

CEP am Beispiel von Esper

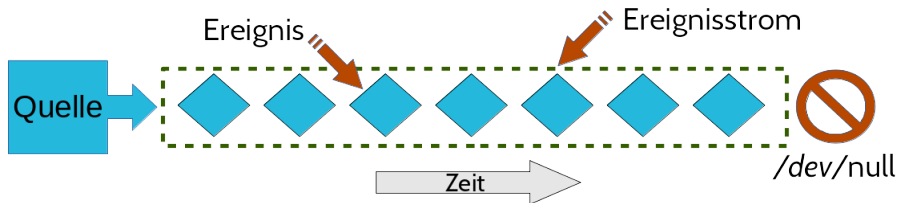
Seminar - Event Driven Architectures

Martin Steinbach

Januar 2019

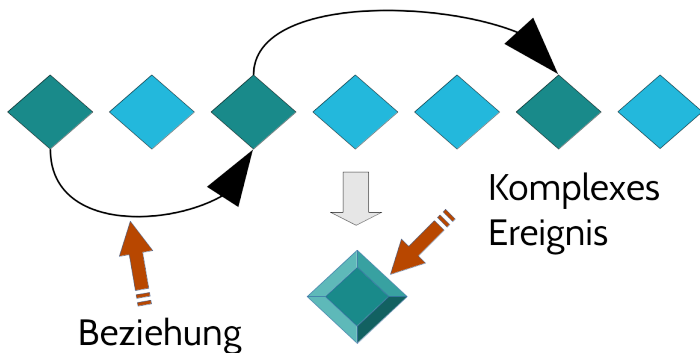
Complex Event Processing

- Analyse von unendlichen Ereignisströmen



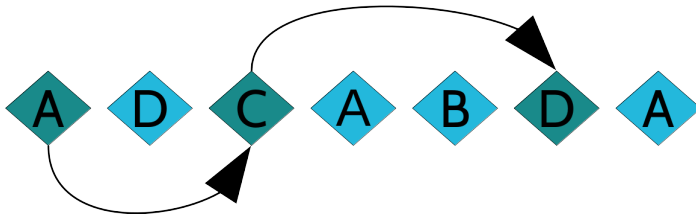
Komplexe Ereignisse

- Ereignismuster beschreiben Beziehungen zwischen Ereignissen
- Komplexe Ereignisse sind Abstraktionen von Ereignismengen



Ereignismuster

A vor C vor D



Ereignistypen



Ereignis in CEP

Metadaten	
Ereignistyp	Kursänderung
Ereignisquelle	Frankfurt
Zeitstempel	21:19:55
ID	98127634
Kontextinformation	
Name: GCME AG	
Aktueller Kurs: 42.1	

- Aufbereitete Rohdaten
- Strukturiert
- Maschinenlesbar
- Atomar

ESPER I

Free Software (GPLv2)

CEP-Engine

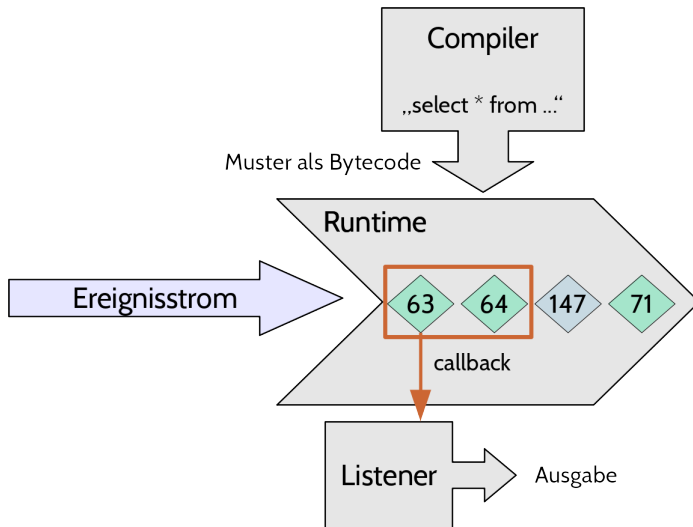
Event **P**rocessing **L**anguage

Java und C#

EPL → Bytecode

SQL - MATCH_RECOGNIZE

ESPER II



Musterdefinition - EPL/SQL

Event Processing Language

- Syntax ähnlich SQL
- Beschreibung durch Aussagenlogik + spezielle Operatoren
- Operiert auf unterschiedlichen Ereignistypen

