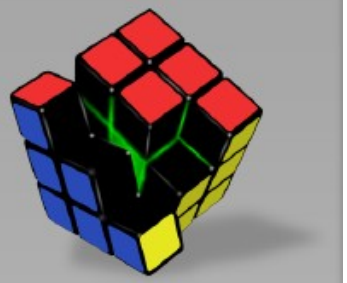


technische universität
dortmund

Ebenenübergreifende Bereitstellung von Daten und Lernergebnissen aus der Systemsoftware

Abschlussvortrag zur Masterarbeit

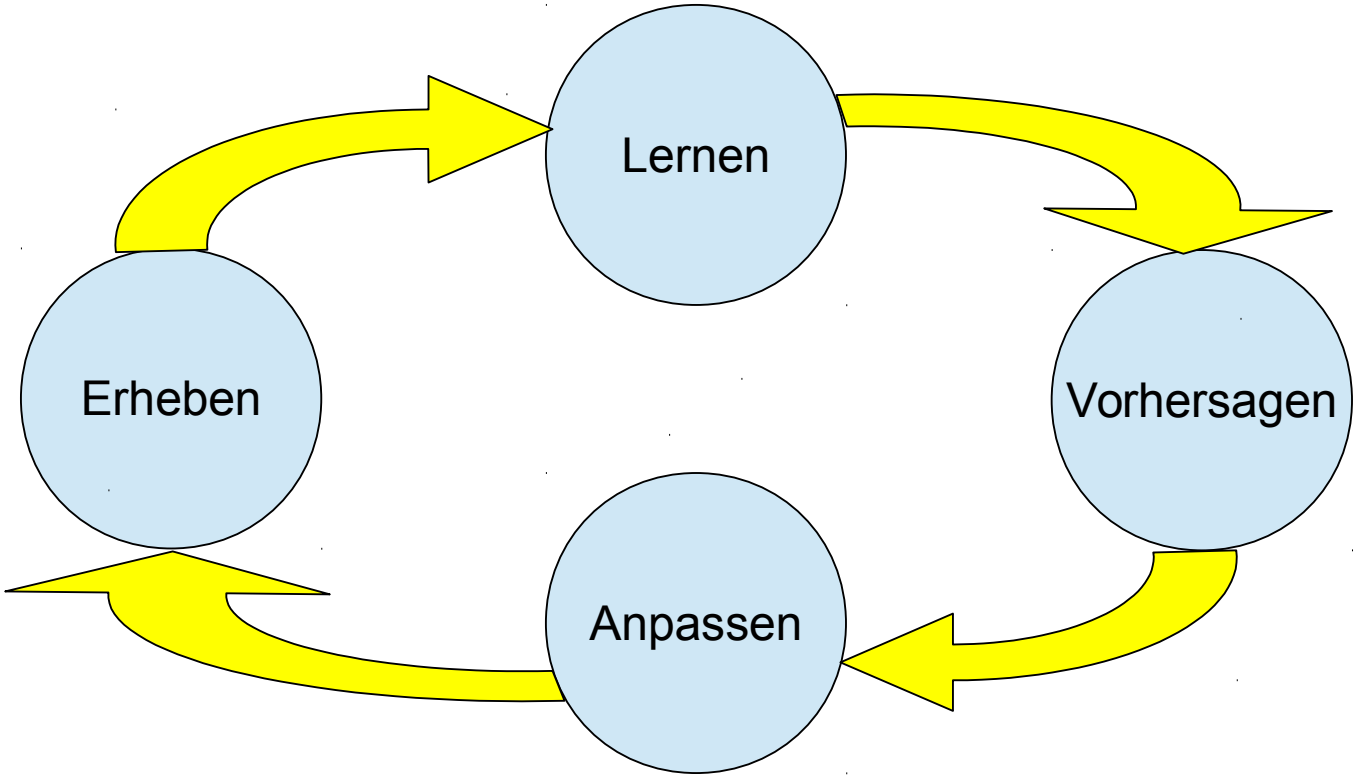
Alexander Lochmann

alexander.lochmann@tu-dortmund.de

AG Eingebettete Systemsoftware
Informatik 12, TU Dortmund

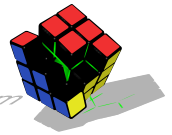
Motivation durch Teilprojekt A1

- Vision: ressourcenschonendes, autonomes Lernen und Anpassen



- Grundvoraussetzung: Werkzeug zur Erhebung von Daten

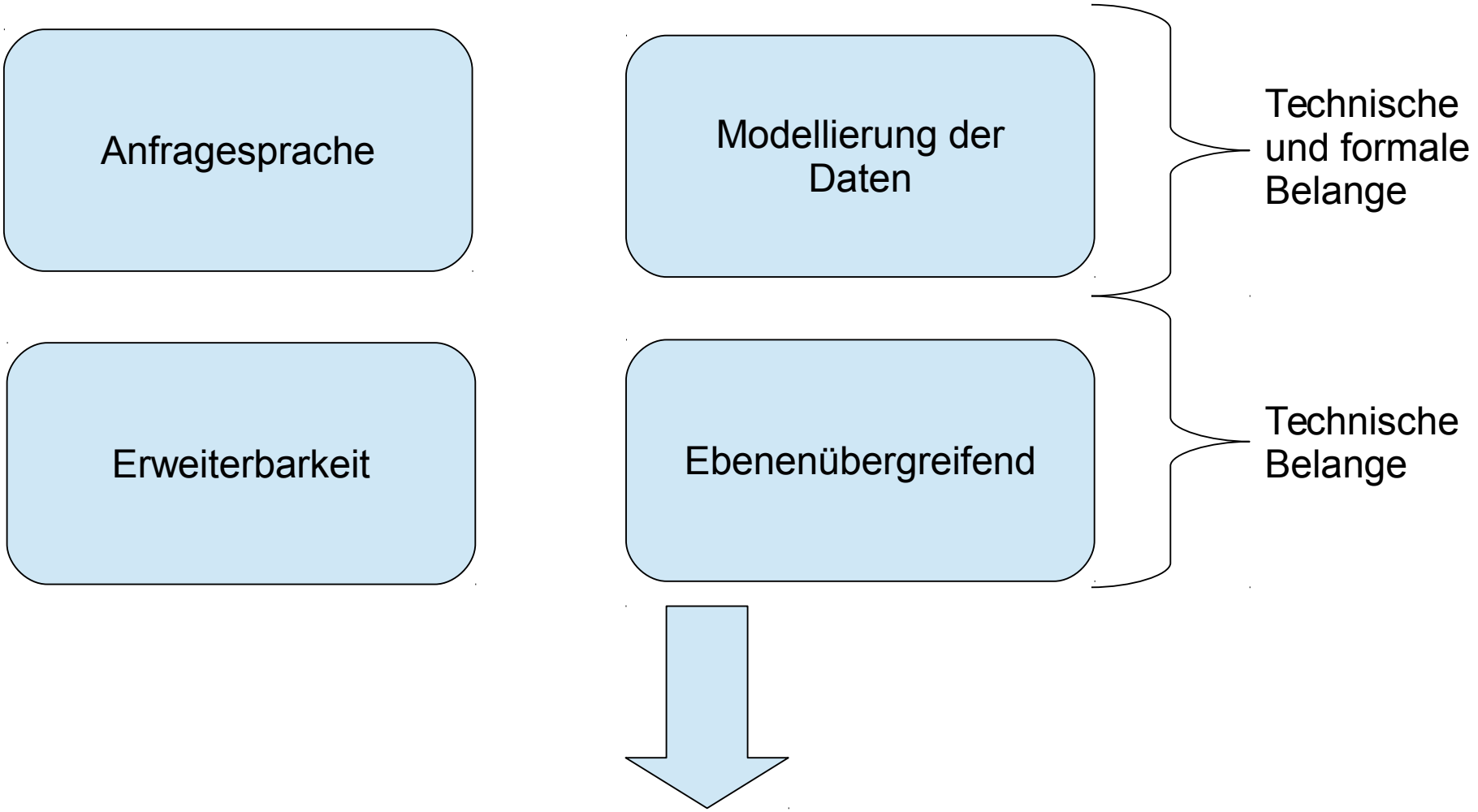
  → Bisherige Werkzeuge ungeeignet



Übersicht

- Anforderungen
- Datenmodell
- Anfragesprache
- Architektur von „*Claosi*“
- Evaluation
- Zusammenfassung

Anforderungen



Cross-layer availability of operating system information = Claosi

Datenmodell

- Basiert auf Metamodell von J. Streicher [1]
- Anreicherung mit weiteren Informationen, z.B. Art der Quelle
- Definition komplexer Datentypen
- Abstraktion von unnötigen Informationen

- Exemplarische Instanz für das Fallbeispiel „Paketverzögerung“

Datenmodell

```
model {
   namespace net {
      object device[str] {
         source txBytes : int;
         source rxBytes : int;
         event onRX() : packetType;
         event onTX() : packetType;
         event onEnqueue() : packetType;
      }
