*



“Lessons Learned” CoRed Evaluation



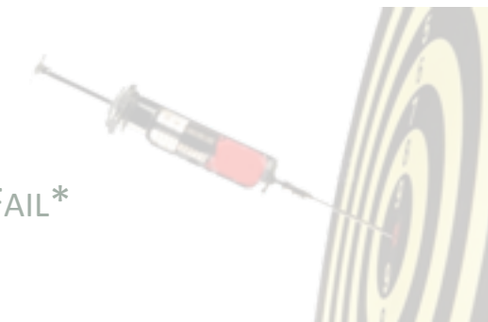
- Riesiger Fehlerraum



- Experimentbeschreibung
  - nicht standardisiert
  - kaum wiederverwendbar
- Kampagnendauer
  - typisch: 1s / Experiment
  - 400 000 Exp. → 110 h

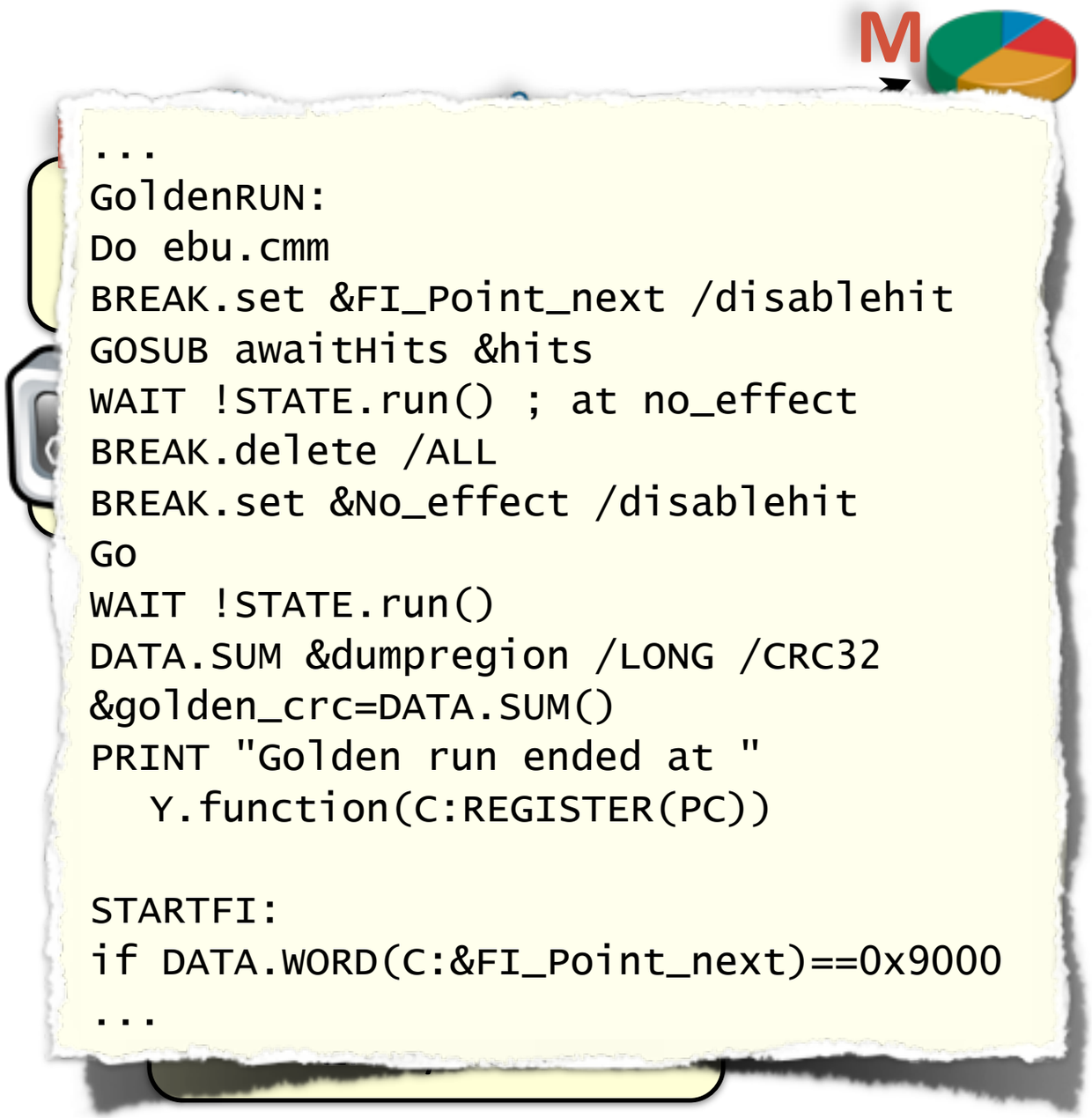
Theorie

Praxis

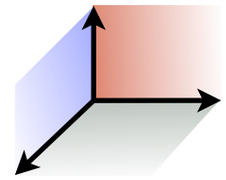


FAIL*

“Lessons Learned” CoRed Evaluation



- Riesiger Fehlerraum



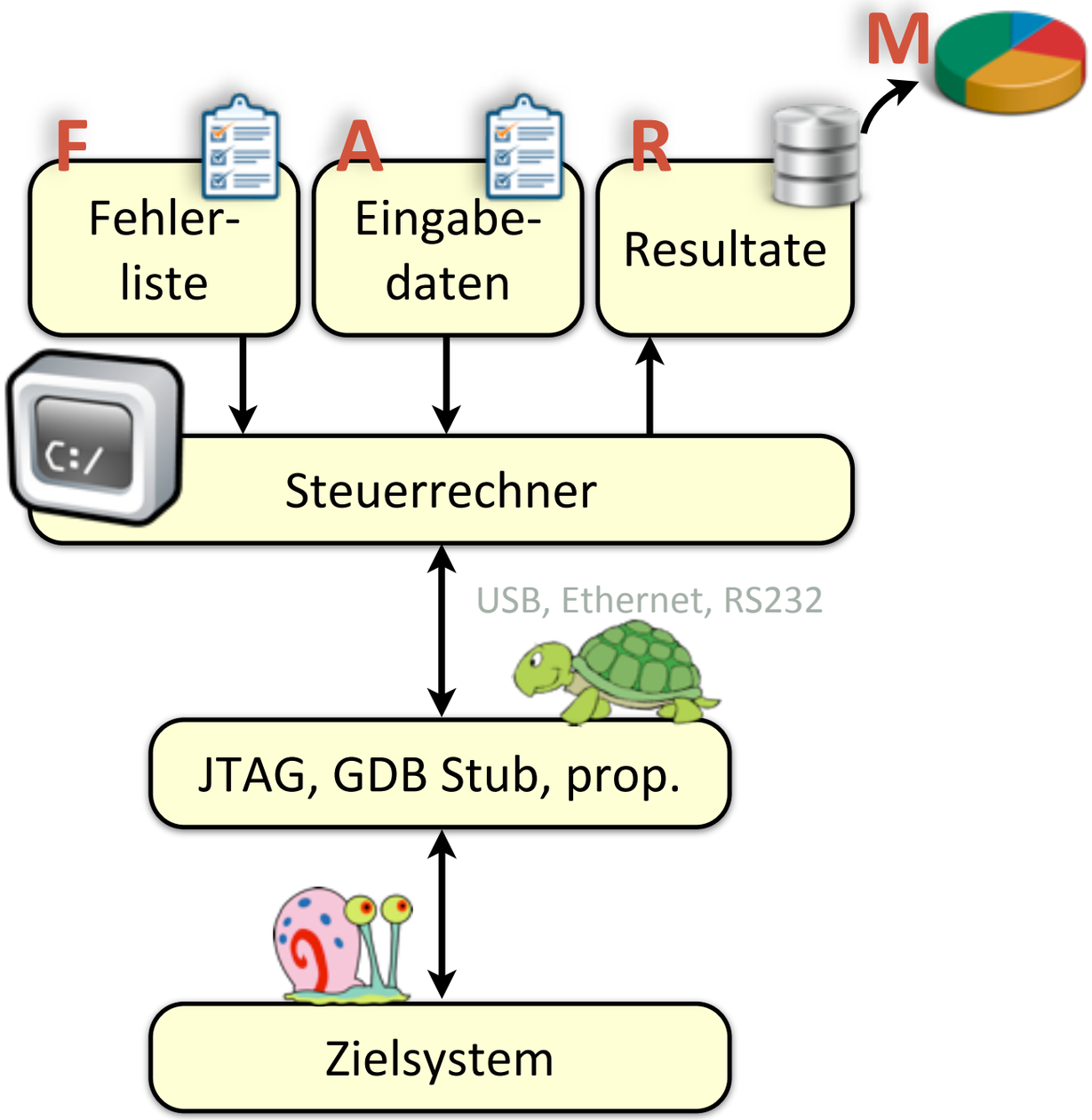
- Experimentbeschreibung
  - nicht standardisiert
  - kaum wiederverwendbar
- Kampagnendauer
  - typisch: 1s / Experiment
  - 400 000 Exp. → 110 h

Theorie

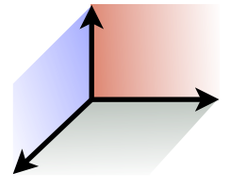
Praxis

FAIL*

“Lessons Learned” CoRed Evaluation



- Riesiger Fehlerraum

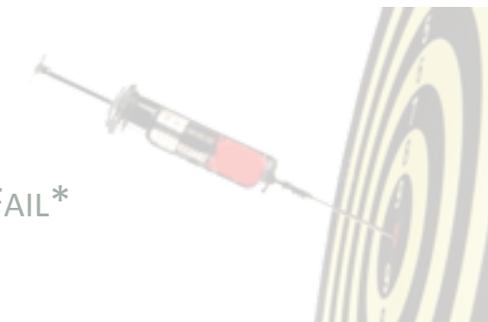


- Experimentbeschreibung
  - nicht standardisiert
  - kaum wiederverwendbar

- Kampagnendauer
  - typisch: 1s / Experiment
  - 400 000 Exp. → 110 h

Theorie

Praxis



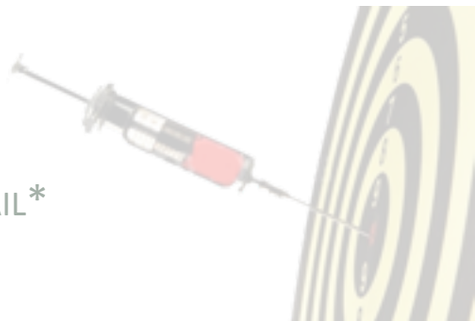
FAIL*

Probleme existierender Lösungen

- Viele existierende Fehlerinjektionswerkzeuge, aber:
  - Jeweils unterschiedliche Zielplattformen, Fehlermodelle, Experimentbeschreibungen, Ergebnisauswertung
  - z.B. FAUmachine, Qinject, Goofi, FTAPE, Ferrari, Fiat, Xception, Mefisto, ... u.v.m. *Namenlose*
- Zum Teil Abspaltungen von existierenden Emulatoren (Qemu -> Qinject)
  - Basierend auf veralteter Emulator Version
  - Wartung problematisch

Theorie

Praxis



FAIL*

Herausforderungen

- Unterstützung verschiedener Architekturen (x86, ARM, Sparc, ...)
- Wiederverwendbare, mächtige Experimentbeschreibung
- Flexible Auswertung der Ergebnisse
- Praktikable Kampagnendauer

DanceOS Evolution:

