


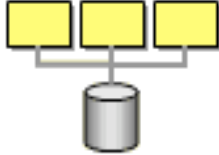

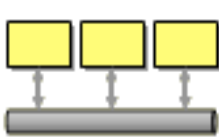


Complex Event Processing (CEP)

Christian Dedek
Papick G. Taboada

- Integrationsprojekte als historischer Hintergrund für SOA und EDA
- Probleme bei der Integration von Systemen
 - Netzwerke sind nicht zuverlässig
 - Netzwerke sind langsam
 - Systeme sind grundsätzlich verschieden
 - Änderungen sind unvermeidbar

- Dateiübertragung
- Gemeinsame Datenbank
- Entfernte Methodenaufrufe
 - Remote Procedure Calls...
- Nachrichtendienste

Integration Styles	
	Introduction to Integration Styles
	File Transfer
	Shared Database
	Remote Procedure Invocation
	Messaging

<http://www.enterpriseintegrationpatterns.com/toc.html>

- Grundprinzipien
 - sehr lose Kopplung
 - basiert vollständig auf Nachrichten
 - abstrahierter Nachrichtentransport
 - Unabhängigkeit der Eventbearbeitung von Ort und Weg zwischen den beteiligten Komponenten

- EDA konzentriert sich auf Anwendungen nach folgendem Muster
 - Events müssen zeitgerecht in Ihrem Eingang erkannt
 - konsumiert
 - und anschließend folgerichtig darauf reagiert werden
- Event getriebene Anwendungen lassen sich in natürlicher Sprache regelartig komprimieren
 - - „wenn .. dann...“
 - **When-clause**
 - **Handler code**

- SOA
 - Services sind Remote Procedure Calls
 - **Point-To-Point Kommunikation**
 - Lösungen für die Probleme Komplexität und Verwaltbarkeit:
 - **Orchestrierung**
 - **Monitoring**
- EDA
 - Nachrichtenbasiert
 - **Geringe Kopplung**

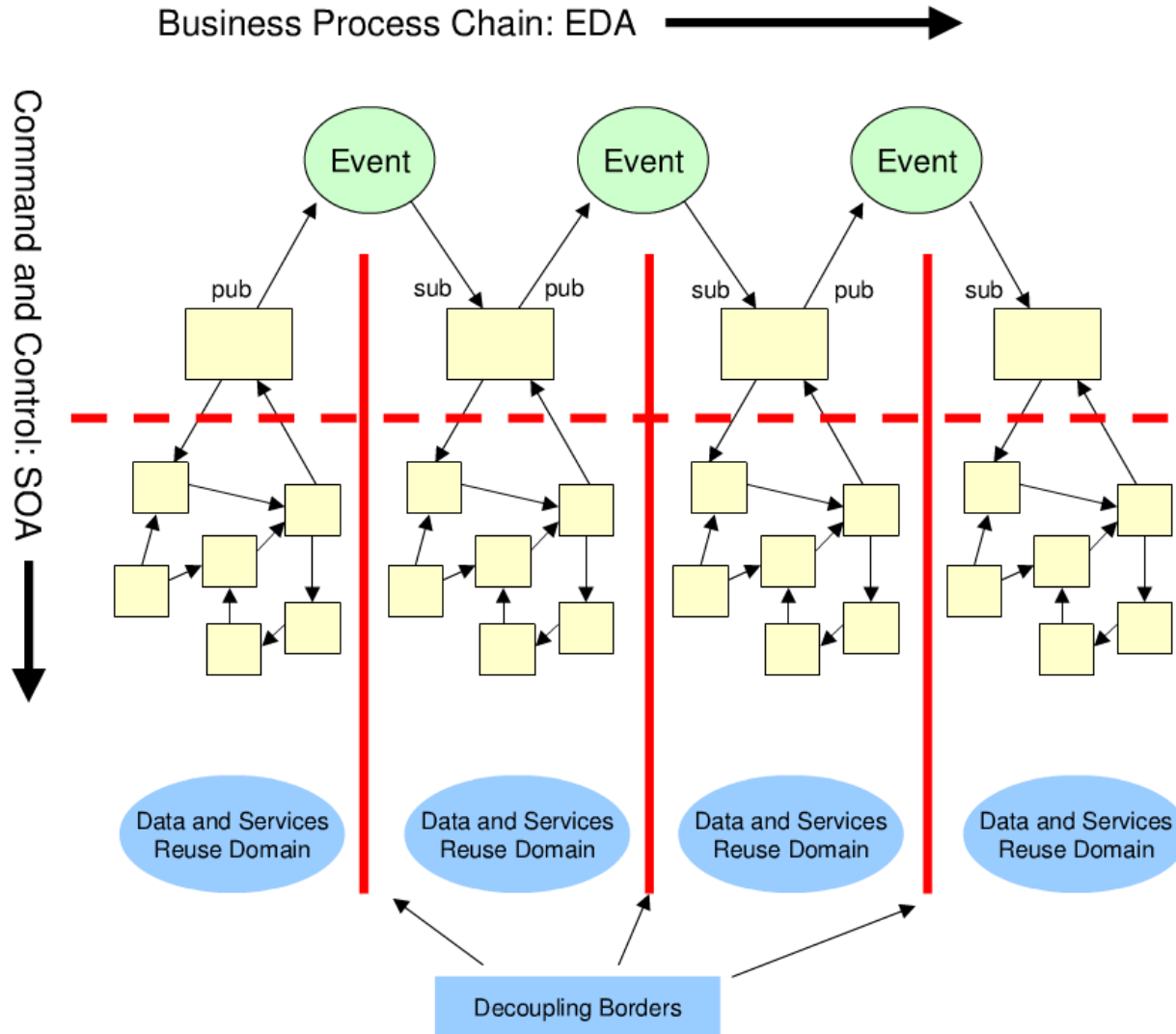
- EDA konzentriert sich auf Anwendungen mit Wahrnehmungsreaktionsmustern
 - Oft verbunden mit komplexer Transaktionalität
 - Sehr viel „loser“ als typische SOA
- SOA Paradigma - „Dienst anbieten und Nachfragen bearbeiten“
 - SOA ermöglicht, Events zwischen Systemen, Protokollen und Rollen auszutauschen