      type packetType {
         byte macProtocol;
         byte[] macHdr;
         byte networkProtocol;
         byte[] networkHdr;
         byte transportProcotol;
         byte[] transportHdr;
         int dataLength;
         process.process.sockets socket;
      }
```

Datenmodell

```
model {
   namespace net {
      object device[str] {
         source txBytes : int;
         source rxBytes : int;
         event onRX() : packetType;
         event onTX() : packetType;
         event onEnqueue() : packetType;
      }
      type packetType {
         byte macProtocol;
         byte[] macHdr;
         byte networkProtocol;
         byte[] networkHdr;
         byte transportProcotol;
         byte[] transportHdr;
         int dataLength;
         process.process.sockets socket;
      }
```

201411031615,“net.device“ → „eth0“

201411031615,“net.device“ → „eth0“,
“net.device.txBytes“ → 42

201411031615,“net.device“ → „eth0“,
“net.device.packetType“→ {1,....,63,4711}

Datenmodell

```
   object socket[int] {
      source int type;
      source int flags;
   }
   source delayTolerance : int;
namespace process {
   object process[int] {
     source utime : int;
     source stime : int;
     source sockets : net.socket;
   }
}
namespace ui {
   source foregroundApp : ui.app;
   object app[string] {
      source process : process.process;
   }
} }
```