CoRed → FT-eCos, ... → **Fail***

Theorie

Praxis

FAIL*

Agenda

- Grundlagen Fehlerinjektion
- Praktische Probleme
- Lösung FAIL*

Entwurfsentscheidungen

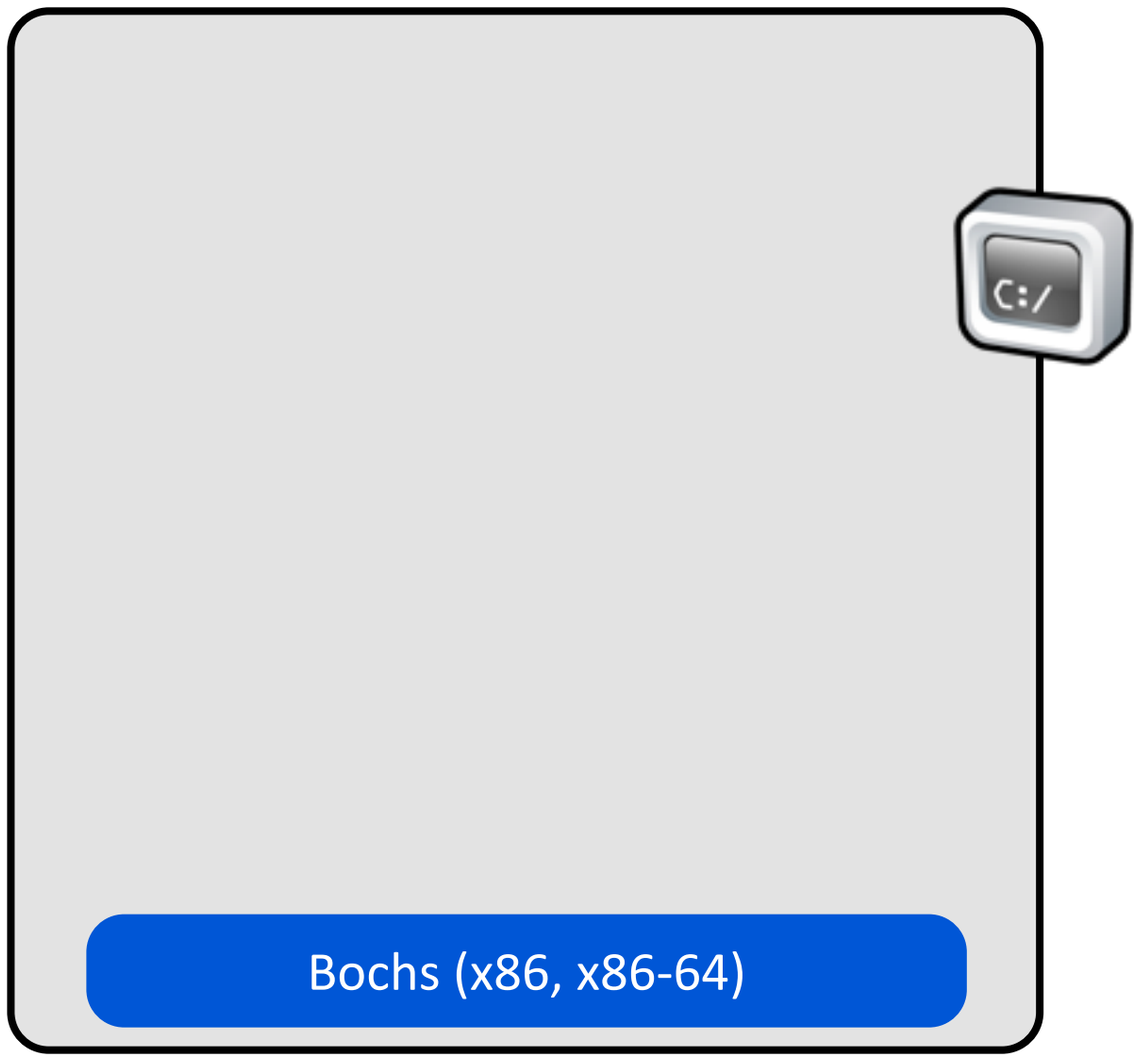
- Ziel: Flexibilität
- Erweiterung existierender virtueller Plattformen (Bochs, gem5, ...)
  - Aktuelle und gewartete Softwarebasis
  - Schnelle Experimente
  - Volle Kontrolle der Umgebung, voller Zugriff
- Abstrakte Schnittstelle zur Plattform
  - Wiederverwendbare Experimentbeschreibung
  - “Leitfaden” für Erweiterungen (neue Prozessoren, etc)

