Concept-based pubsub and beyond

TECHNISCHE UNIVERSITÄT DARMSTADT

DVS

Publish/Subscribe is Becoming More and More Important

TECHNISCHE UNIVERSITÄT DARMSTADT

today's distributed systems require fast-paced data distribution

→ publish/subscribe model
(push-based interaction, implicit invocation)

- lightweight
- decoupled
- low-latency
- scalable

anonymity

2005: $1 billion

2010: $20 billion

DVS

The Problem is Heterogeneity

weight = 25

25 kg

25 lb

25 t

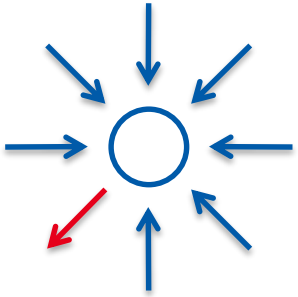
Why this is more than data integration

There are commercial products for data integration in databases

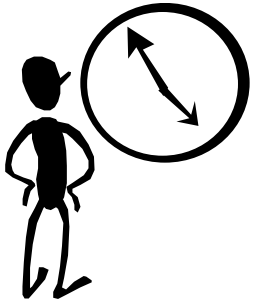
But: EBS have some key additional challenges

Source: www.ingenesist.com

anonymity

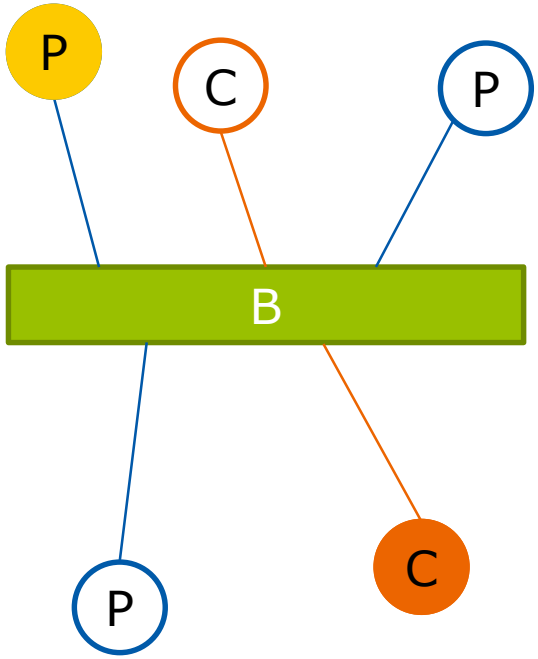


dynamics



latencies important

Transformations to the Rescue



Producer sends metadata to broker

Consumer sends metadata to broker

Broker matches/transforms

Event Model

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition: Type

A type T is either a primitive type or a complex type. Complex types are declared as T extends $T_1, \ldots, T_n[a_1{:}T_1', \ldots, a_n{:}T_n']$. $\overline{T}$ are (complex) super-types of T. The attributes of T include those of all ist super-types as well as $\overline{a}$.

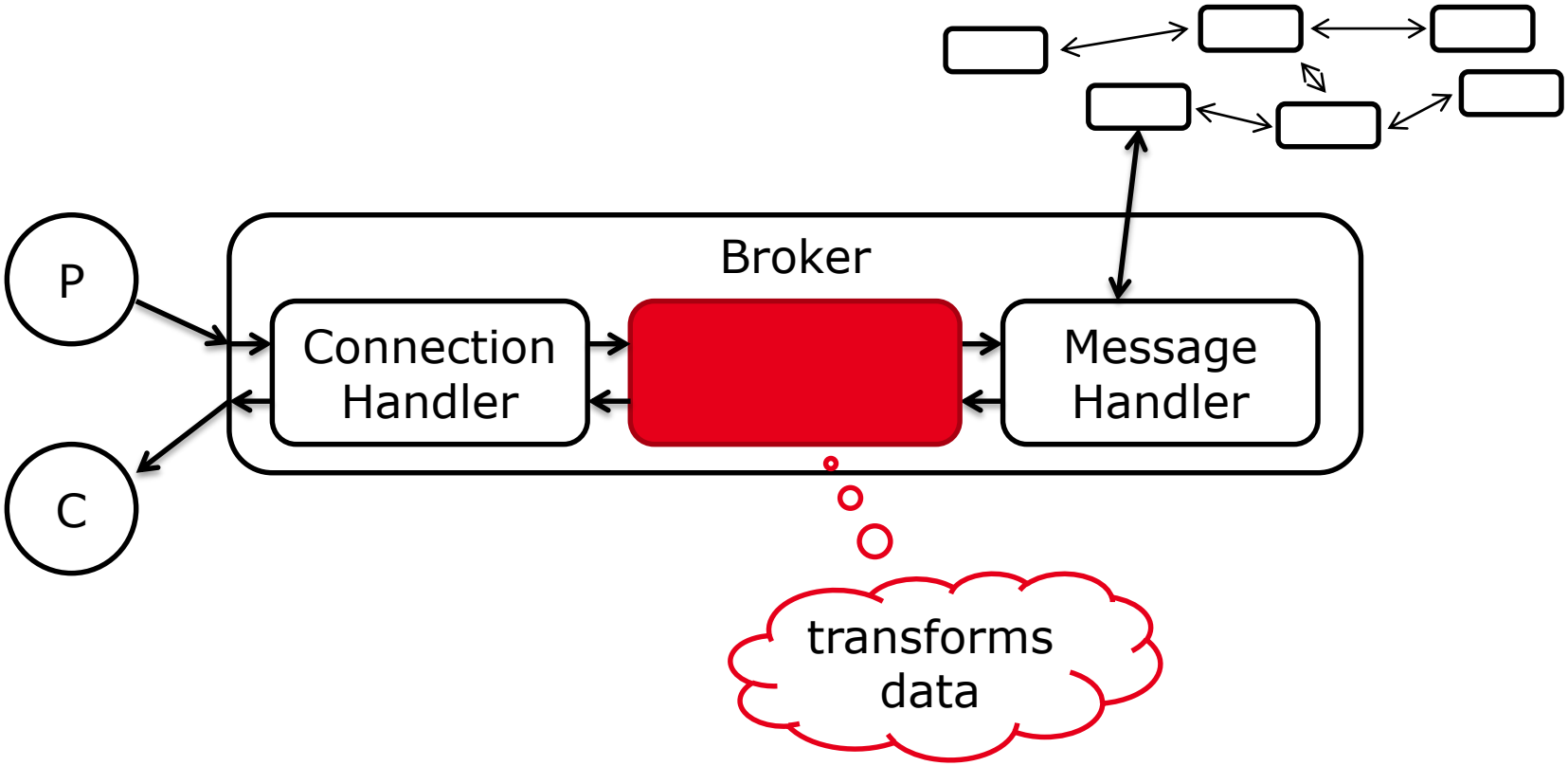
Example:

```
InvoiceLine
    Money amountToPay
        string currency
        float amount
    Money price
        string currency
        float amount
    ItemSpecs specs
        float height
        float width
        float depth
```