201411031615,“net.socket“ → 4711

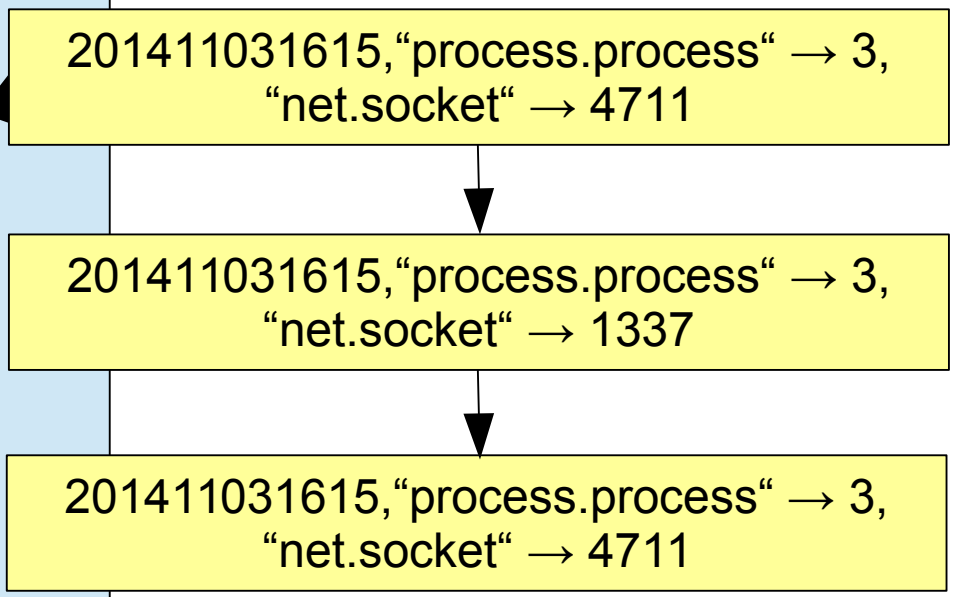
Datenmodell

```
object socket[int] {
    source int type;
    source int flags;
}
source delayTolerance : int;
namespace process {
  object process[int] {
    source utime : int;
    source stime : int;
    source sockets : net.socket;
  }
}
namespace ui {
  source foregroundApp : ui.app;
  object app[string] {
    source process : process.process;
  }
} }
```

201411031615,“net.delayTolerance“ → 21

Datenmodell

```
object socket[int] {
    source int type;
    source int flags;
}
source delayTolerance : int;
namespace process {
  object process[int] {
    source utime : int;
    source stime : int;
    source sockets : net.socket;
  }
}
namespace ui {
  source foregroundApp : ui.app;
  object app[string] {
    source process : process.process;
  }
}}
```



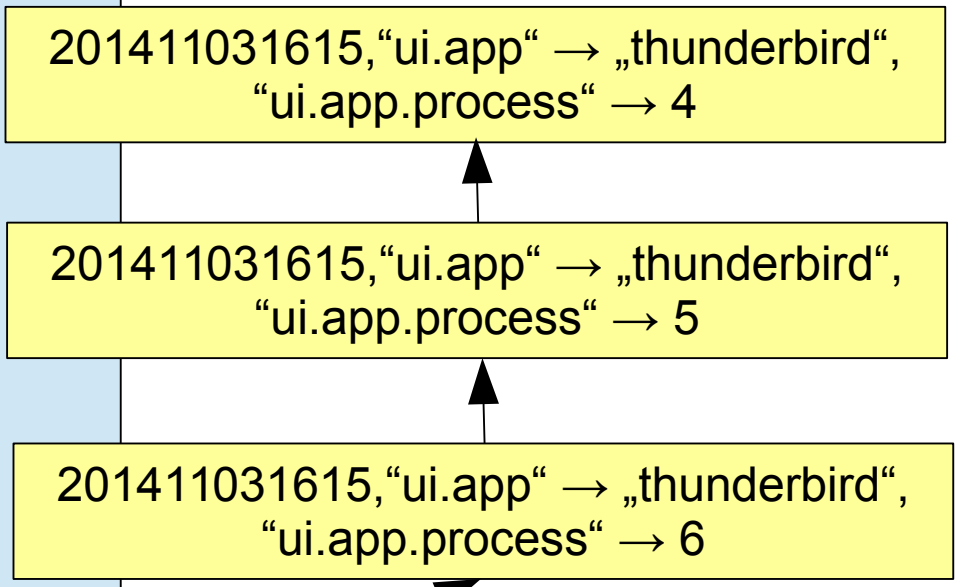
Datenmodell

```
    object socket[int] {
      source int type;
      source int flags;
    }
    source delayTolerance : int;
  namespace process {
    object process[int] {
      source utime : int;
      source stime : int;
      source sockets : net.socket;
    }
  }
  namespace ui {
    source foregroundApp : ui.app;
    object app[string] {
      source process : process.process;
    }
  } }
```

201411031615,
“ui.foregroundApp“ → „thunderbird“

Datenmodell

```
    object socket[int] {
       source int type;
       source int flags;
    }
    source delayTolerance : int;
  namespace process {
    object process[int] {
      source utime : int;
     source stime : int;
     source sockets : net.socket;
    }
  }
  namespace ui {
    source foregroundApp : ui.app;
    object app[string] {
      source process : process.process;
    }
  } }
```



Anfragesprache

- Als Vorbild dient Datenstromsystem *Aurora* [2]
- Vollständige Software zur Verarbeitung von Datenströmen
- Benutzt externe Datenquellen
- Bietet Definition von Operatoren und Tupeln

- Basierend auf *Aurora* wurde neuer Dialekt entworfen
- Anfrage umfasst nun Quelle bis „Ziel“
- Operatoren übernommen, modifiziert oder entworfen