Theorie

Praxis

FAIL*

Erweiterung existierender Plattformen



Experiment-specific user-defined

Existing Emulator

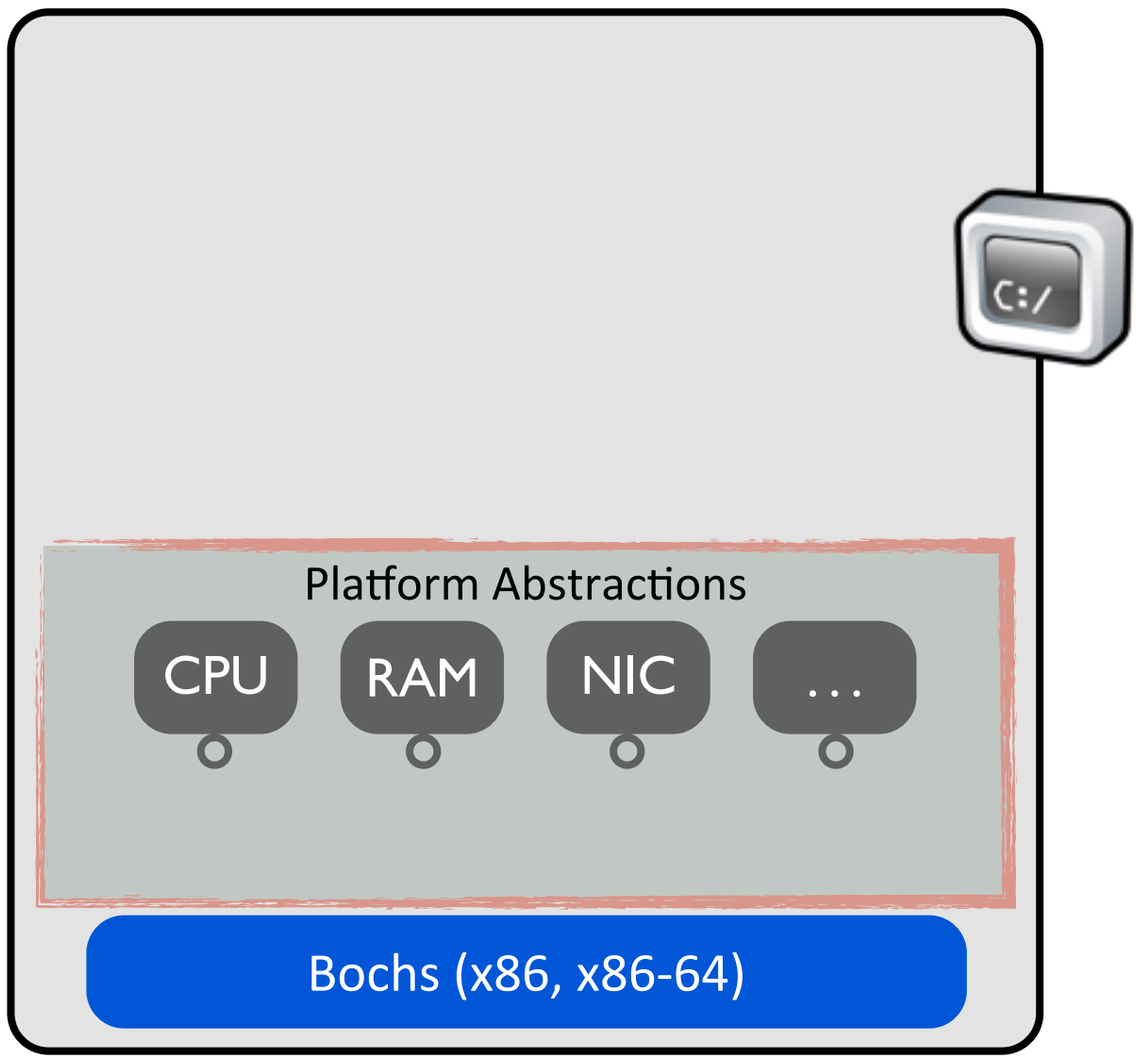
Fail* Code

Theorie

Praxis

FAIL*

Erweiterung existierender Plattformen



- Abstrakte Plattformbeschreibung

Experiment-specific user-defined

Existing Emulator

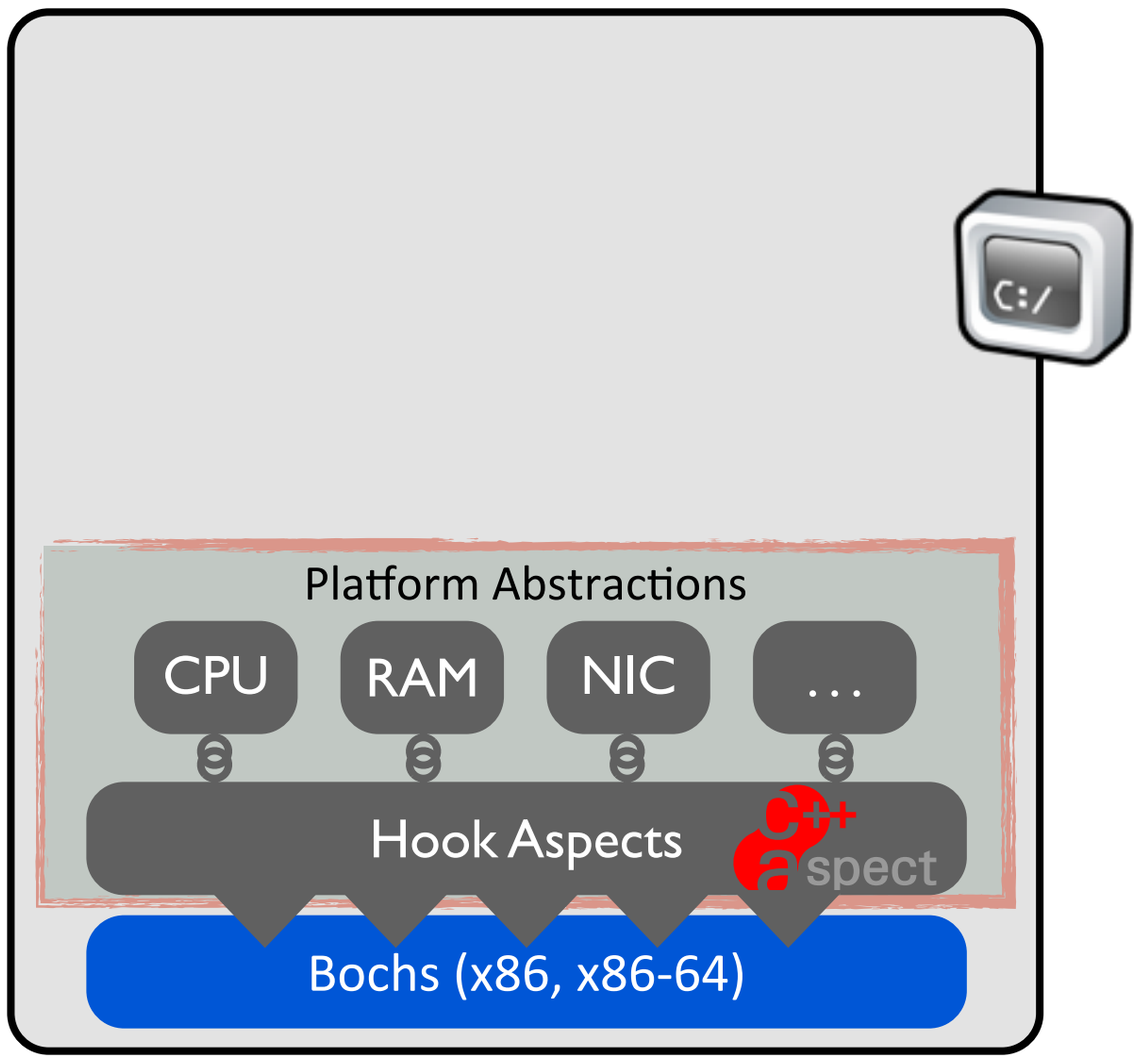
Fail* Code

Theorie

Praxis

FAIL*

Erweiterung existierender Plattformen



- Abstrakte Plattformbeschreibung

Experiment-specific user-defined

Existing Emulator

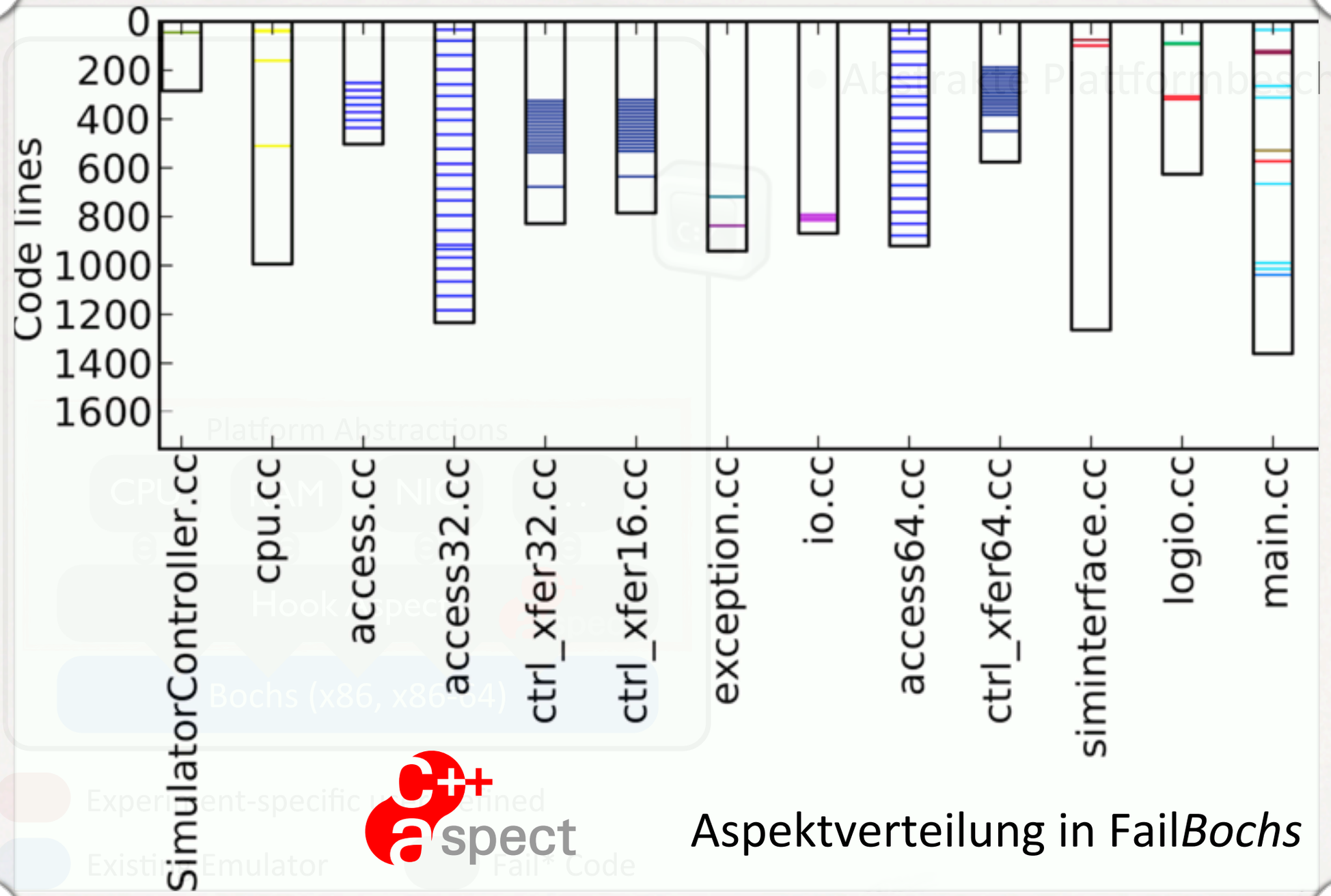
Fail* Code

Theorie

Praxis

FAIL*

Erweiterung existierender Plattformen

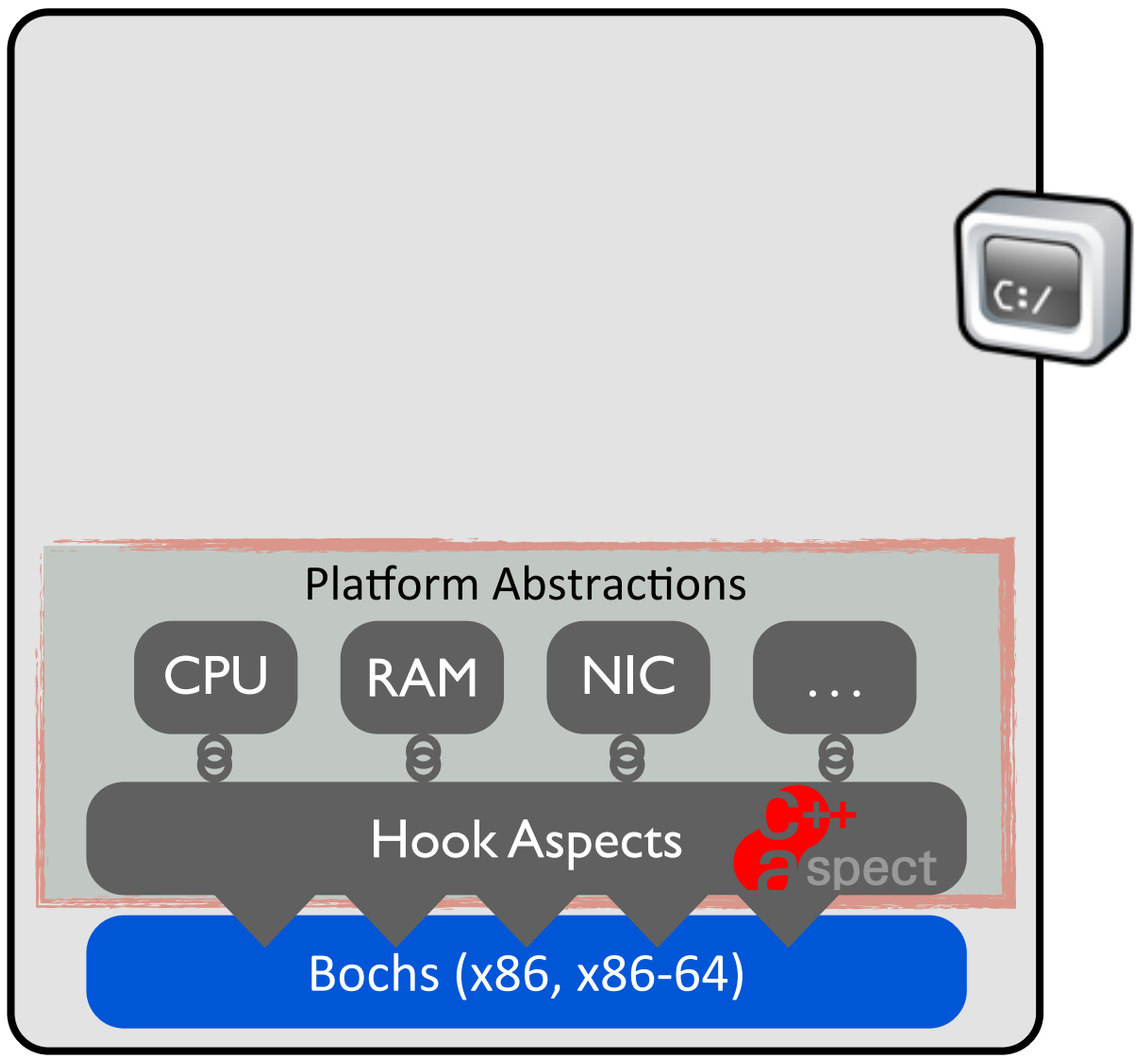


reibung

aspect

Aspektverteilung in Fail*Bochs*

Erweiterung existierender Plattformen



- Abstrakte Plattformbeschreibung