$$T = \text{InvoiceLine}$$
$$\overline{T} = \emptyset$$
$$\overline{a} = \{\text{amountToPay} : \text{Money}, \text{price} : \text{Money}, \text{specs} : \text{ItemSpecs}\}$$

DVS

High-level Architecture



Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

General form:

$\text{ItemSpecs} \rhd \text{toUSItemSpecification}$

Nested specification:

$\text{InvoiceLine.Money} \rhd \text{toDollars}$

Attribute specification:

$\text{InvoiceLine.amountToPay} \rhd \text{toDollars}$

Subtyping support:

$\text{InvoiceLine.spec(WeightedItemSpecs)} \rhd \cdots;$

;

DVS

Rules Have Intuitive Syntax and Priorities

TECHNISCHE UNIVERSITÄT DARMSTADT

pattern ⊳ function

3 Nesting level

ItemSpecification ⊳ toUSSpec

CustomsDeclaration.ItemSpecification ⊳ toIdentity

2 Attributes

InvoiceLine. Money ⊳ addTaxToMoney

InvoiceLine.amountToPay ⊳ toDollars

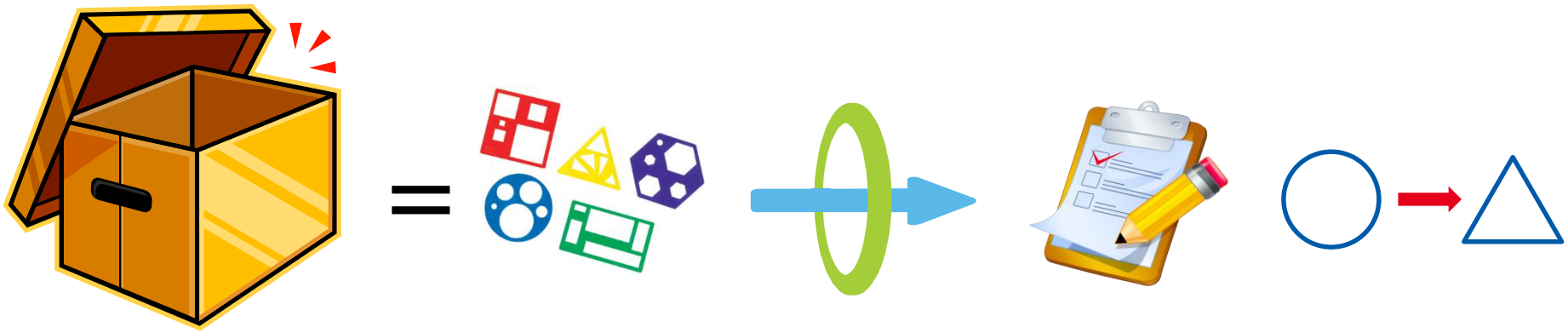
1 Subtyping

InvoiceLine.spec ⊳ ...

InvoiceLine.spec(WeightedItemSpec) ⊳ ...