Anfragesprache - Operatoren

GenStream(element, urgent) = stream
    Source(...,period)
    Event(...)
    Object(...,OBJECT_CREATE)

Join(stream,persistent element,predicates) = stream

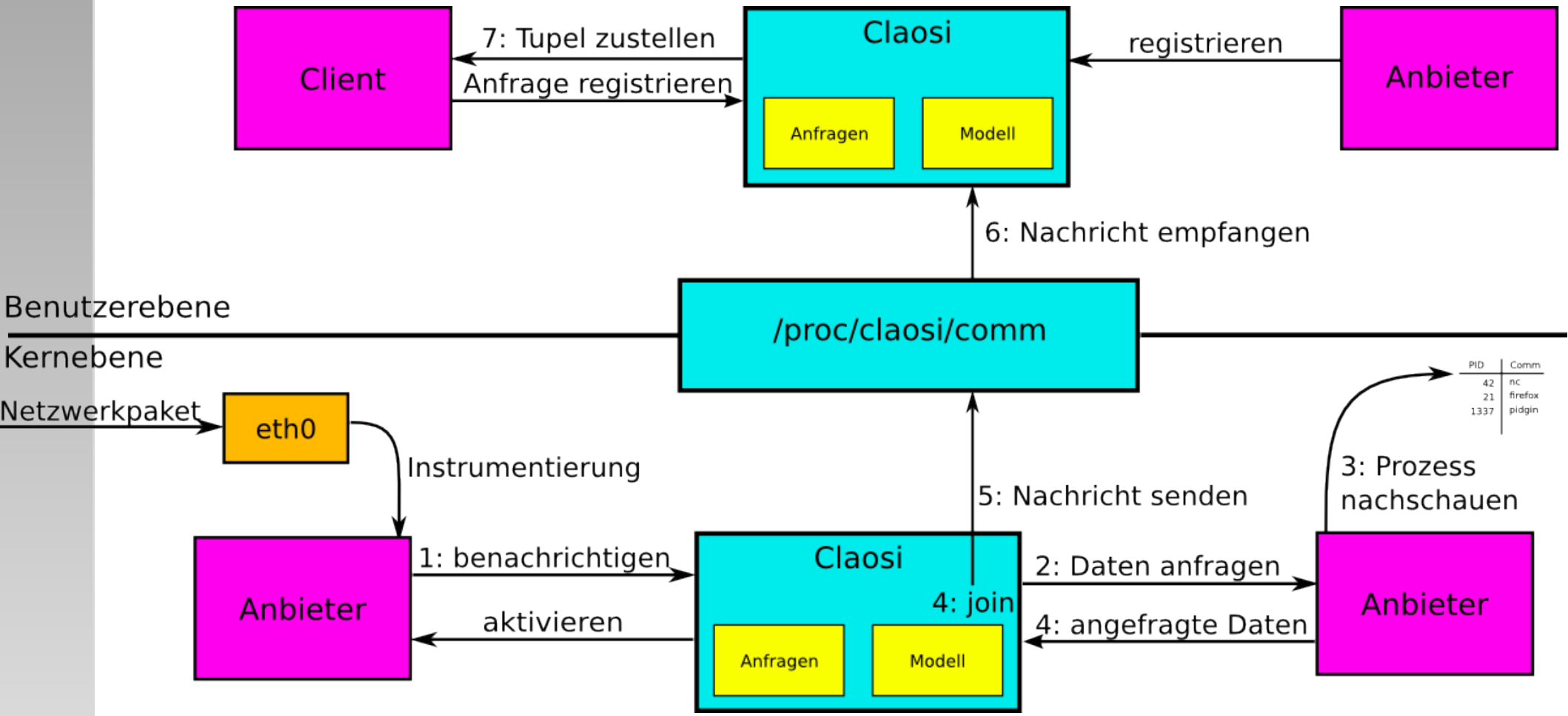
Select(stream, $datum_i$, $datum_k$, $datum_z$,...) = stream