Experiment-specific user-defined

Existing Emulator

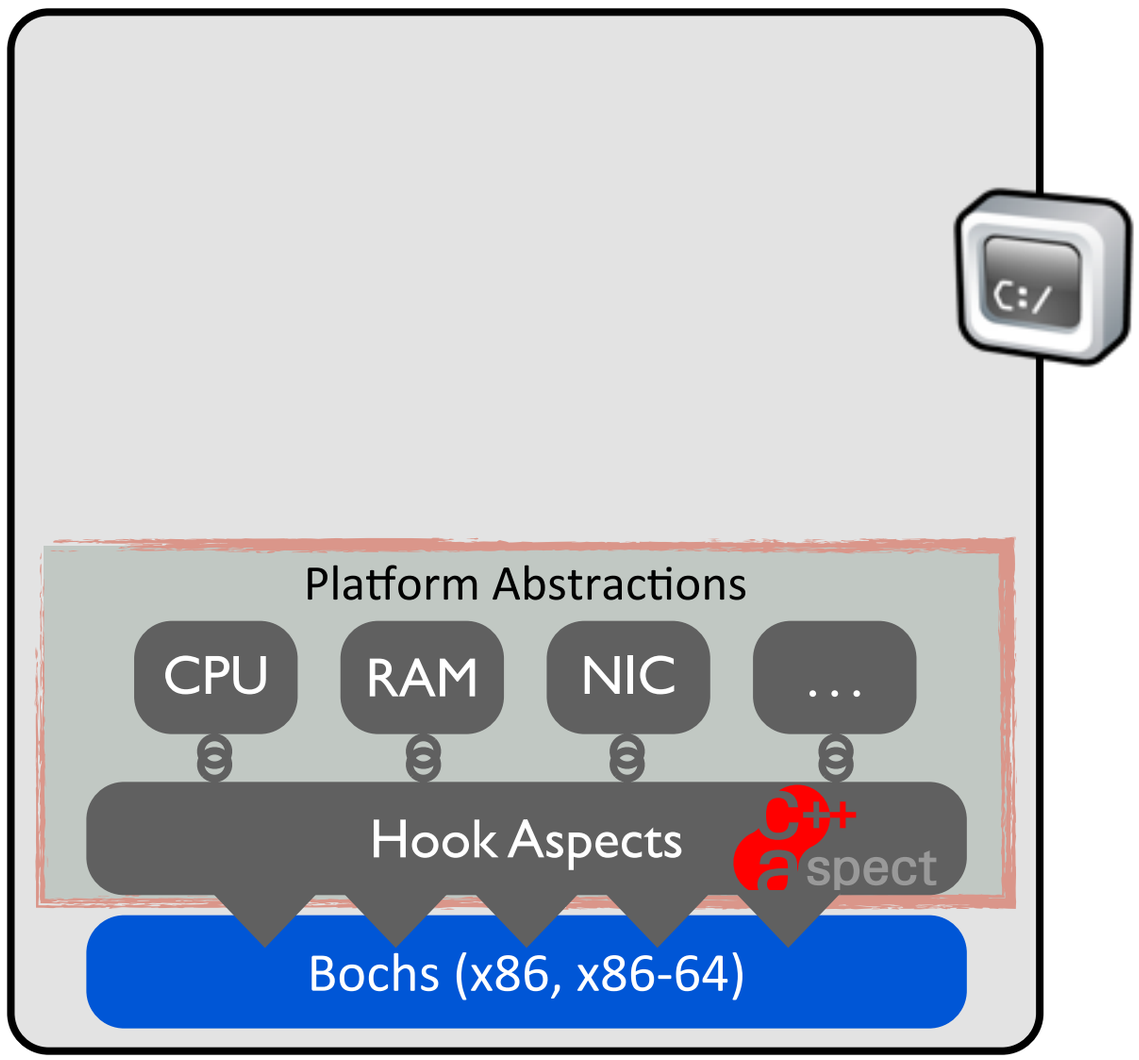
Fail* Code

Theorie

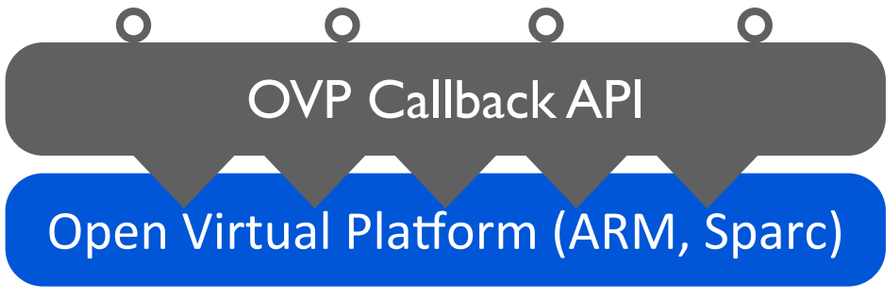
Praxis

FAIL*

Erweiterung existierender Plattformen



- Abstrakte Plattformbeschreibung



Experiment-specific user-defined

Existing Emulator

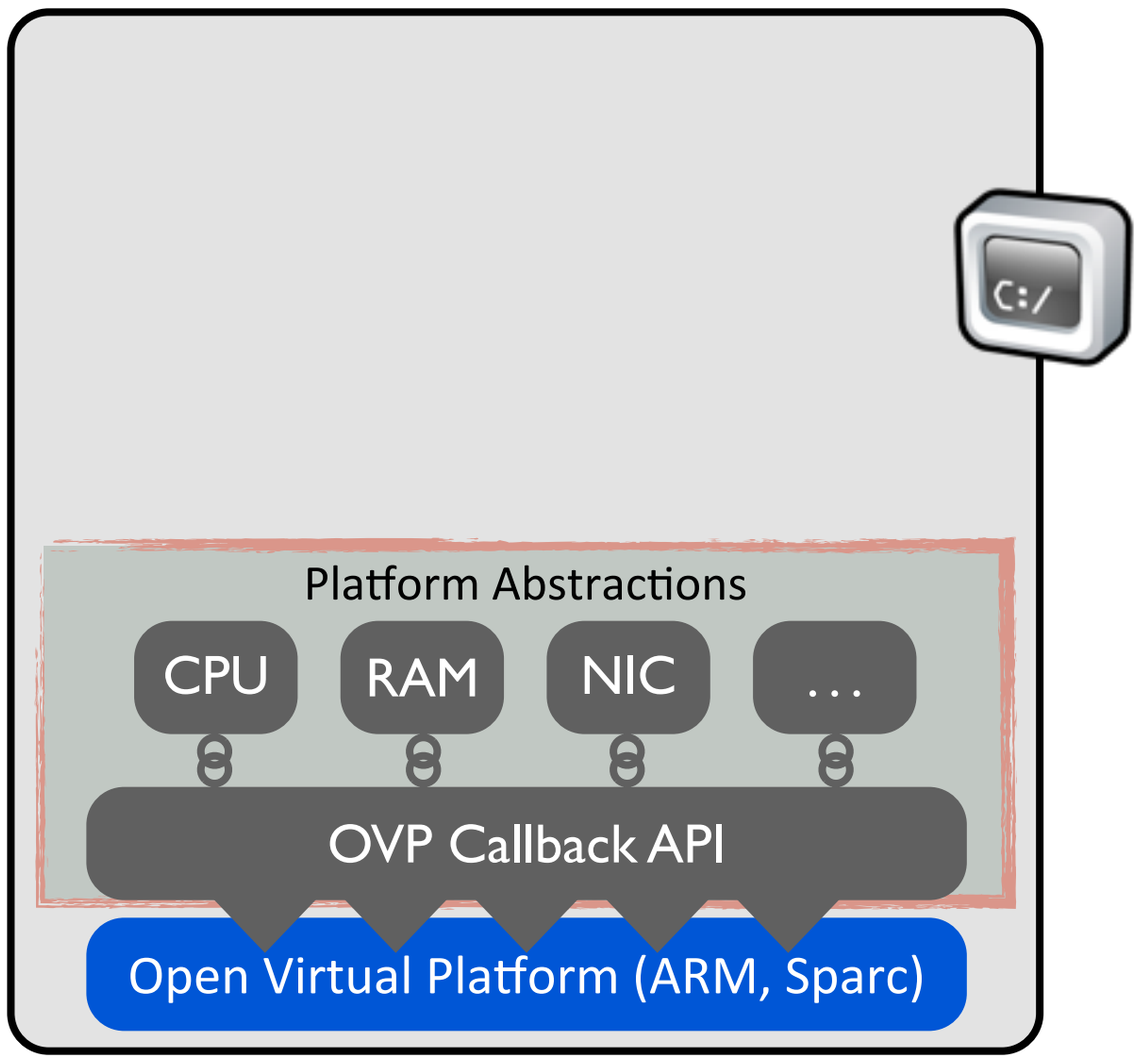
Fail* Code

Theorie

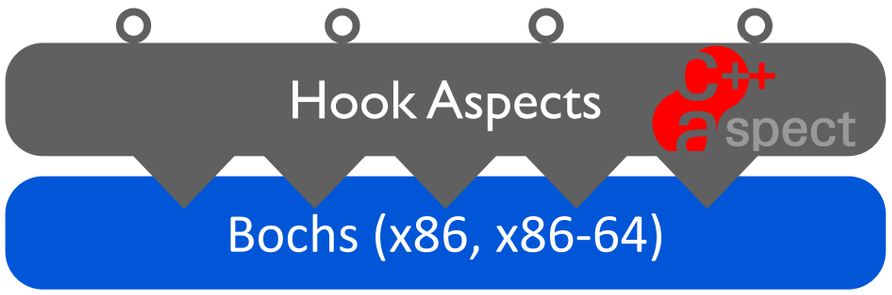
Praxis

FAIL*

Erweiterung existierender Plattformen



- Abstrakte Plattformbeschreibung



Experiment-specific user-defined

Existing Emulator

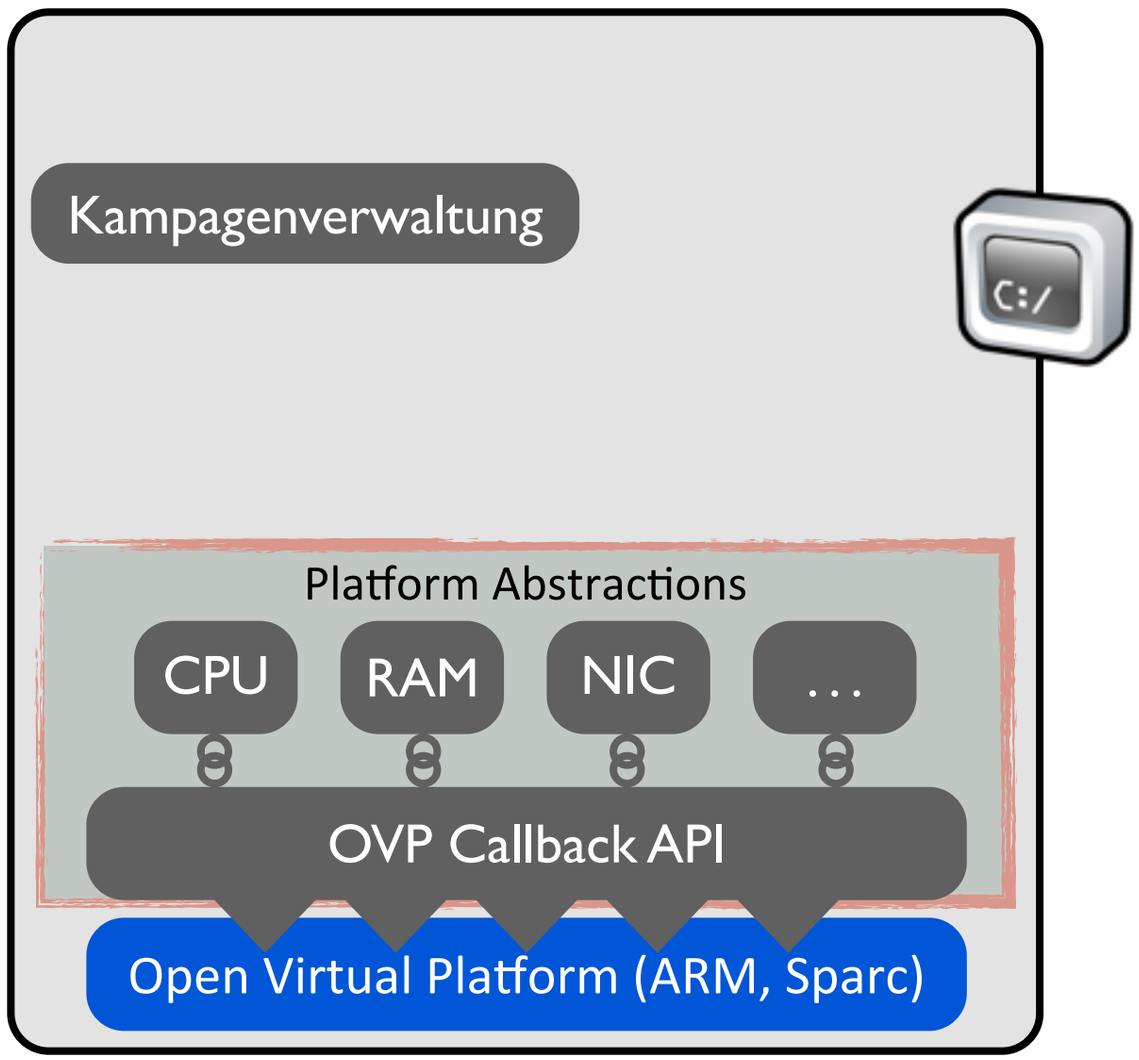
Fail* Code

Theorie

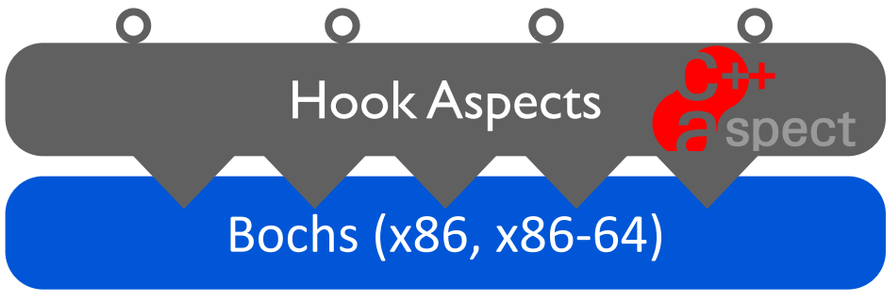
Praxis

FAIL*

Erweiterung existierender Plattformen



- Abstrakte Plattformbeschreibung
- Aufteilung Kampagne/Experiment
  - 1 Kampagne -> N Experimente