SQL-Erweiterung: MATCH_RECOGNIZE

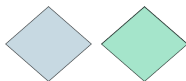
- Einbettung in EPL-Anfrage
- Standardisiert in SQL:2016 (ISO/IEC 9075:2016)
- Beschreibung durch reguläre Ausdrücke
- Operiert auf einem Ereignistyp

EPL-Operatoren

wichtige Operatoren von EPL

A and B:	$A \wedge B$, Reihenfolge egal
A or B:	$A \vee B$, Reihenfolge egal
A -> B:	Sequenz, B tritt zeitlich nach A ein
(A -> B) and not C:	kein C vor A oder B
every A:	detektiert alle A im Strom
[5] A:	Repeat, Detektion nach 5 A

Esper - Ereignistypen

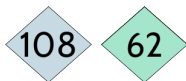


Ereignistyp: StockTick

symbol: 'IBM'/'SAP'

price: Integer

message: -



NewsTick

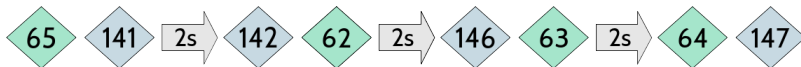
'IBM'

-

'good'/'bad'



EPL - einfache Anfragen



EPL - einfache Anfragen

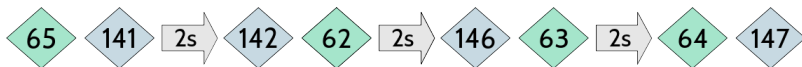
```
select PARAMETER from SOURCE where CONDITION;
```

Eine einfache Anfrage

```
select * from StockTick where price > 100;  
=  
select * from StockTick(price > 100);
```



EPL - einfache Anfragen

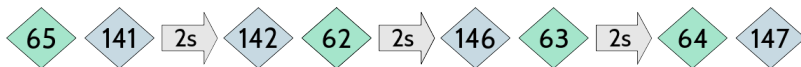


Aggregation

```
select avg(price) from StockTick(symbol='IBM')
```



EPL - einfache Anfragen



Aggregation

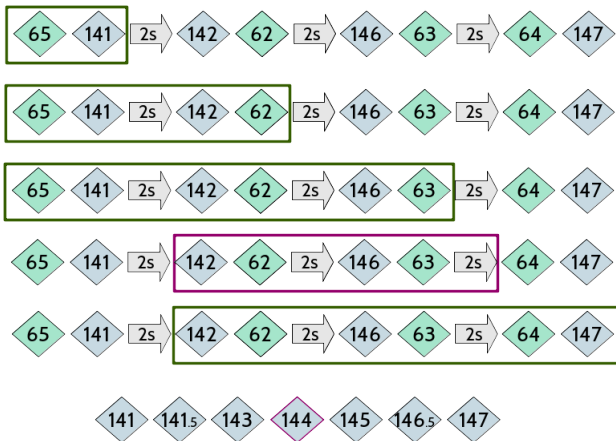
```
select avg(price) from StockTick(symbol='IBM')
```



- Operationen auf ∞ -Datenstrom
- Anfragen würde ewig Ereignisse erstellen

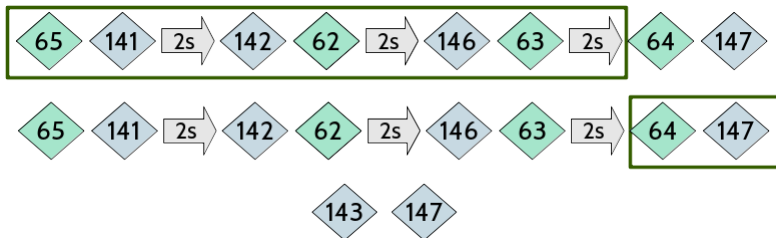
Fenster - einfache Zeitfenster

```
select avg(price) from StockTick.win:time(6 sec) where  
symbol='IBM'
```