Filter(stream, predicates) = stream

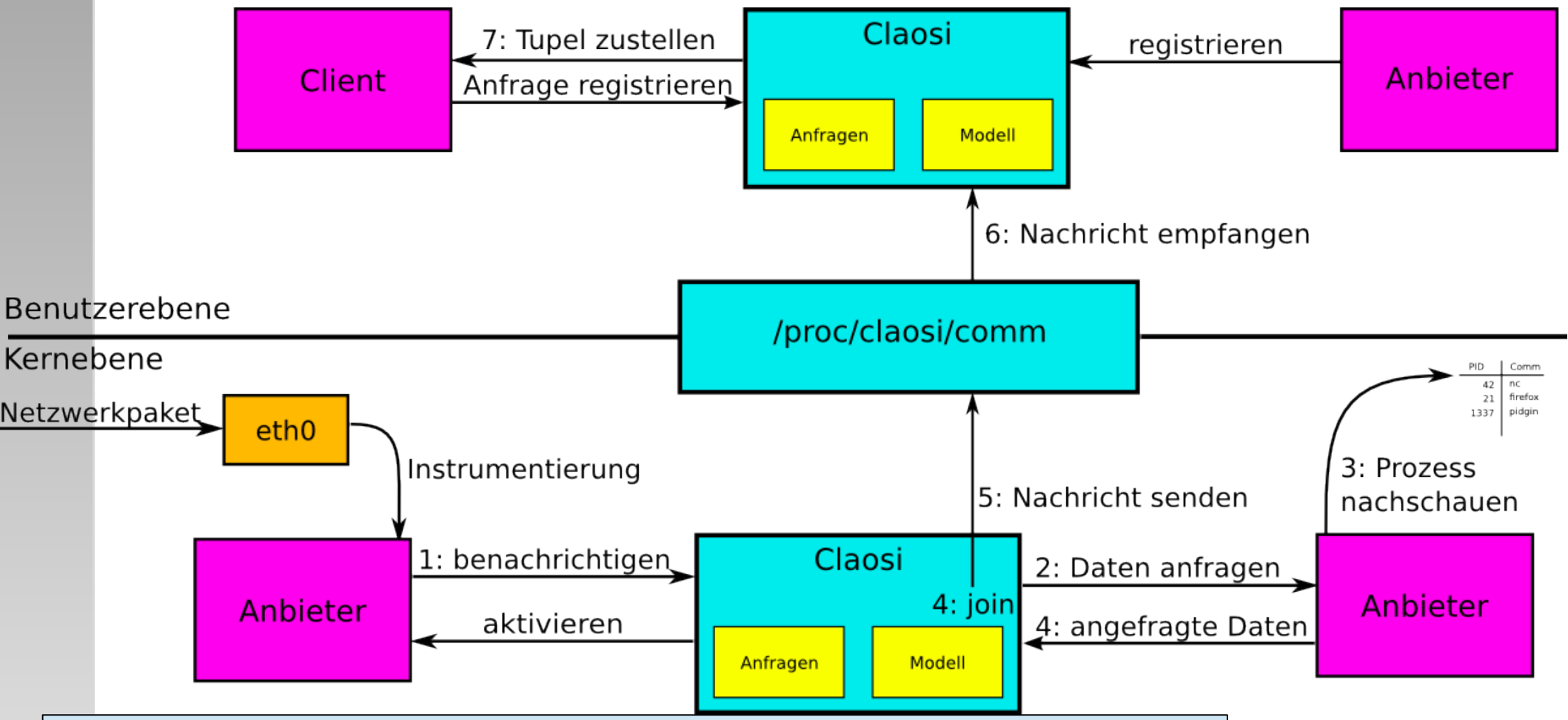
Sort(stream, window size,elements) = stream

Aggregate(stream, function, window size, elements) = stream

Claosi


Client
7: Tupel zustellen
Anfrage registrieren
Claosi
Anfragen
Modell
registrieren
Anbieter
6: Nachricht empfangen
Benutzerebene
Kernebene
/proc/claosi/comm
Netzwerkpaket
eth0
Instrumentierung
5: Nachricht senden
PID
Comm
42
nc
21
firefox
1337
pidgin
3: Prozess nachschauen
Anbieter
1: benachrichtigen
aktivieren
Claosi
4: join
Anfragen
Modell
2: Daten anfragen
4: angefragte Daten
Anbieter

Claosi




```
v = Event(net.device[eth1].onTx,0)
    Join(v,process.process[*].sockets,
         Stream.net.packetType.socket == Join.process.process.sockets)
```

Evaluation

- Bewertung des Werkzeugs hinsichtlich Einfluss auf das gesamte System
- Fragestellung: Wie viel kostet der generische Ansatz?

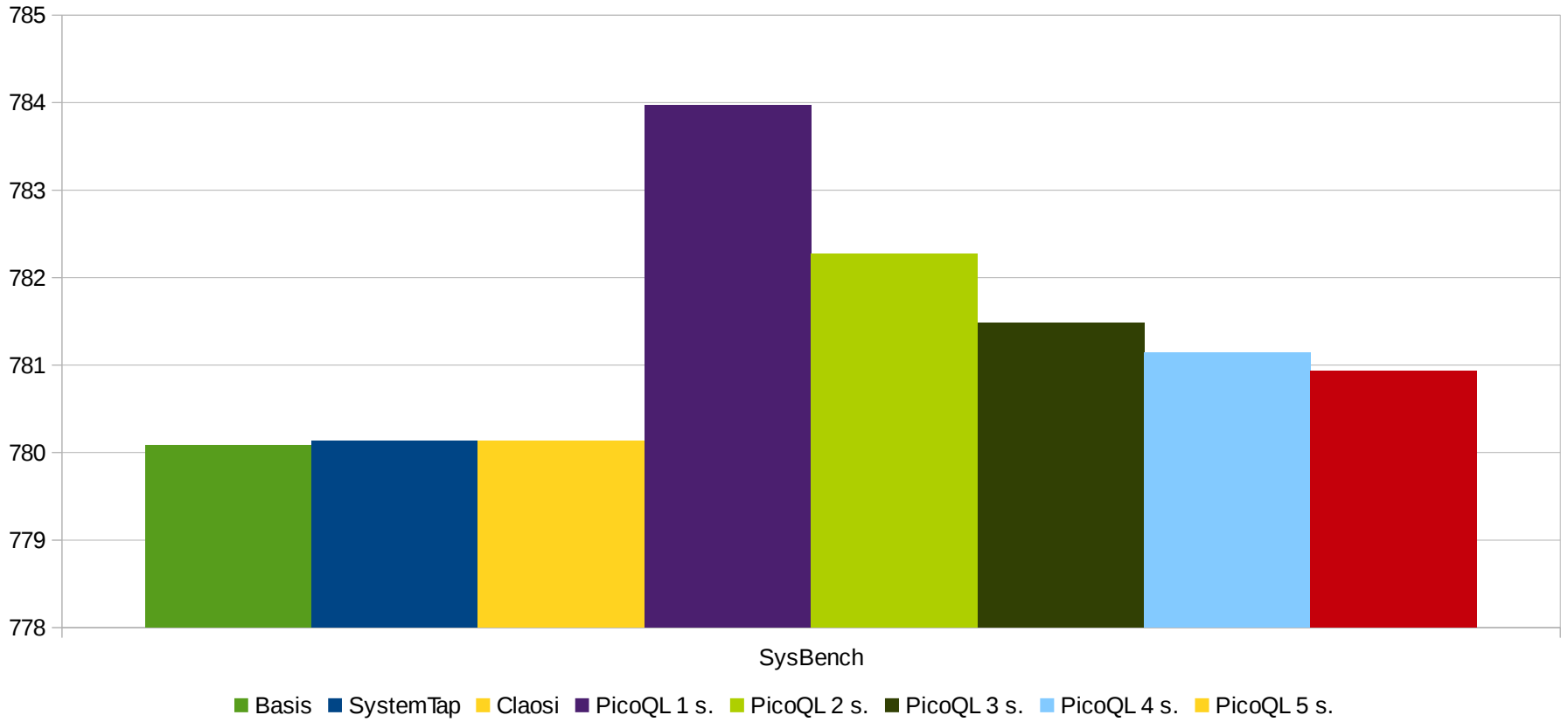
- Messung der Laufzeiten von
  - *SysBench* [3]: Berechnung von Primzahlen
  - Übersetzen des Linux-Kerns
- Arbeitsplatzrechner mit Intel Core i5 bei 3,4GHz
- Vergleich mit *SystemTap* [5] und *PicoQL*[6]