DVS

Interpretation Environments as a Container



Interpretation Environment

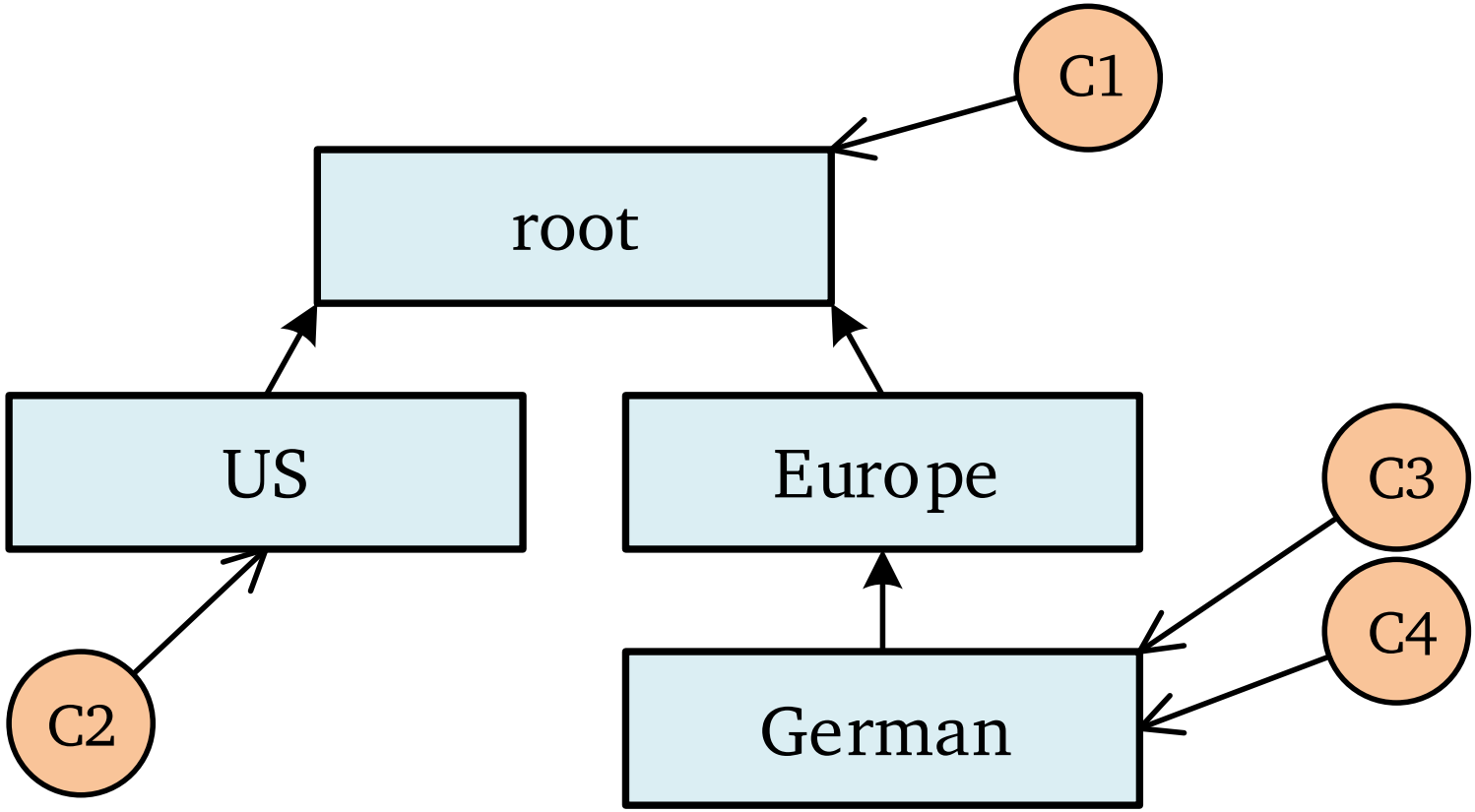
Types

Mappings

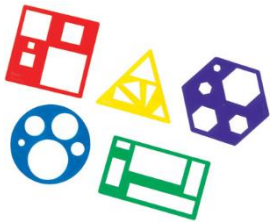
Rules

Conversion functions

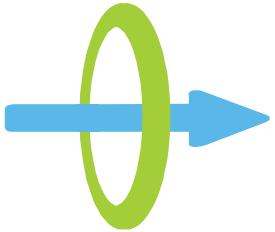
Interpretation Environments Form Hierarchies



Interpretation Environment Specialization

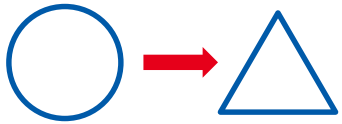


Local types
- straightforwardly inherited



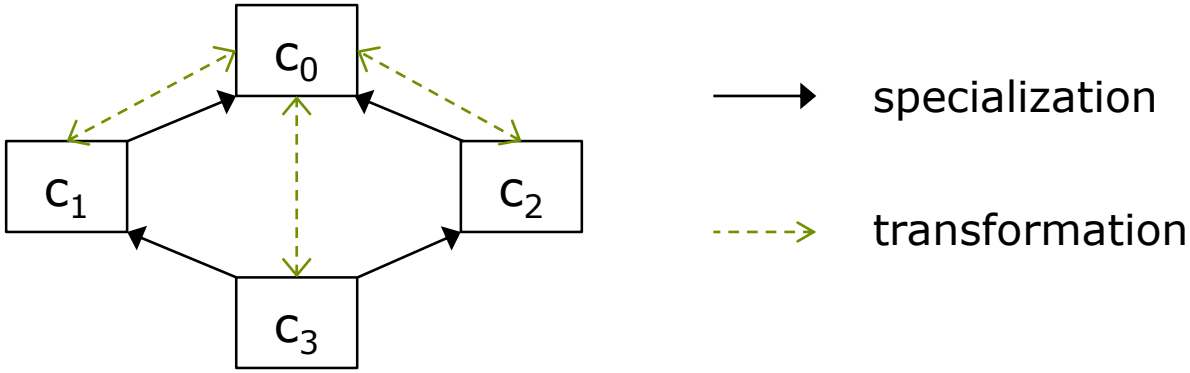
Mappings
- overriding: identical left side
- overloading: through subtypes

Rules
- overriding: identical patterns



Conversion functions
- overriding: possibly but unlikely

Transformations Are Always Direct



Example Interpretation Environment Definition

TECHNISCHE UNIVERSITÄT DARMSTADT

Types

- InvoiceLine
  - Money amountToPay
    - string currency
    - float amount
  - Money price
    - string currency
    - float amount
  - ItemSpecification spec
    - float height
    - float width
    - float depth