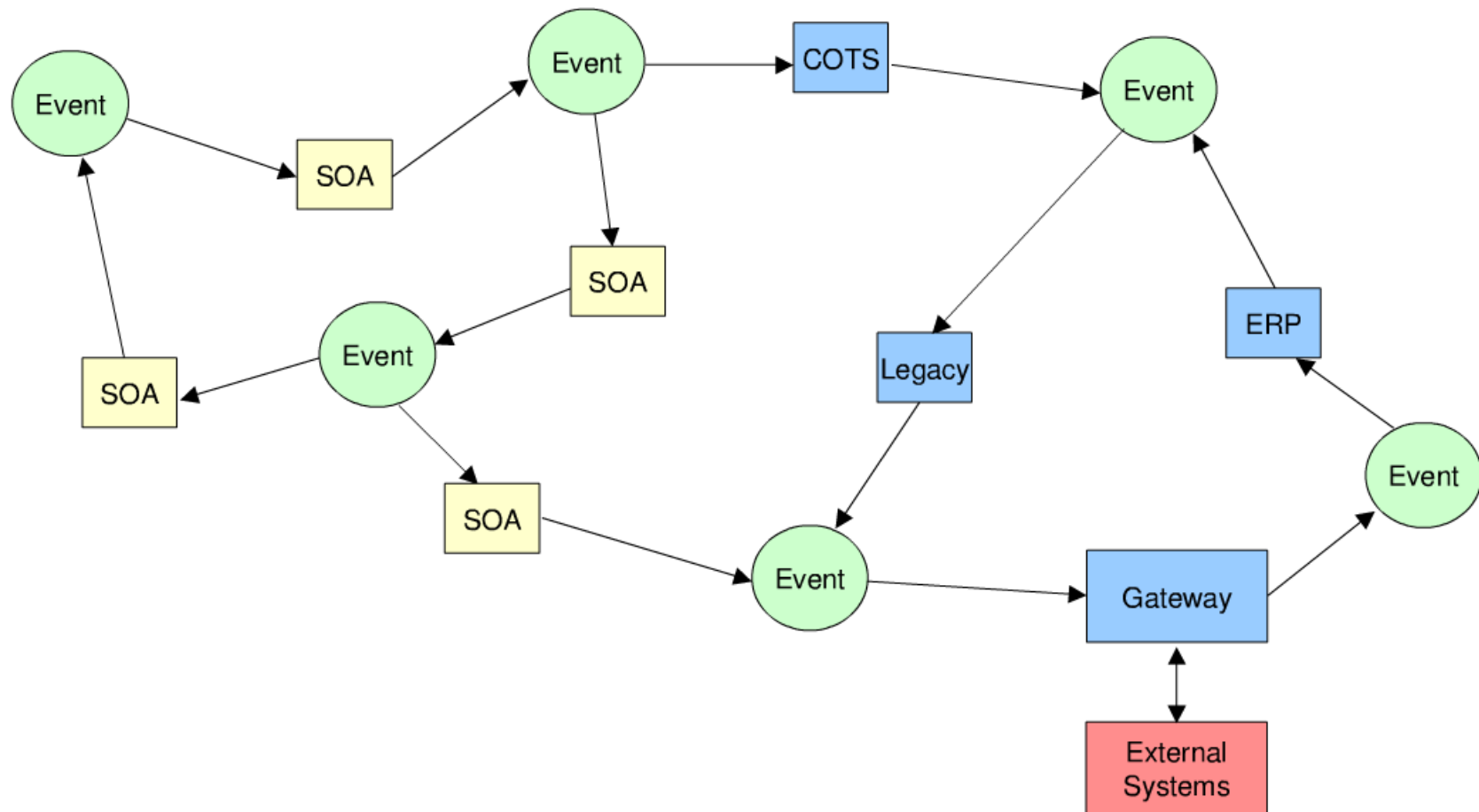
EDA extends SOA...