Experiment-specific user-defined

Existing Emulator

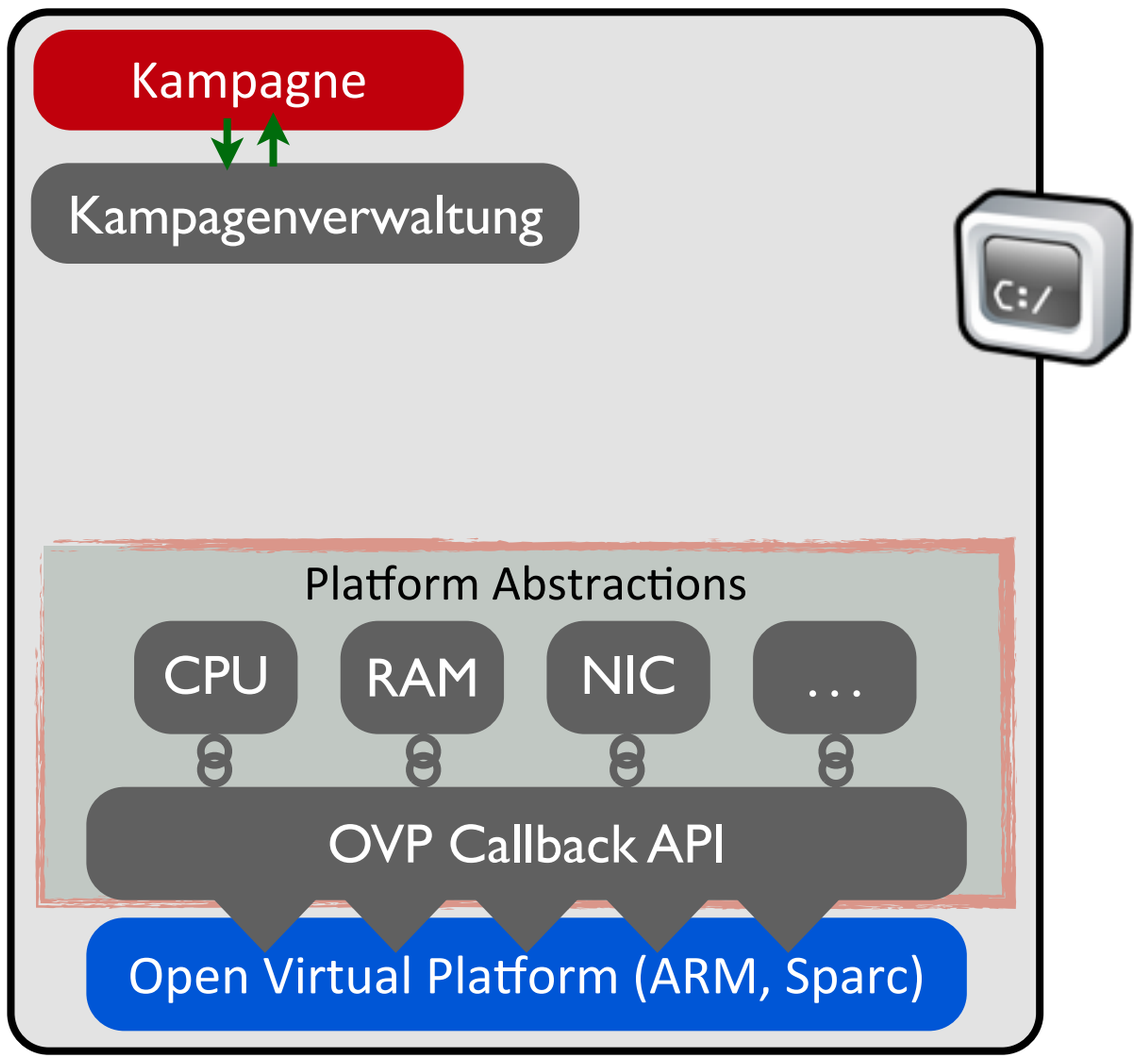
Fail* Code

Theorie

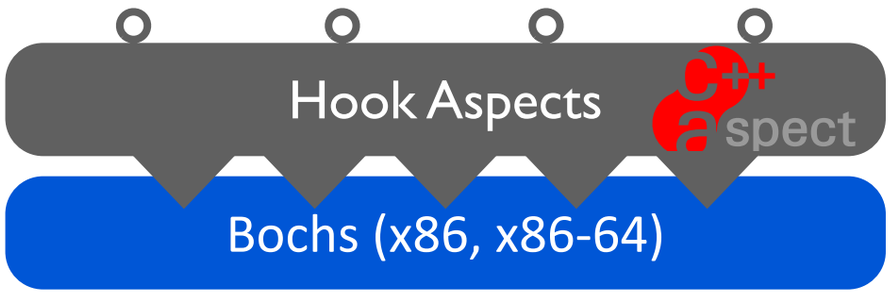
Praxis

FAIL*

Erweiterung existierender Plattformen



- Abstrakte Plattformbeschreibung
- Aufteilung Kampagne/Experiment
  - 1 Kampagne -> N Experimente



Experiment-specific user-defined

Existing Emulator

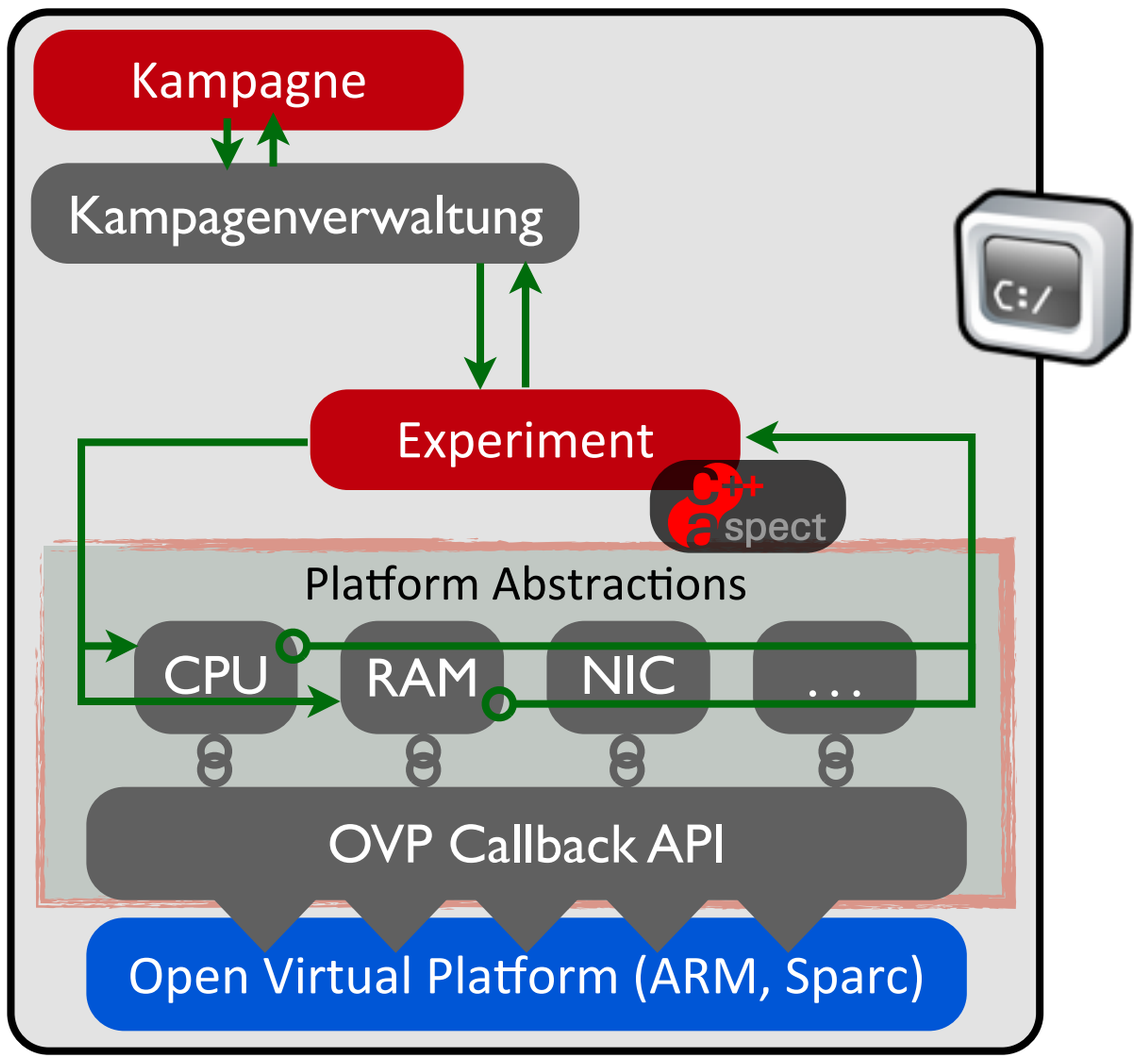
Fail* Code

Theorie

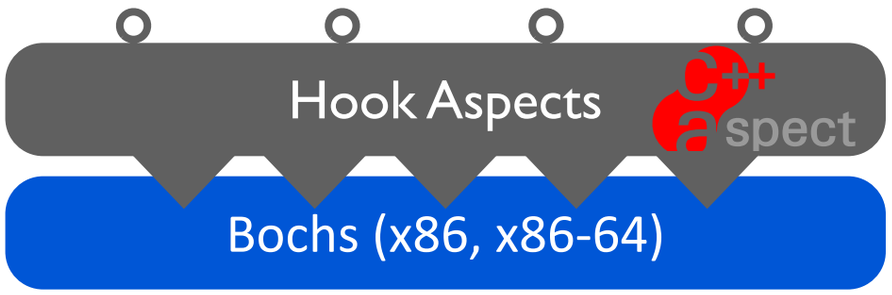
Praxis

FAIL*

Erweiterung existierender Plattformen



- Abstrakte Plattformbeschreibung
- Aufteilung Kampagne/Experiment
  - 1 Kampagne -> N Experimente



Experiment-specific user-defined

Existing Emulator

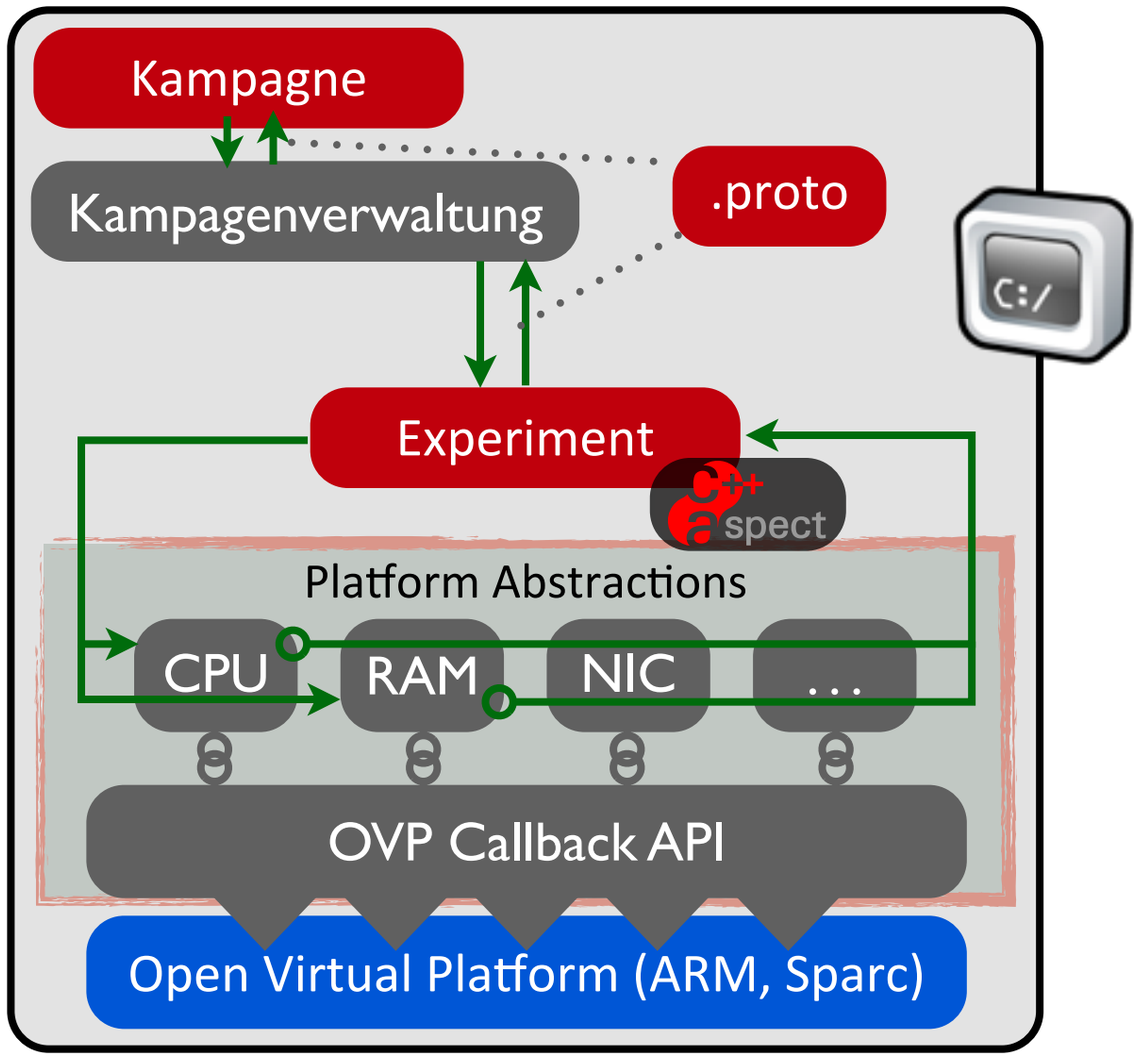
Fail* Code

Theorie

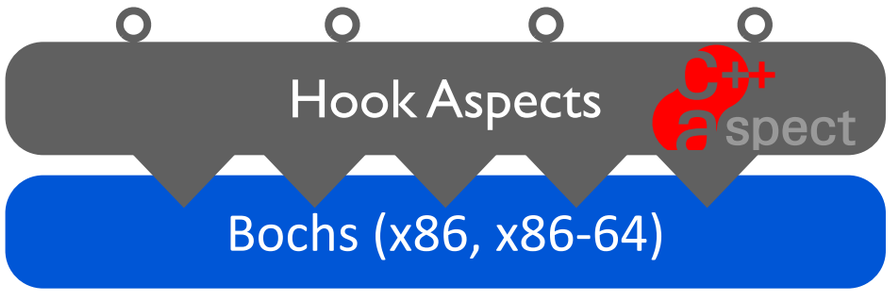
Praxis

FAIL*

Erweiterung existierender Plattformen



- Abstrakte Plattformbeschreibung
- Aufteilung Kampagne/Experiment
  - 1 Kampagne -> N Experimente
- Nachrichtenaustausch per Google Protocol Buffers