Transformation Rules

Mappings

Conversion Functions

meters-to-yards = value * 1.09
meters-to-feet = value * 3.28
meters-to-inch = value * 39.37
toIdentity = value
⋮

Types

- USInvoiceLine
  - Money amountToPay
    - string currency
    - float amount
  - Money price
    - string currency
    - float amount
  - USItemSpecification spec
    - float height
    - float width
    - float depth
    - float weight

Transformation Rules

r1 = ItemSpecification ⊳ toUSItemSpecification
r2 = Money ⊳ toDollars
r3 = InvoiceLine.price ⊳ toIdentity

Mappings

InvoiceLine ⇛ USInvoiceLine

Conversion Functions

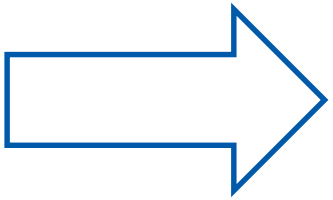
```
toUSItemSpecification = {
        // conversion
}

toDollars = {
        // conversion
}
```


DVS

Example Transformation

TECHNISCHE UNIVERSITÄT DARMSTADT

```
InvoiceLine
    Money amountToPay
        string currency  ["euro"]
        float amount     [100]
    Money price
        string currency  ["euro"]
        float amount     [100]
    ItemSpecs specs
        float height     [1]
        float width      [0.5]
        float depth      [0.3]
```



```
USInvoiceLine
    Money amountToPay
        string currency  ["usd"]
        float amount     [125]
    Money price
        string currency  ["euro"]
        float amount     [100]
    USItemSpecs specs
        float height     [39.37]
        float width      [19.69]
        float depth      [11.81]
        float weight     [172.5]
```


DVS

Implementation

Realization as a plugin
✓ no modification to code

Expressive

Extensible

Safe

Putting contexts to work

TECHNISCHE UNIVERSITÄT DARMSTADT

```
@Context("us")
@MapsTo("events.eventobjectstypes.root.Shipment")
public class USShipment {
        @Unit("lb")
        private double weight;
…
```

Annotations

```
…
Shipment.weight = lb2kg
…
```

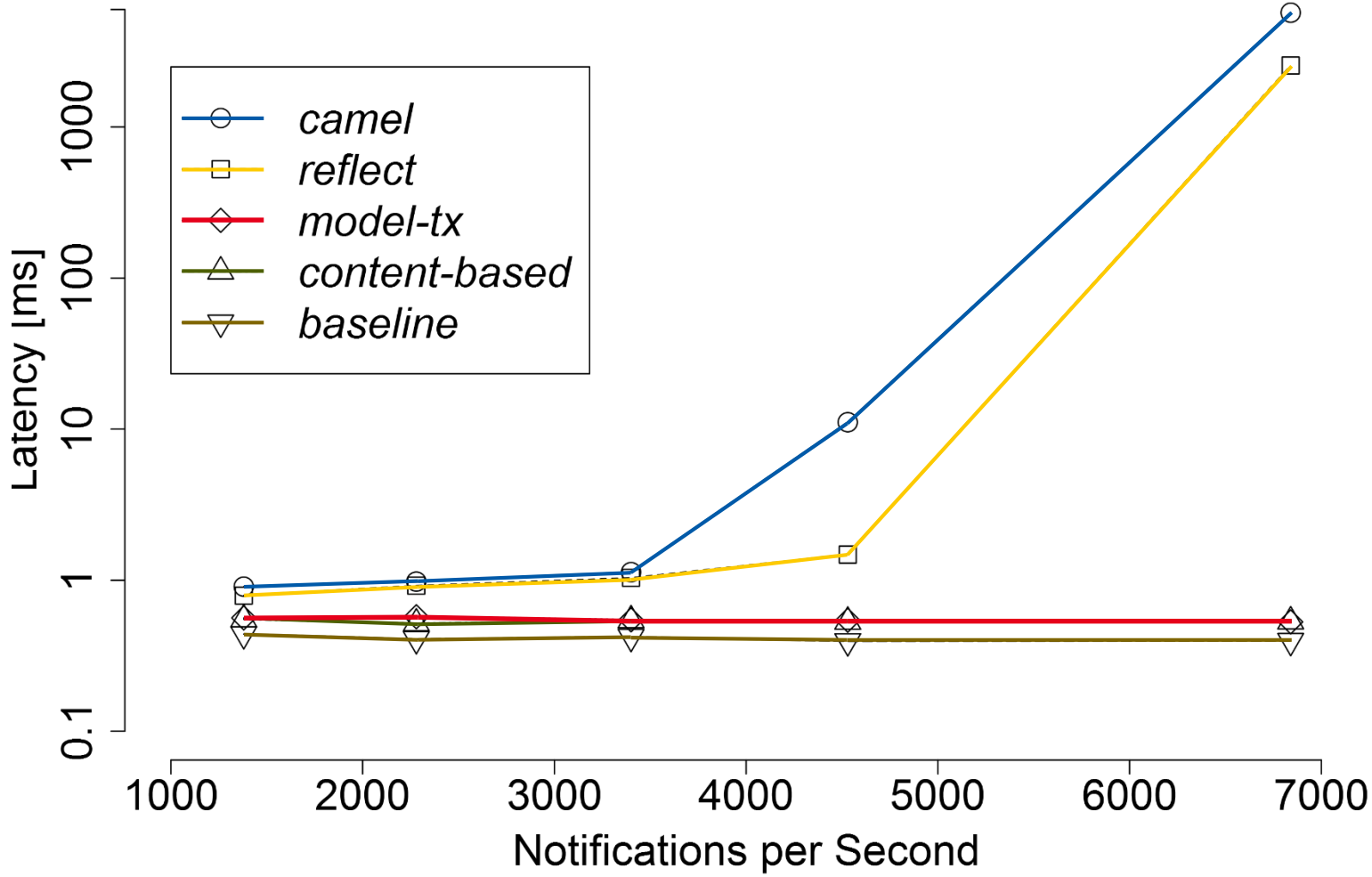
Java properties

```
if (object instanceof USShipment) {
        Shipment s = new Shipment();
        …
        s.setWeight(lb2kg(object.getWeight());
…
```