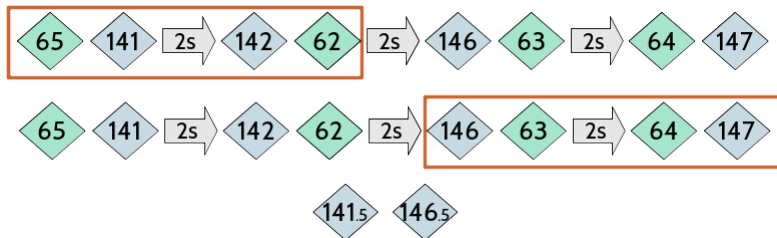
Fenster - batch-Zeitfenster

```
select avg(price) from StockTick.win:time_batch(6 sec)
where symbol='IBM'
```



Fenster - batch-Längenfenster

```
select avg(price) from StockTick.win:length_batch(4) where  
symbol='IBM'
```

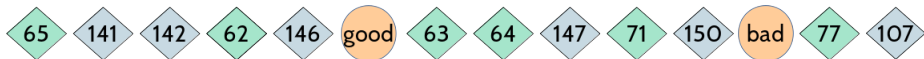


Ereignistypbasierte Regeln

pattern[]

- Betrachtung unabhängig von Zeit und Länge
- boolesche Ausdrücke + Sequenzoperator: \rightarrow
- `pattern[A \rightarrow B]`
 - ▶ A, B sind Ereignistypen
 - ▶ a,b deren Instanzen
 - ▶ B tritt zeitlich nach A ein
 - ▶ A ist Initiator, B ist Detektor
- every-Operator (*Event Consumption Modes*)

Ereignistypbasierte Regeln



```
select b from pattern[a=StockTick(symbol='IBM') →  
                      b=StockTick(symbol='SAP')];
```



Ereignistypbasierte Regeln - Consumption Modes I

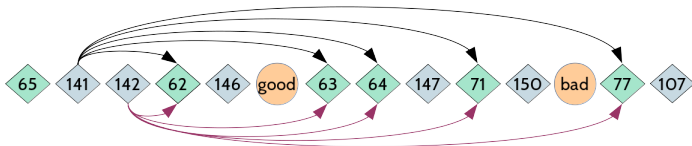


Unrestricted Consumption Mode:

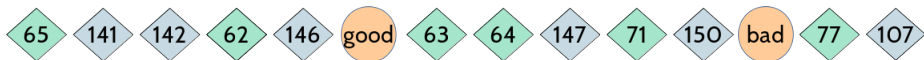
pattern[**every** A \rightarrow **every** B]

select a,b from pattern[**every** a=StockTick(symbol='IBM') \rightarrow
every b=StockTick(symbol='SAP')];

- Jeder Initiator konsumiert alle Detektoren
- sehr große Ergebnismenge



Ereignistypbasierte Regeln - Consumption Modes II

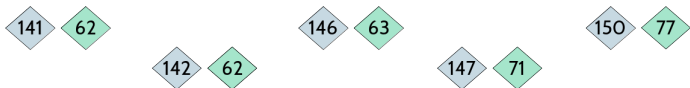


Continuous Consumption Mode:

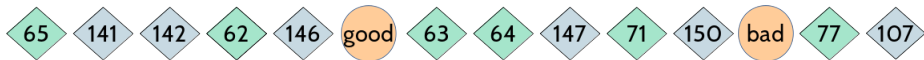
```
pattern[ every A → B ]
```

```
select a,b from pattern[every a=StockTick(symbol='IBM') →  
                        b=StockTick(symbol='SAP')];
```

- Jeder Initiator konsumiert nur nachfolgenden Detektor



Ereignistypbasierte Regeln - Consumption Modes III

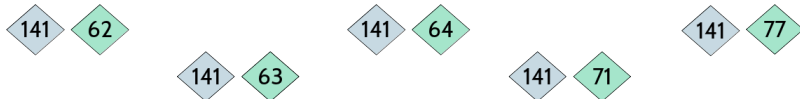


Continuous Detection Mode:

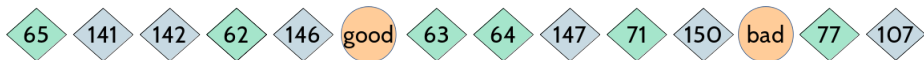
```
pattern[ A → every B]
```

```
select a,b from pattern[ a=StockTick(symbol='IBM') →  
every b=StockTick(symbol='SAP')];
```