Evaluation

- Bewertung von drei Szenarien
  - **A**: zu jedem erzeugten Prozess die Systemzeit und den Namen bestimmen
  - **B**: zu jedem Netzwerkpaket den Prozess bestimmen
  - **C**: Netzwerkpakete nach der Größe filtern
- Bei B & C Übertragung von Daten zu Zweitrechner per Gbit-LAN
- Anfragen nur im Kern verarbeitet

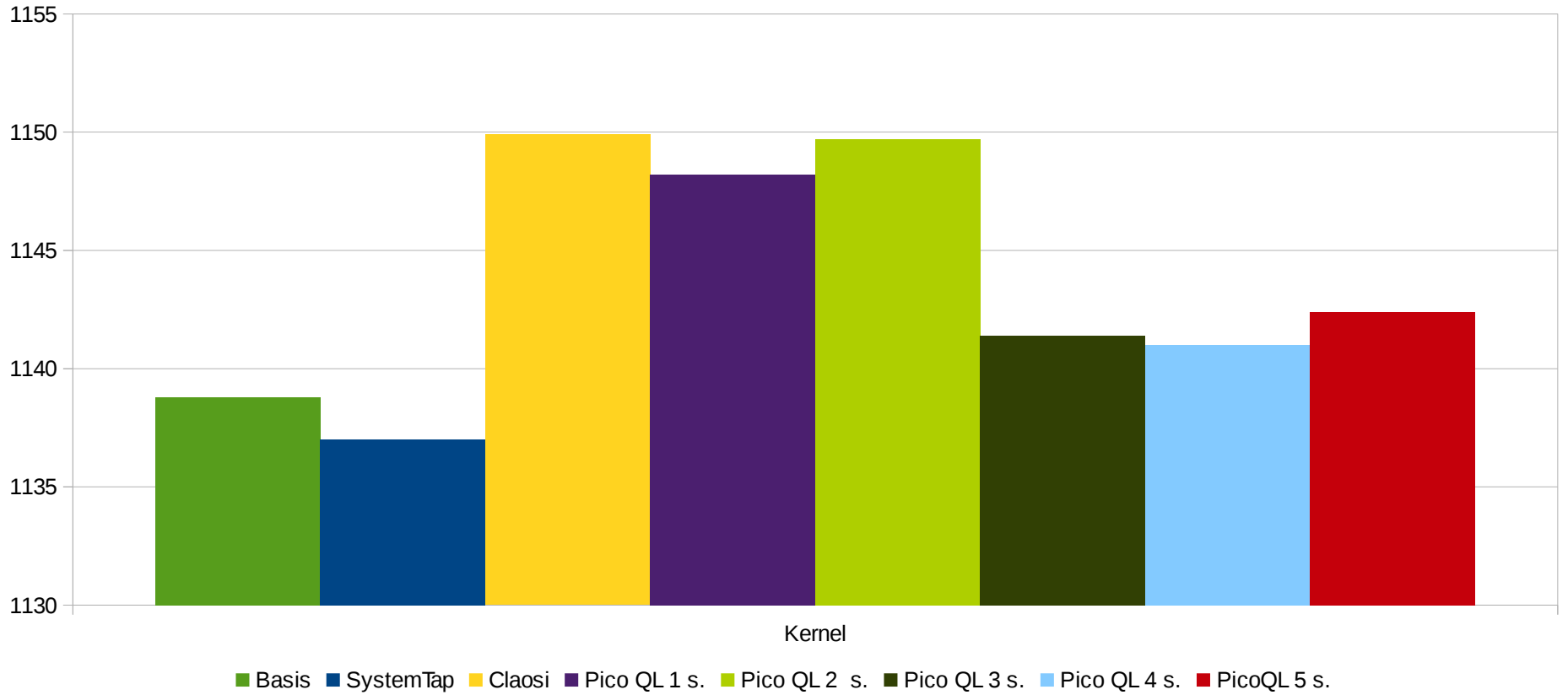
Evaluation – Szenario A SysBench

```
x = Object("process.process",OBJECT_CREATE,0)
x = Join(x,"process.process.comm",Stream.process.process==Join.process.process)
x = Join(x,"process.process.stime",Stream.process.process==Join.process.process)
```