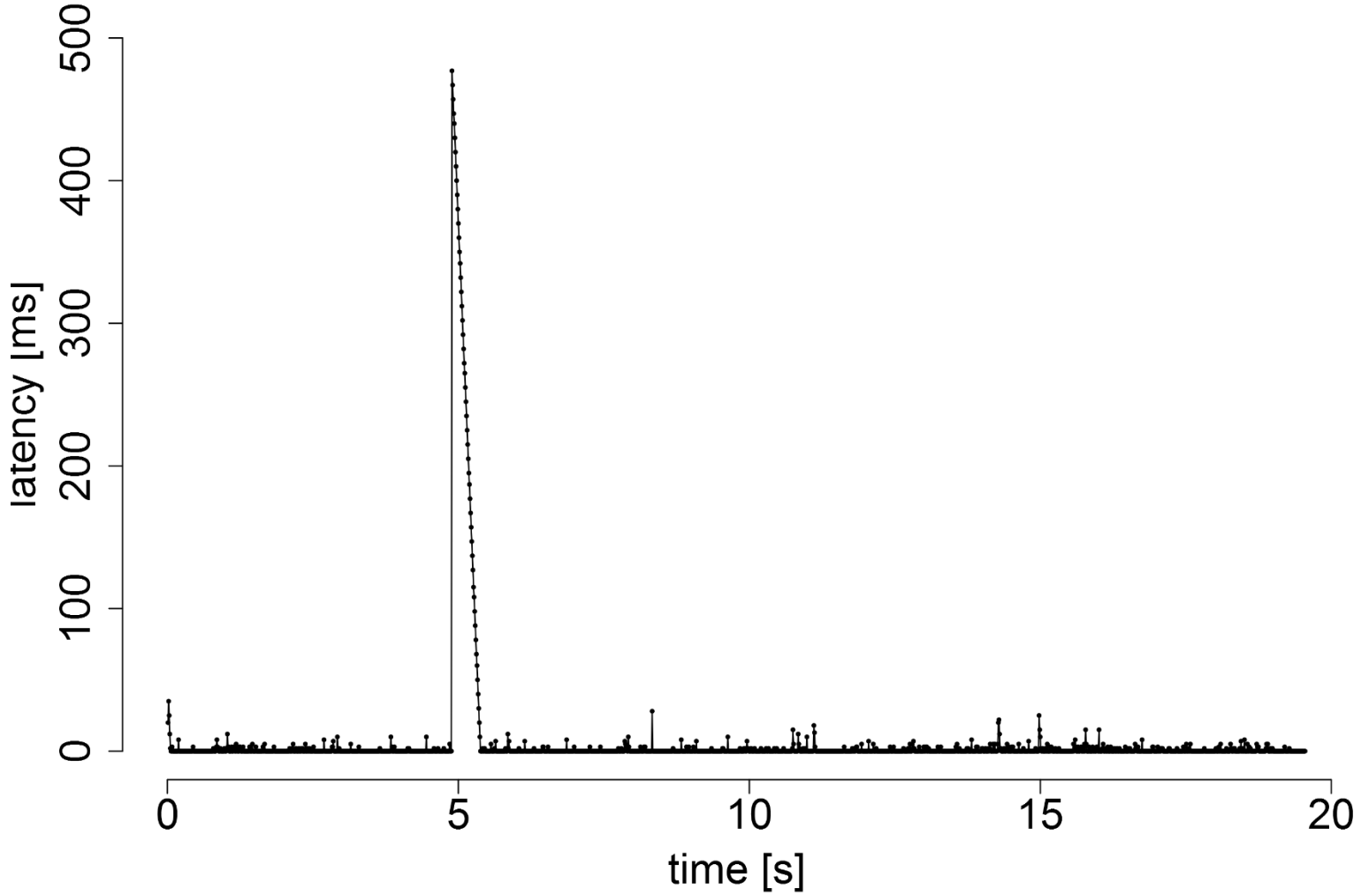
Code

DVS

Performance – Overhead



Performance – Extension



Code Metrics

TECHNISCHE UNIVERSITÄT DARMSTADT

	manual			ACTrESS		
	specify	change	extend	specify	change	extend
SPECjms2007	137	15	92	22	2	1
marketcetera	108	9	37	16	2	1
HTM	237	16	133	21	2	1
DRADEL	417	16	139	30	3	2
ERS	351	12	117	21	3	2