Experiment-specific user-defined

Existing Emulator

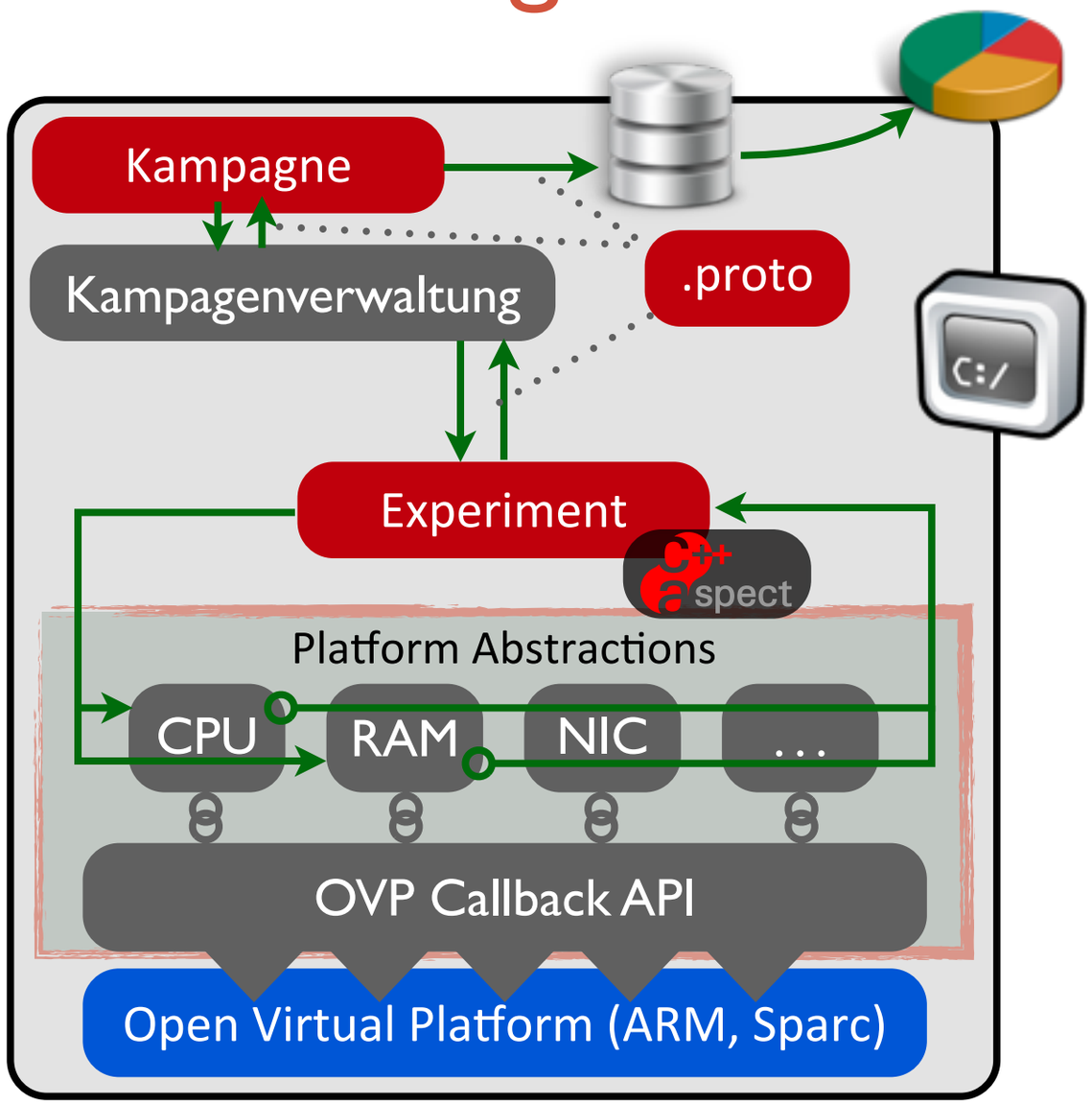
Fail* Code

Theorie

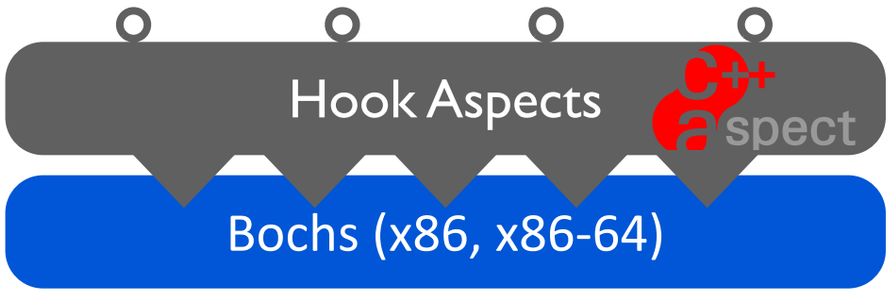
Praxis

FAIL*

Erweiterung existierender Plattformen



- Abstrakte Plattformbeschreibung
- Aufteilung Kampagne/Experiment
  - 1 Kampagne -> N Experimente
- Nachrichtenaustausch per Google Protocol Buffers
- beliebige Nachverarbeitung



Experiment-specific user-defined

Existing Emulator

Fail* Code

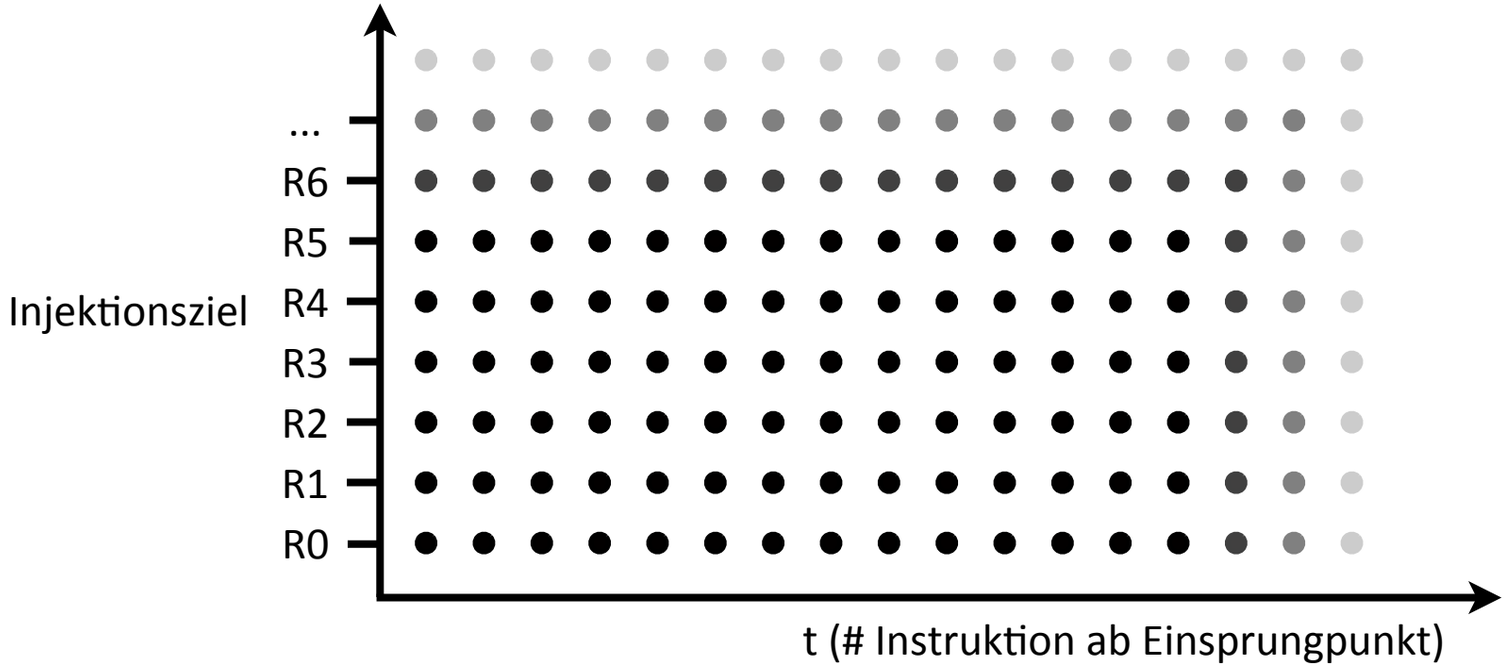
Theorie

Praxis

FAIL*

Beispielexperiment

- Analysierte Software: **einzelne C-Funktion**
- Fehlermodell: **Einzelbitkipper**
- Überdeckung: **Alle Register, Bits, Instruktionen**



Theorie

Praxis

FAIL*

Beispielexperiment - Nachrichtenbeschreibung

- Google Protocol Buffers
- Erzeugt Datenstrukturen mit Zugriffsmethoden, (De-) Serialisierung
  - C++, Java, Python, Matlab, Perl, R, Lua, Scala, Haskell, Erlang, Ruby, OCaml, Visual Basic, ...

FaultCoverageMsg.proto

```
message FaultCoverageMsg {
   // input parameters
   required int32 instr_offset = 1;
   required int32 bitpos = 2;
   required int32 inject_register = 3;
   // results
   enum ResultType {
      LOG_NORMAL = 1;
      LOG_TRAP = 2;
      LOG_TIMEOUT = 3;
   }
   optional ResultType resulttype = 4;
   optional String details = 5;
}
```

Theorie

Praxis

FAIL*

Beispielexperiment - Kampagnenbeschreibung

- Erzeugt Parametersätze für Einzelexperimente
- Sammelt Ergebnisse

Kampagne

```
// iterate over all target registers
RegisterManager& rm = simulator.getRegisterManager();
for (RegisterManager::iterator it = rm.begin(); it != rm.end(); ++it) {
  Register *reg = &(*it);
  // iterate over all bit positions within this register
  for (int bitpos = 0; bitpos < reg->getWidth(); ++bitpos) {
    // iterate over all injection positions
    for (int instr = 0; instr < COVERAGE_NUMINSTR; ++instr) {
      FaultCoverageParam *p = new FaultCoverageParam;
      p->msg.set_instr_offset(instr);
      p->msg.set_bitpos(bitpos);
      p->msg.set_inject_register(reg->getId());
      campaignmanager.addParam(p); // enqueue parameter set
} } }
FaulCoverageParam *result; // gather results
while(result = static_cast<FaultCoverageParam *>(campaignmanager.getDone())){
   doSomeAftermathWith(result); // post process results
}
```

Theorie

Praxis

FAIL*

Beispielexperiment - Experimentbeschreibung

- Konsumiert (einzelnen) Parametersatz
- Eigentliche Fehlerinjektion

Experiment

```
// retrieve parameter set from campaign
jc.getParam(par);

// restore previously saved simulator state (from golden run):
// we're now at the entry of the analyzed function
simulator.restore("sav/p_entry.sav");
// breakpoint n instructions (def. in parameter set) in the future
BPEvent ev_fi_instr(ANY_ADDR, par.instr_offset());
addEventAndWait(&ev_fi_instr); // Switch to emulator Coroutine
// Emulator coroutine runs until breakpoint event ...
// FI: single bit-flip in register specified in parameter set
Register r = simulator.getRegisterManager().
             getRegister(par.inject_register());
r.setData(r.getData() ^ (1 << par.bitpos()));

// ...
```