Evaluation – Szenario A Kernel

```
x = Object("process.process",OBJECT_CREATE,0)
x = Join(x,"process.process.comm",Stream.process.process==Join.process.process)
x = Join(x,"process.process.stime",Stream.process.process==Join.process.process)
```



Evaluation - Szenario B

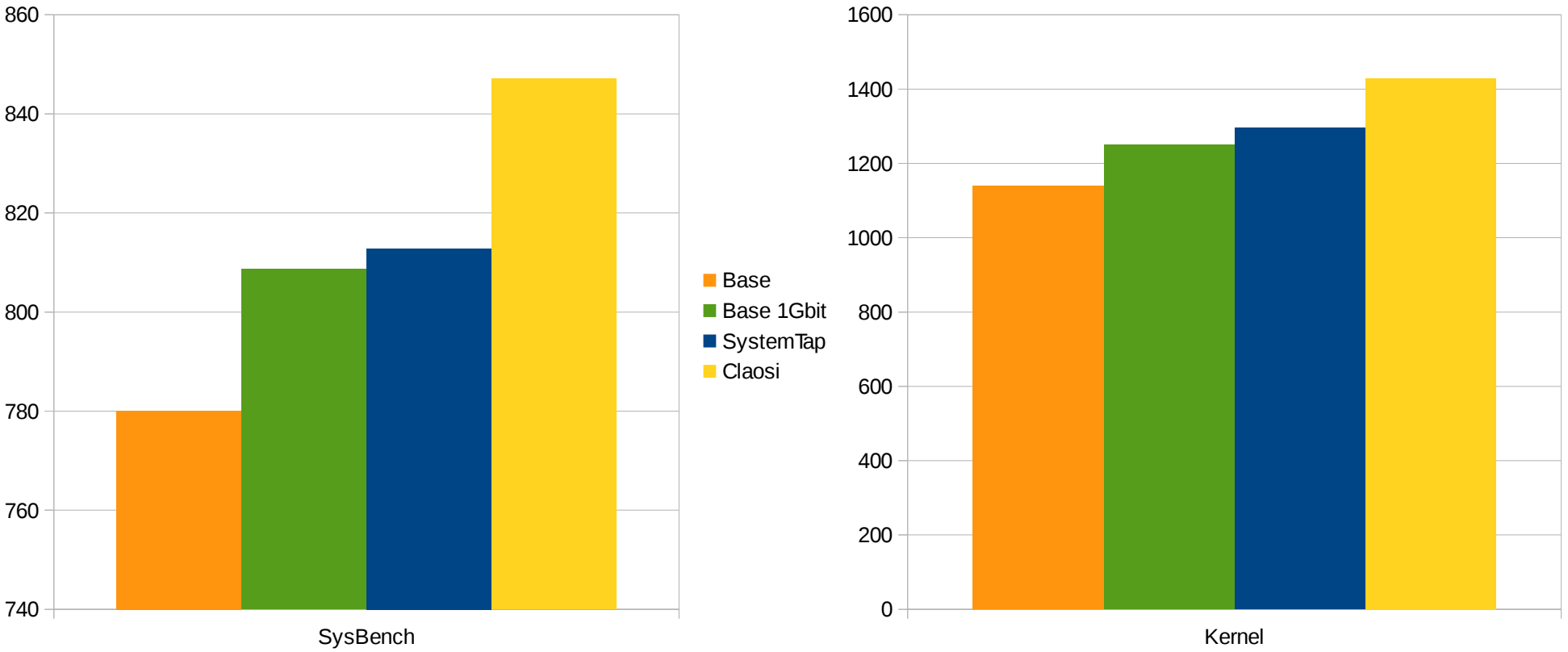
```
v = Event(net.device[p4p1].onTx,0)
      Join(v,process.process[*].sockets,
            Stream.net.packetType.socket == Join.process.process.sockets)

w = Event(net.device[p4p1].onRx,0)
       Join(w,process.process[*].sockets,
            Stream.net.packetType.socket == Join.process.process.sockets)
```

- Hohe Datenrate → viele Jobs → keine rechtzeitige Abarbeitung → Arbeitsspeicherverbrauch steigt
- Erst bei 10 Mbit/s stellte sich Gleichgewicht ein
- Keine Messwerte, da keine adäquaten Vergleichswerte vorhanden

Evaluation – Szenario C

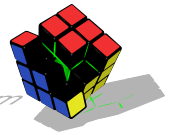
```
v = Event(net.device[p4p1].on{Tx,Rx},0)
v = Filter(v,net.packetType.dataLength >= {1000,100})
    Select(v,net.packetType)
```



Ausblick

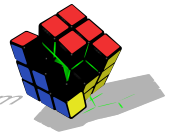
- Optimierung des ...
  - Join-Operators
  - Auflösen von Pfaden im Datenmodell, z.B. mittels Hashing
- Betrachtung von anderen Ansätzen wie ...
  - *Continous Query Language* [4]
  - Weiteren Implementierungen der *Lambda*-Architektur
- Konzept der Anfragesprache überdenken

  → Genügt der ein reiner Datenstromansatz?