lines of code

DVS

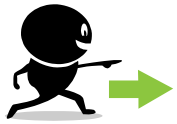
Conclusion

Heterogeneity is a challenge in push-based systems

Model to overcome interpretation problems
Techreport details formal model

Contexts as building blocks & reusable entities

Implementation: flexible and fast



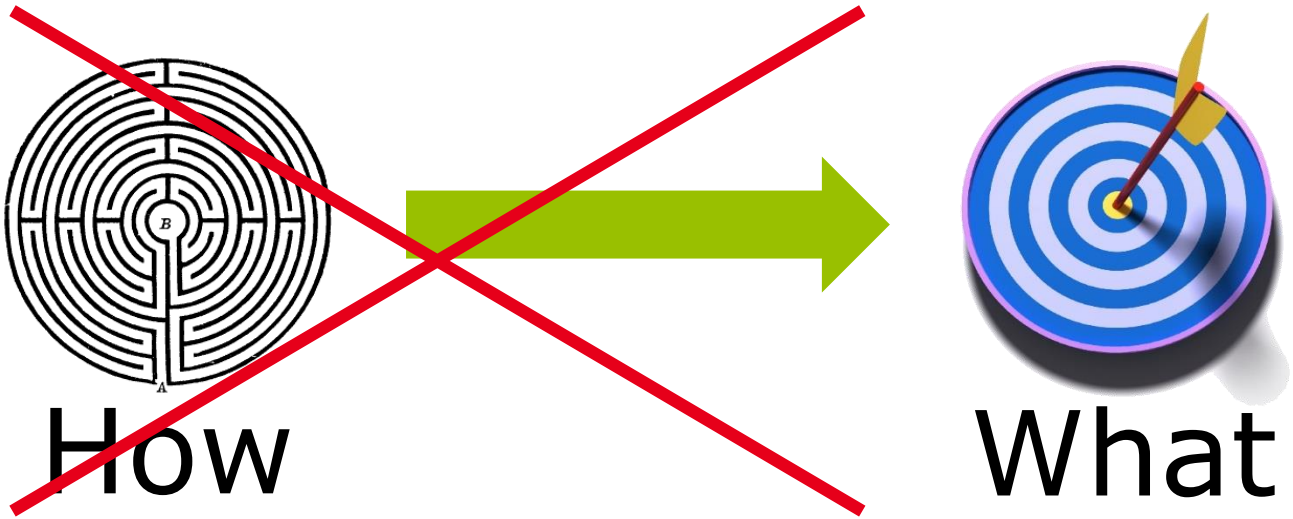
Future work:

- optimal decentrailzed transformation
- code generation improvements
- coordinated stateful transformations

http://www.dvs.tu-darmstadt.de/research/events/actress/

CONTEXTUALIZING EVENTS


Event Processing
=
Complexity


How
What

How it is done today

```
insert into UserStream
  select * from pattern[timer:interval(0)],
  sql:db1 ['select * from user']

insert into RoomStream
  select * from pattern[timer:interval(0)],
  sql:db1 ['select * from room']

insert into EmployeeEvent
  select p.id, p.position
  from PersonPositions as p, UserStream as u
  where u.id = p.id
    and u.status = 'employee'

insert into NonPublicEvent
  select p.id, p.position, r.room
  from PersonPositions as p, RoomStream as r, UserStream u
  where inside(p.position, r.coordinates)
    and u.id = p.id
    and u.status = 'guest'
    and r.security != 'public'

select * from pattern
  [a=NonPublicEvent and not b=EmployeeEvent]
  where within(a.position, b.position, 5)
```