Aktivierungs-muster

Fehlerinjektion

Theorie

Praxis

FAIL*

Beispielexperiment - Experimentbeschreibung

Experiment

```
// ...
// Aftermath: traps, timeout, or normal exit
TrapEvent ev_trap(ANY_TRAP);
addEvent(&ev_trap);
BPEvent ev_timeout(ANY_ADDR, 10000);
addEvent(&ev_timeout);
BPEvent ev_func_end(ADDR_FUNC_END);
addEvent(&ev_func_end);
// wait for function exit, trap or timeout
BaseEvent *ev = waitAny();
// store experiment result in parameter set object ...
if (ev == &id_func_end) {
        int result = simulator.abi_func_retval();
        par.set_resulttype(LOG_NORMAL);
        par.set_result(result);
} else if (ev == &ev_trap) {
        par.set_resulttype(LOG_TRAP);
} else if (ev == &ev_timeout) {
        par.set_resulttype(LOG_TIMEOUT);
}
// ... and communicate it back to the campaign controller
jc.sendResult(par);
```

Beobachtung

Zurücksenden der Ergebnisse

FAIL* Zwischenresümee

- ☑ Unterstützung verschiedener Architekturen
  - Erweiterung existierender virtueller Plattformen
- ☑ Wiederverwendbare, mächtige Experimentbeschreibung
  - “Simulator Abstraction Layer”
- ☑ Flexible Auswertung der Ergebnisse
  - Sprachunabhängige Nachrichtenbeschreibung (Protocol Buffers)

DanceOS Evolution:

CoRed → FT-eCos, ... → Fail* - → *Dependable CiAO*

↘ ☐ Praktikable Kampagnendauer

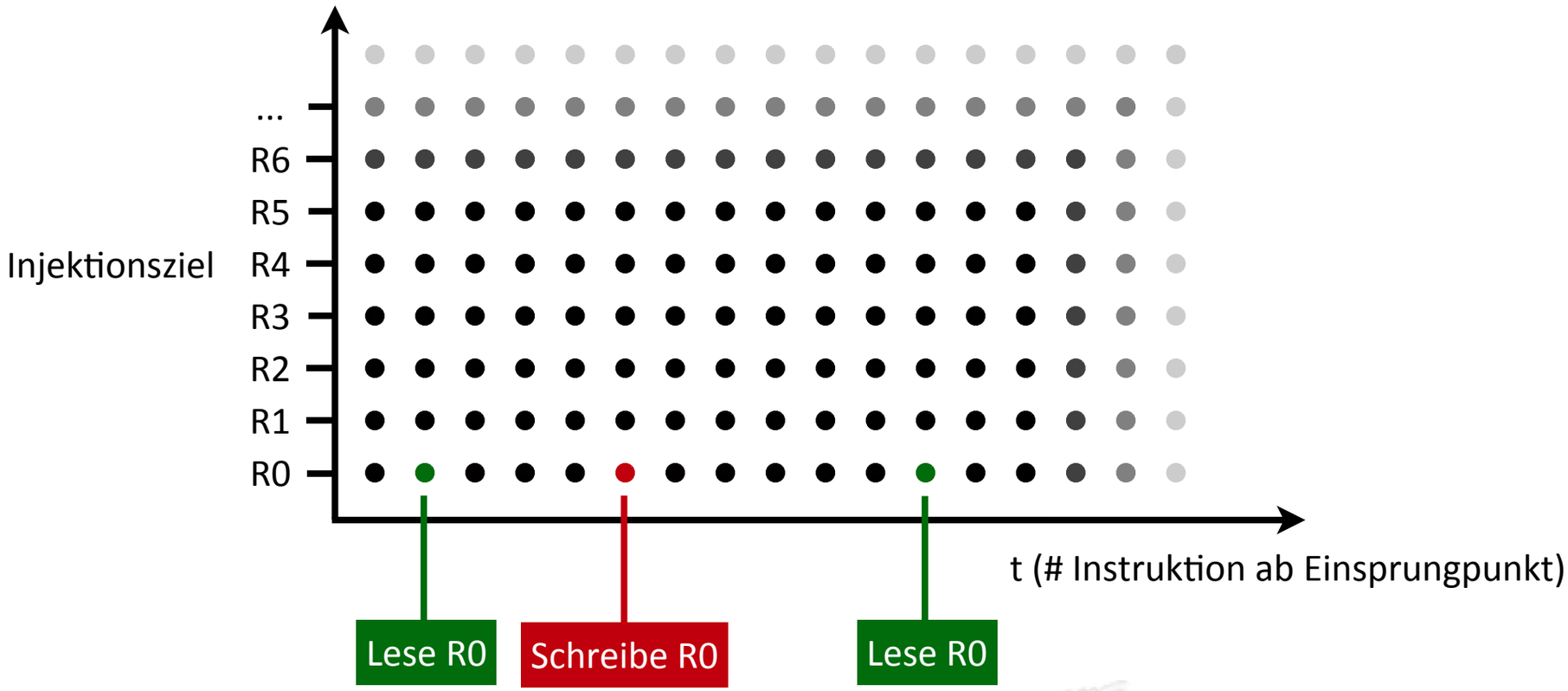
Theorie

Praxis

FAIL*

Praktikable Kampagnendauer I

- Fehlerraum noch immer riesig
- Einschränkung durch “Fault-Space Pruning”
  - Tilgung von unwirksamen und idempotenten Injektionen



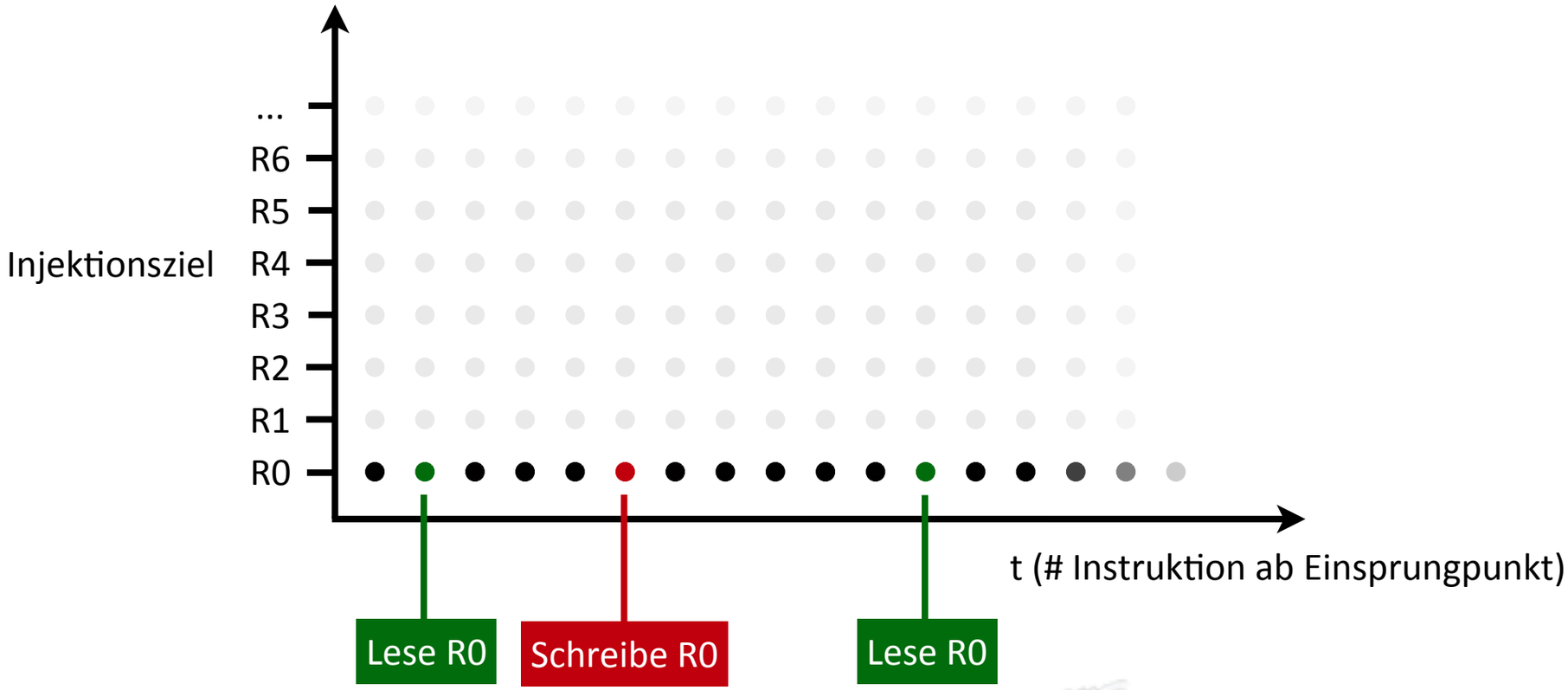
Theorie

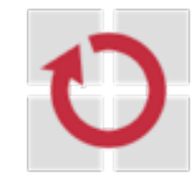
Praxis

FAIL*

Praktikable Kampagnendauer I

- Fehlerraum noch immer riesig
- Einschränkung durch “Fault-Space Pruning”
  - Tilgung von unwirksamen und idempotenten Injektionen