- Ein Initiator konsumiert alle nachfolgenden Detektoren



Ereignistypbasierte Regeln - Consumption Modes IV

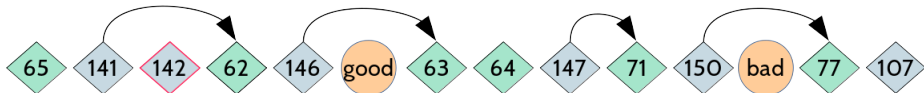


Regular Consumption Mode:

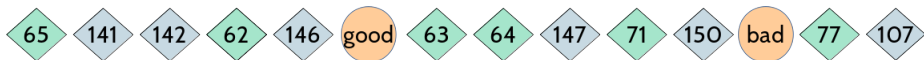
```
pattern[every (A  $\rightarrow$  B)]
```

```
select a,b from pattern[every (a=StockTick(symbol='IBM') $\rightarrow$   
b=StockTick(symbol='SAP'))];
```

- Initiator sucht nach Detektor
- Initiatoren auf dem Pfad werden nicht betrachtet

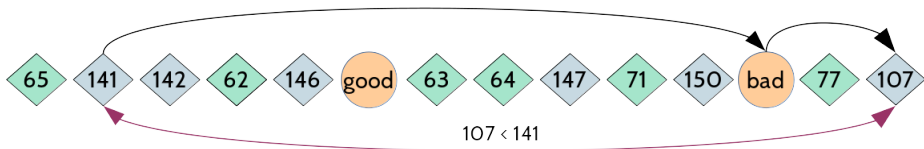


Ereignistypbasierte Regeln - Beispiel

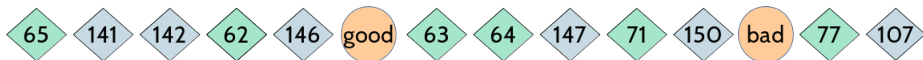


Wollen herausfinden, ob Nachrichtenmeldung Einfluss auf Aktienkurs hat

```
select c from pattern[a=StockTick(symbol='IBM') ->  
    b=NewsTick(symbol='IBM' and message='bad') ->  
    c=StockTick(symbol='IBM' and price < a.price)];
```



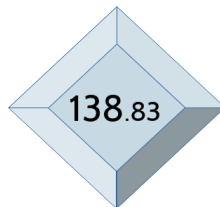
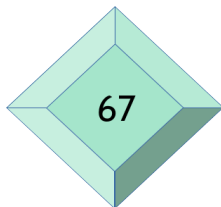
Komplexe Ereignisse



Wollen neues Ereignis: Mittelwert der letzten 6 Aktien einer Firma

```
insert into TargetEvent(symbol, average)
select symbol, avg(price)
from StockTick#groupwin(symbol)#length_batch(6);

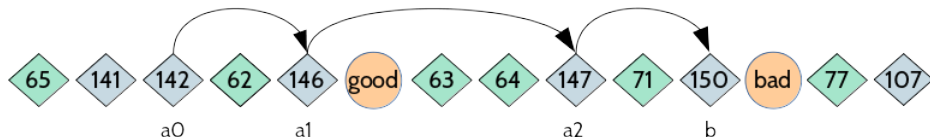
select * from TargetEvent;
```



SQL: Match_Recognize



```
select * from StockTick match_recognize (  
  partition by symbol  
  measures A[0].price as a0,A[1].price as a1, A[2].price as  
  a2, B.price as b  
  pattern (A{2,3} B)  
  define  
  A as A.price < 150, B as B.price >= 150)
```



- Hohe Einstiegshürde
 - ▶ EPL gut Dokumentiert
 - ▶ Software eher nicht
- Sehr großer Leistungsumfang
 - ▶ Doku > 900 Seiten
 - ▶ TIMTOWDI
- EPL-Prüfung mittels EPL-Online¹
- Sehr leistungsfähige Architektur

¹<https://esper-epl-tryout.appspot.com/epltryout/mainform.html>

Danke für die Aufmerksamkeit.

`martin.steinbach@uni-rostock.de`