What wir gerne hätten

TECHNISCHE UNIVERSITÄT DARMSTADT

IF

person A with status='unknown' IS INSIDE

room R with security='restricted' AND

person B with status='known' IS NOT WITHIN 5m of A

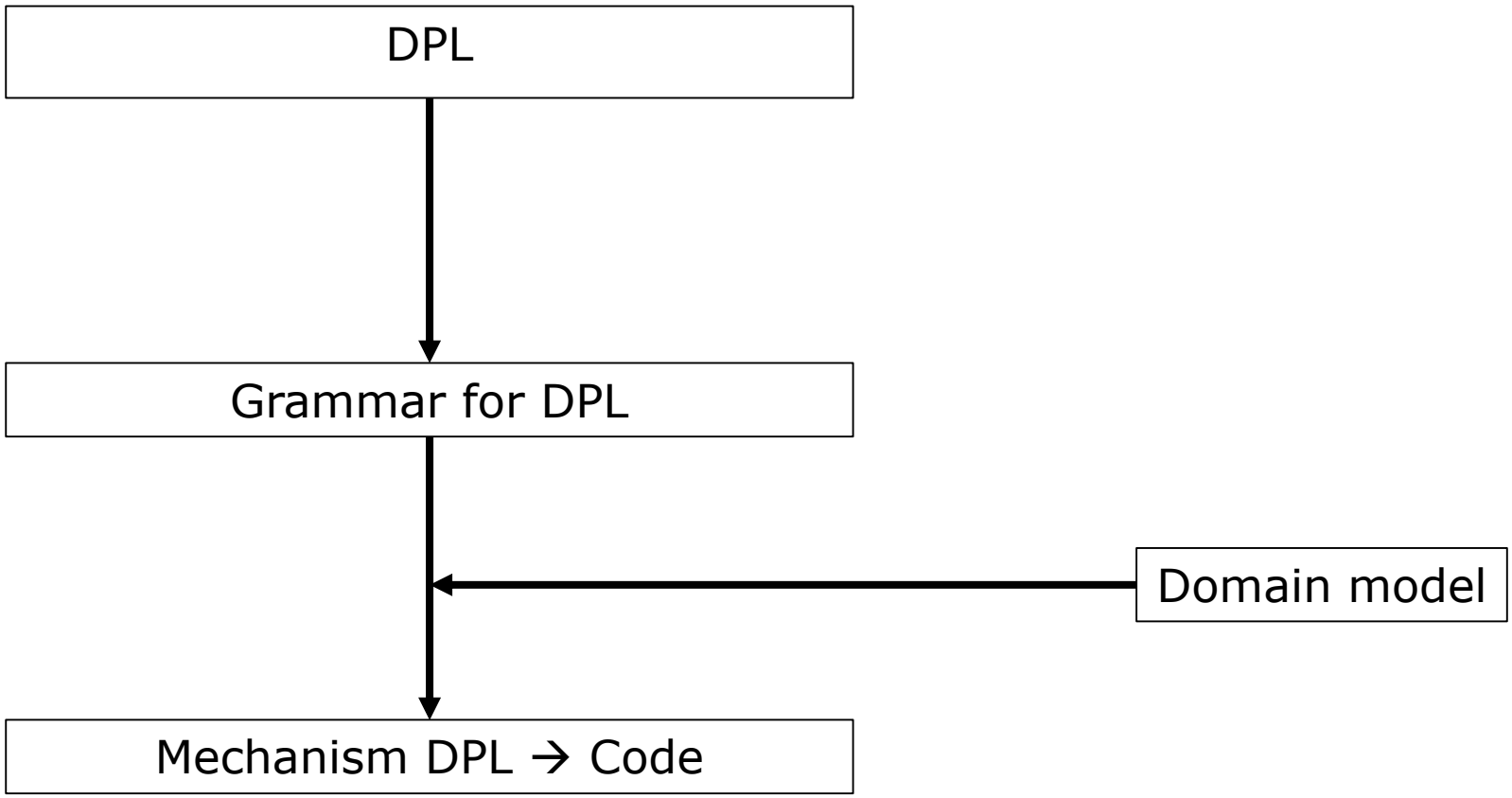
OR B IS NOT INSIDE R

THEN

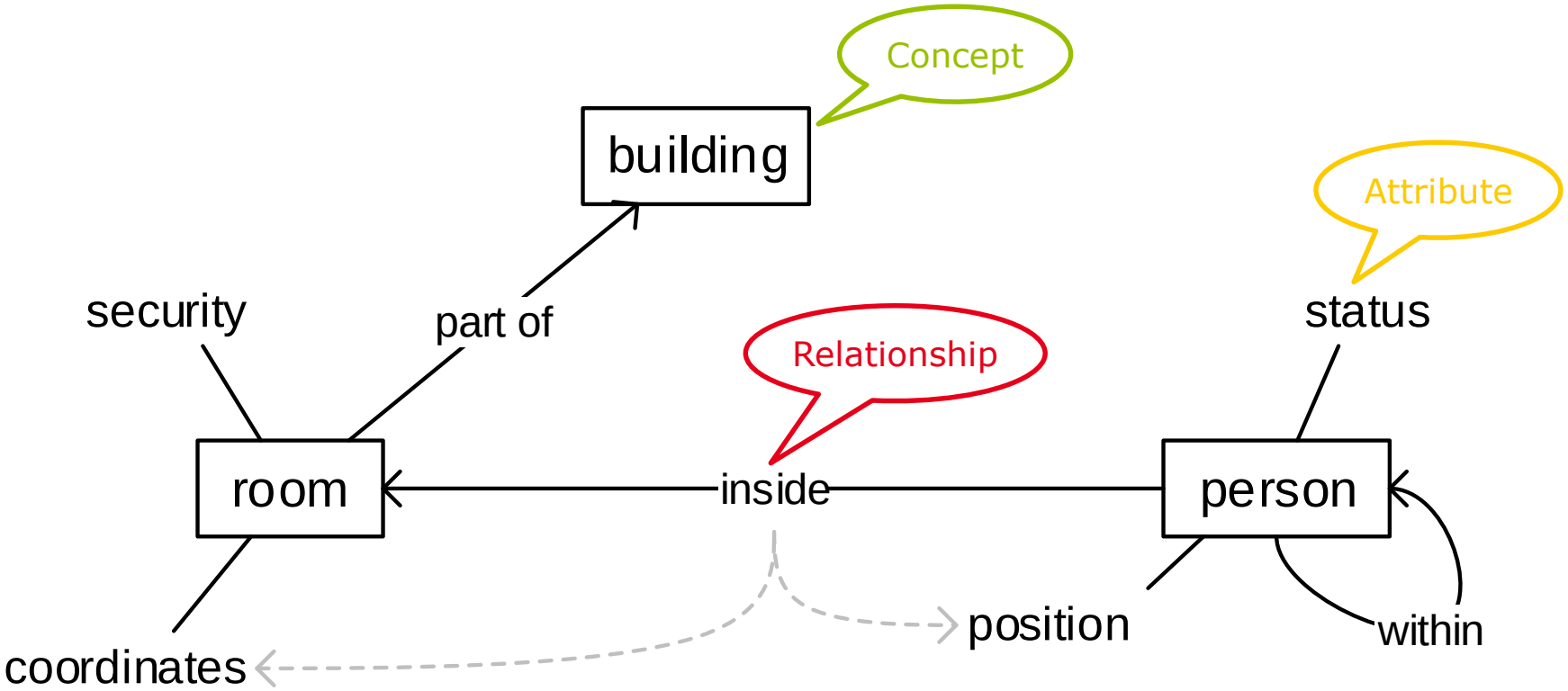
raise alarm

DVS

Components

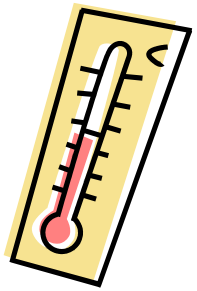


Domain model



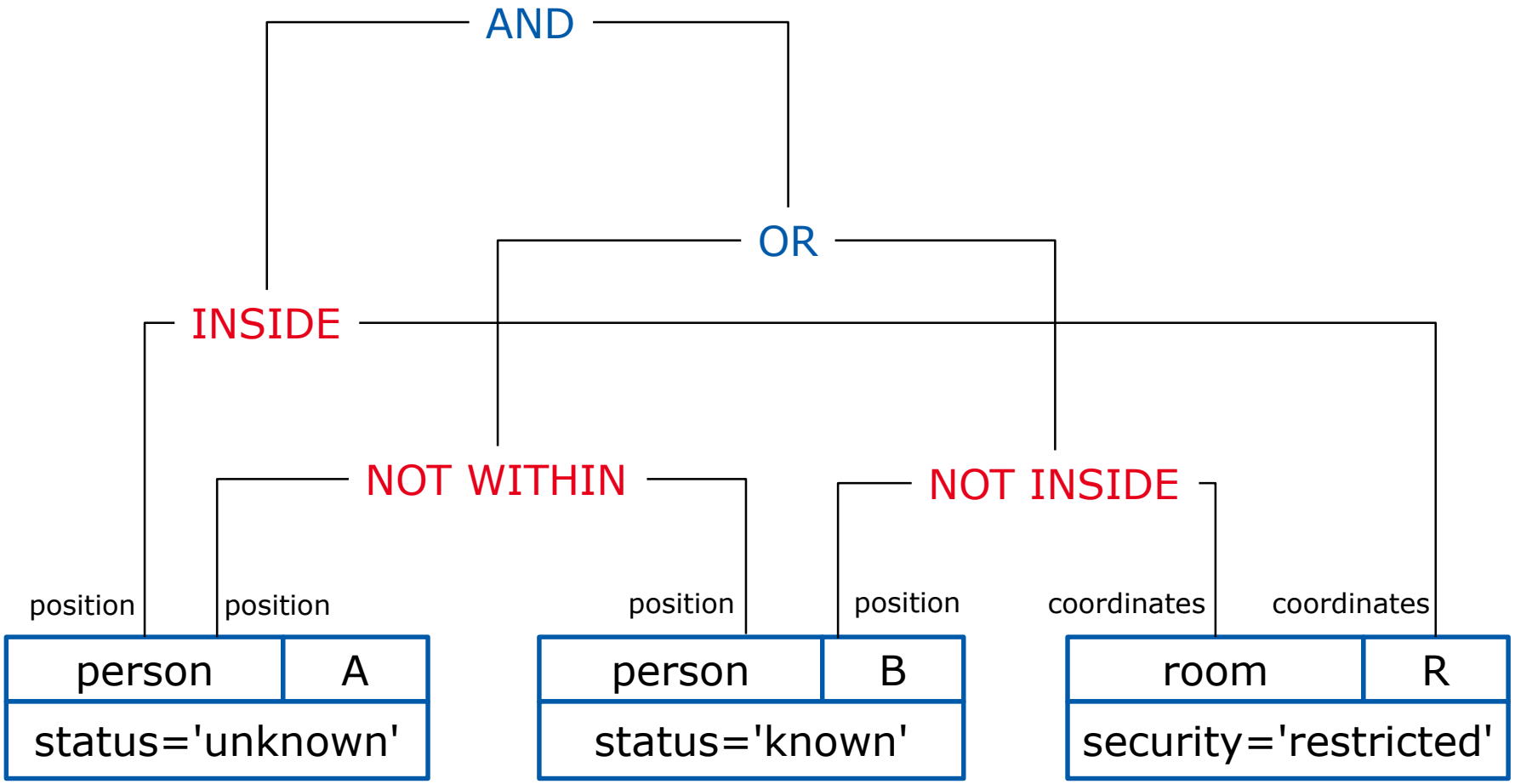
Static vs. dynami attributes

static: person.status



dynamic: person.position

Policy Logic Tree



Handling NOT

TECHNISCHE UNIVERSITÄT DARMSTADT

We cannot constantly generate events of who is not within 5m of someone else.

➔ Modularize processing:

- leaf-checking enrichers
- relationship-checking enrichers
- AND/OR checking enrichers

NOT-handling enrichers are off by default

Switch them on only for a short time when other events trigger AND/OR, and pass identity of known concepts.

Example enrichment

```
<RFIDtag-detected>
    <id>5</id>
    <at>(5, 3)</at>
</RFIDtag-detected>
```

```
<guest>
    <id>5</id>
    <at>(27.4, 40.3)</at>
</guest>
```

```
<guest-inside-restricted>
    <id>5</id>
    <at>(27.4, 40.3)</at>
    <room>205</room>
</guest-inside-restricted>
```

violates NOT
→ deactivate

```
<employee-inside-restricted>
    <id>29</id>
    <at>(26.7, 41.2)</at>
    <room>205</room>
</employee-inside-restricted>
```

NOT within
→ keep waiting

```
<employee>
    <id>29</id>
    <at>(26.7, 41.2)</at>
</employee>
```

```
<RFIDtag-detected>
    <id>29</id>
    <at>(4, 3)</at>
</RFIDtag-detected>
```