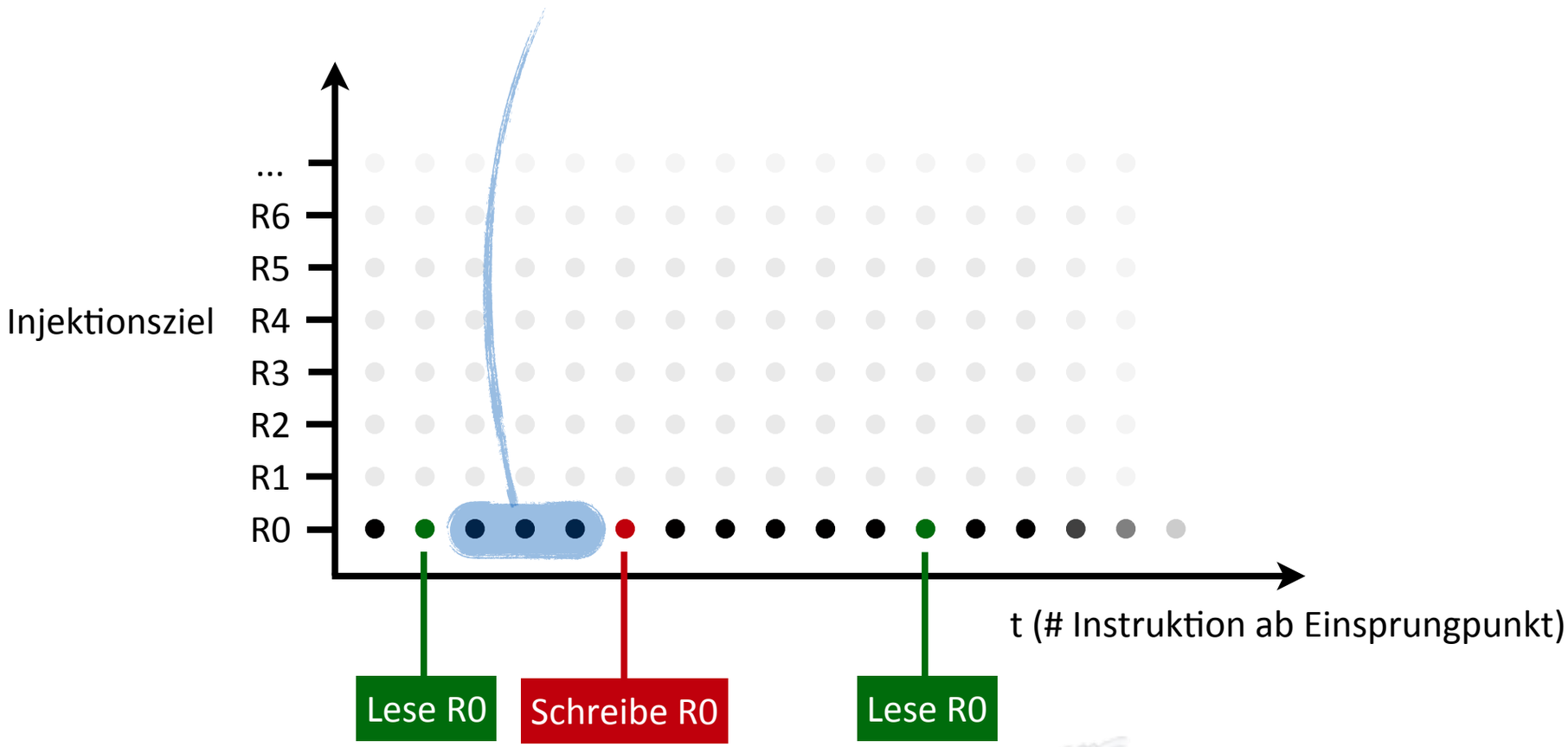
Theorie

Praxis

FAIL*

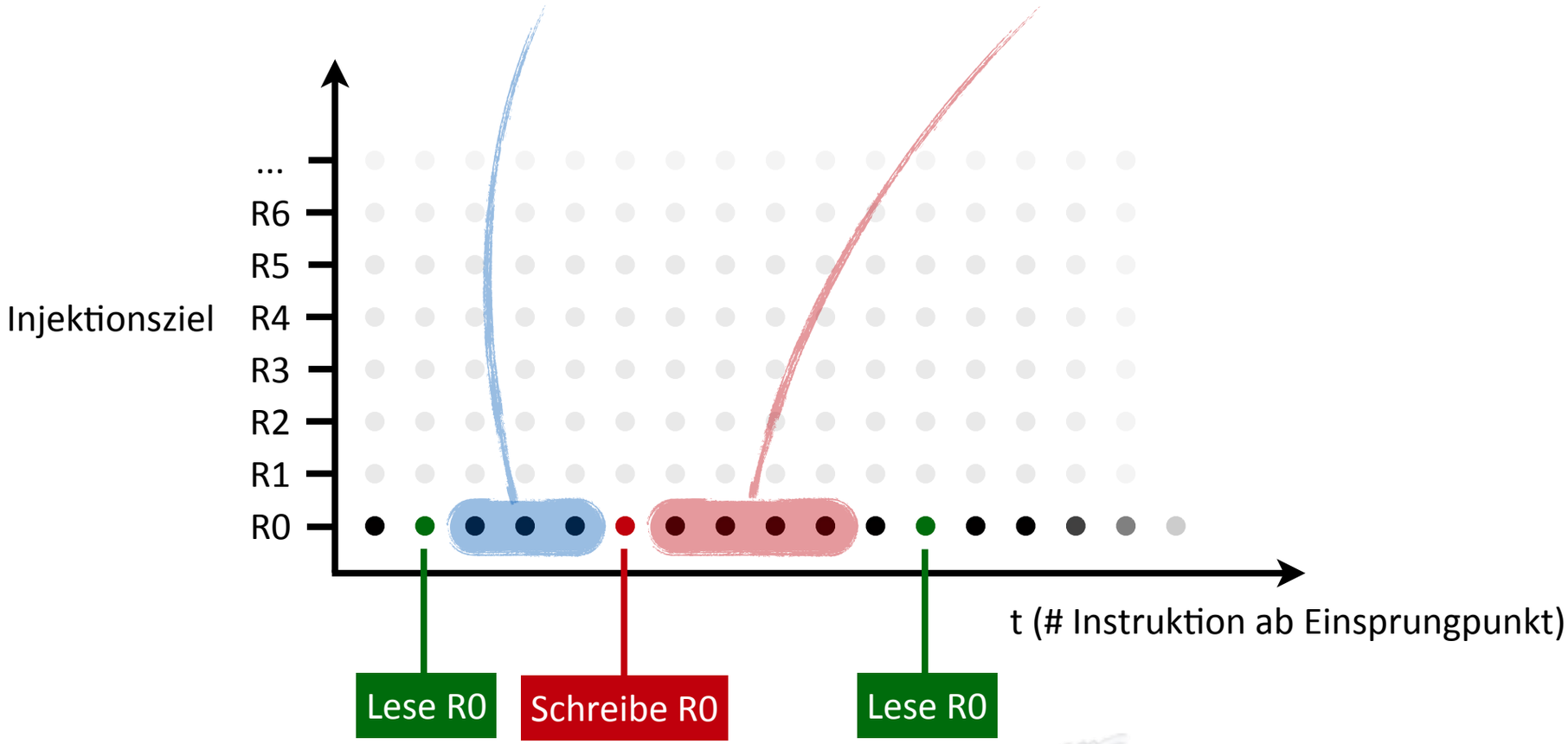
Praktikable Kampagnendauer I

- Fehlerraum noch immer riesig
- Einschränkung durch “Fault-Space Pruning”
  - Tilgung von unwirksamen und idempotenten Injektionen



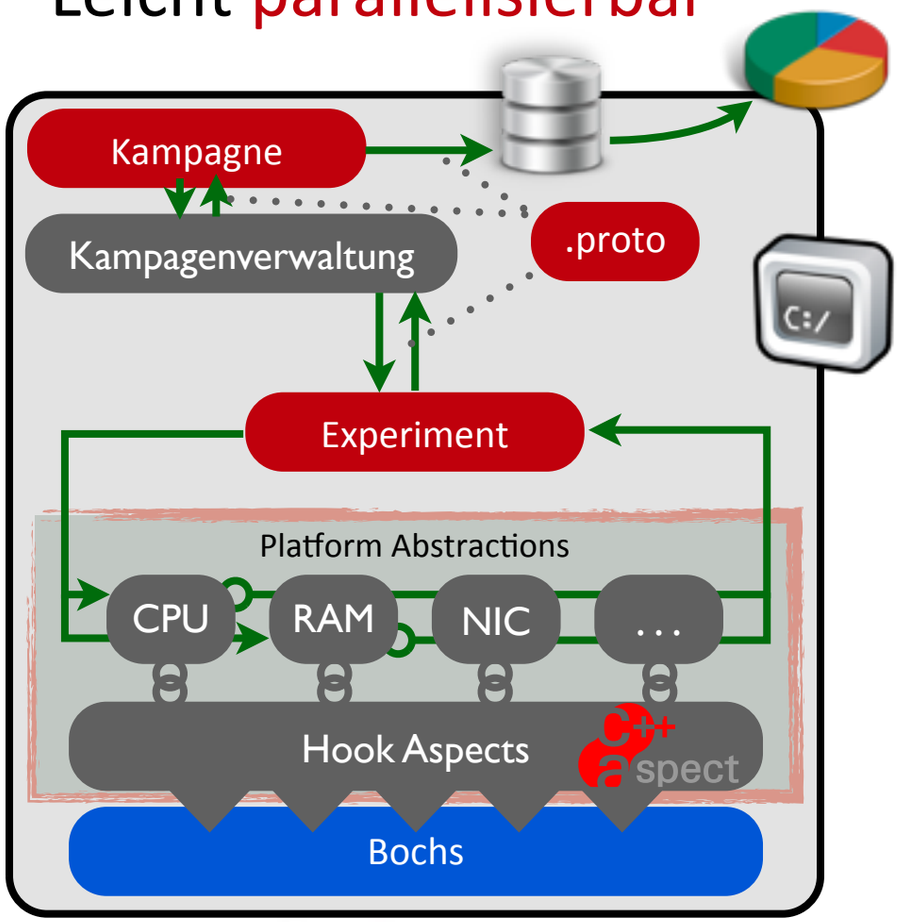
Praktikable Kampagnendauer I

- Fehlerraum noch immer riesig
- Einschränkung durch “Fault-Space Pruning”
  - Tilgung von unwirksamen und idempotenten Injektionen



Praktikable Kampagnendauer II

- Einzelexperimente sind voneinander unabhängig
  - Leicht parallelisierbar



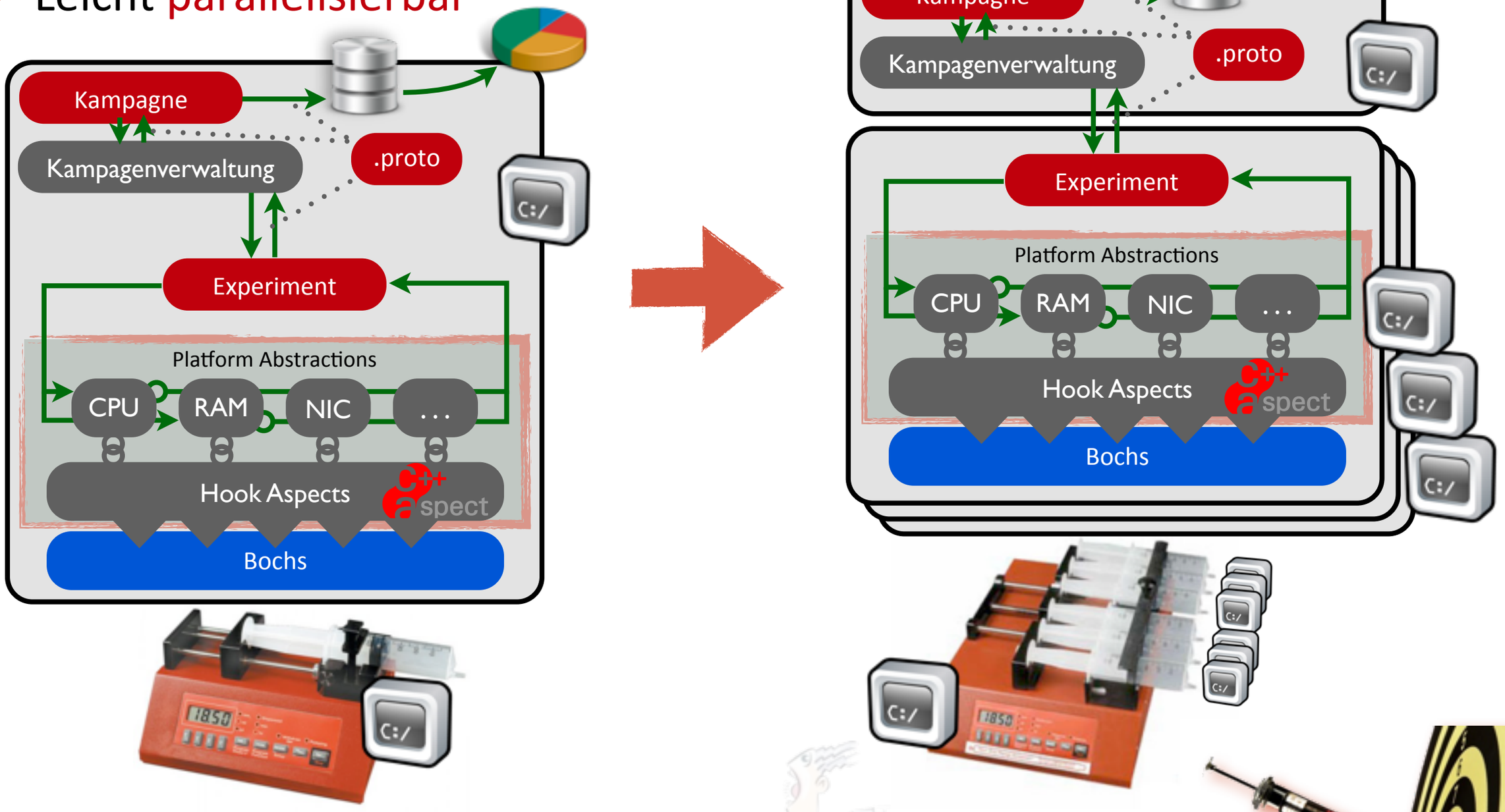
Theorie

Praxis

FAIL*

Praktikable Kampagnendauer II

- Einzelexperimente sind voneinander unabhängig
  - Leicht parallelisierbar



Theorie Praxis FAIL*