Zusammenfassung

- *Claosi* als Werkzeug zur ebenenübergreifenden Bereitstellung von Daten
- Dynamisch durch Anbieter erweiterbar
- Lernergebnisse stehen als gewöhnliche Anbieter bereit
- Anfragesprache zur Erzeugung und Verarbeitung von Daten
- Abstraktion von konkreter Implementierung durch Datenmodell



Bibliographie

[1] Streicher, Jochen: *Data Modeling of Ubiquitous System Software*; Version: 07.2014; http://sfb876.tu-dortmund.de/PublicPublicationFiles/streicher_2014a.pdf

[2] Abadi et. al.: *Aurora: A New Model and Architecture for Data Stream Management*. In: The VLDB Journal 12 (2003), August, Nr. 2, 120–139.

[3] *SysBench: a system performance benchmark*. https://launchpad.net/sysbench.

[4] Arasu, A.; Babu, S. & Widom, J.: *The CQL Continuous Query Language: Semantic Foundations and Query Execution*. VLDB Journal, Very Large Data Bases Endowment Inc., 2006

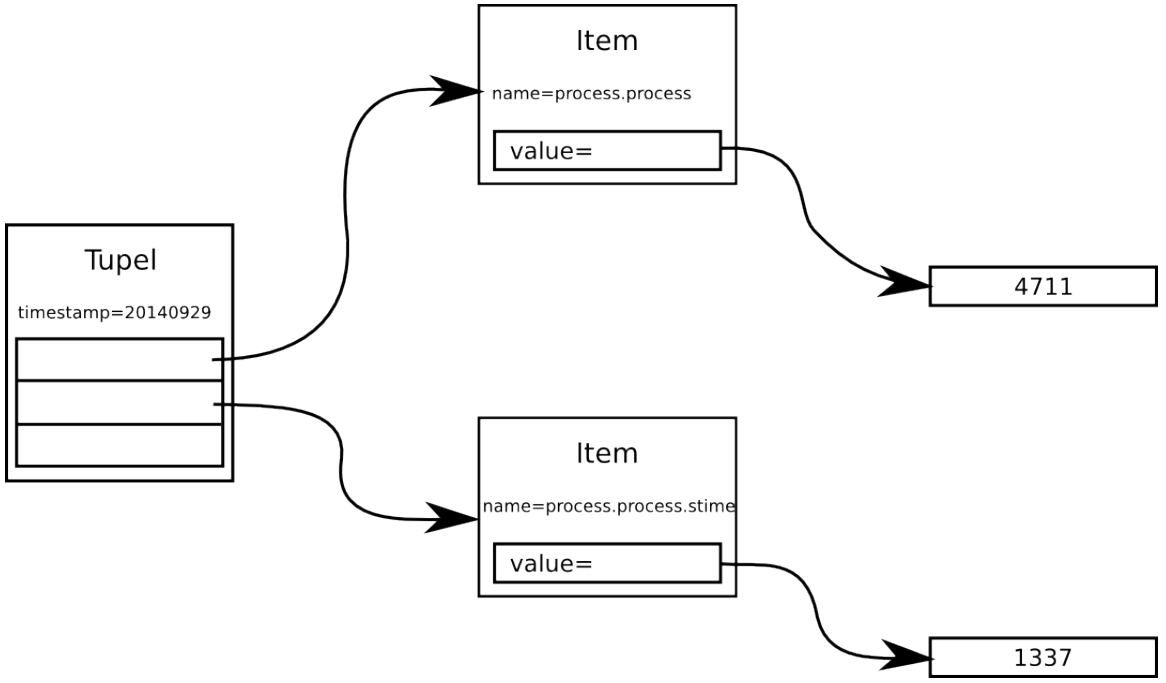
[5] Red Hat, Inc.: *SystemTap*. https://sourceware.org/systemtap/ - Version: 09.2014

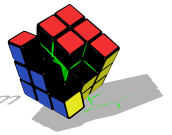
[6] Fragkoulis et. al.: *Relational Access to Unix Kernel Data Structures*. In: Proceedings of the Ninth European Conference on Computer Systems (2014)

Anfragesprache - Ausgabeformat

(Zeitstempel *us*, { $Datum_1$, $Datum_2$, …, $Datum_n$})

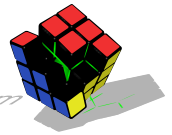
$Datum_i$ = Schlüssel → Wert





Motivation

- Ziel von SFB8761 Teilprojekt A1 ist die „Verbesserung“ von ubiquitären Systemen
- Verbesserung heißt u.a. Energieverbrauch senken oder Antwortzeiten reduzieren
- Verbesserung hinsichtlich beliebiger Kriterien
- Zur Laufzeit Anpassung der Systemsoftware
- Justage erfolgt gemäß Vorhersage durch Lernverfahren



Datenmodell - Implementierung

- Implementierung als Baumstruktur
- Traversierung erforderlich und umgesetzt durch:
  - Jeder Knoten besitzt Zeiger auf Kinder
  - Verweis auf Vaterknoten
- Verwaltung von Funktionszeigern zur (De-)Aktivierung und Abfrage der Anbieter
- Vermerken der registrierten Anfragen pro Knoten