- Use SOA:
 - Vertical interaction, request and reply processes

- Use EDA
 - Horizontal communication between tiers in a process chain





- Unter "Real Time Business" versteht man die zeitnahe Auslieferung und Evaluierung von wirtschaftlichen Ereignissen.
- Zeitnah?
 - Informationslatenz
 - Analyzelatenz
 - Aktionslatenz

- Wirtschaftliche Potentiale entdecken
 - Marktbedürfnisse münden in Nachfrage
 - Früherkennung schafft Wettbewerbsvorteile
- Unterstützende Trends
 - Zunehmende Verfügbarkeit elektronischer Daten
 - Elektronische Bereitstellung von Detailinformationen
 - **Feinkörnige Daten**
 - **Häufige Updates**
 - Zeitnaher Zugriff auf Reaktionsschnittstellen möglich

- Sehr hohe Anforderungen
 - Extrem hohe Anzahl an Events (>> 100.000)
- OLTP Architekturen können nicht so viele Ereignisse verarbeiten
- „High transactions per second“
 - eBay, Amazon 1,000 - 10,000 TPS
- „Medium transactions per second“
 - International web application 100 - 1,000 TPS
- Low transactions per second
 - Small internal OLTP 10 - 100 TPS

- Auf der Seite des „Transaction Processing Performance Council“ liegt der aktuelle Spitzenwert bei 68,000 Transaktionen pro Sekunde (TPC-C). Dabei handelt es sich um einen Server mit 64 Intel 1.6 GHz Prozessoren, also insgesamt 128 Kernen.
- 11 978 134 US\$ = 8.09552176 million Euros

- „By 2011, a new generation of application platforms stemming from the convergence of diverse XTP-enabling technologies will supersede Java EE and .NET as the platforms of choice for large-scale, business-critical applications (0.8 probability).“
- „By 2010, support for advanced, event-centric programming models will be a standard feature in all application platforms aimed at XTP applications (0.8 probability).“
 - Quelle: Gartner RAS Core Research Note G00146107 Q1/2007

- Beobachtung einer Zustandsänderung
 - Quittierung einer Autorisierungsprüfung
 - Antwortzeit der letzten Anfrage
 - Messwert eines Sensors im Haus
 - Aktienpreisänderung
- benötigen eine technische Repräsentation
 - Schlüssel/Wert- Paare
 - XML
 - POJO

- KFZ-sensoren
 - Ermittlung von Betriebsgrößen durch verschiedene häufig über ein Bussystem verbundener Sensoren
 - **Drehgeschwindigkeit des Rades**
 - **Beschleunigung in allen drei Raumachsen**
 - **Luftdruck der Reifen**
 - **Aussenlufttemperatur**
 - **Luftfeuchtigkeit**
 - **Sitzbelastung**
 - **Lambda-Wert**
- Anwendungsbeispiele
 - ABS
 - Einspritzsysteme
 - Airbagabschaltung
 - Verhaltensforschung?

- SEP
 - Simple Event Processing
- ESP
 - Stream Event Processing
- CEP
 - Complex Event Processing

- Ein Ereignis kann als wichtige Zustandsänderung in einer Nachrichtenquelle angesehen werden.
- Diese Ereignisse lösen durch ihr Auftreten Prozesse in den Systemen aus, die die Nachricht empfangen werden.
- Das ist Verarbeiten von Nachrichten, wie es in der „Java Messaging Service“ Spezifikation beschrieben ist.

- Unter „Event Stream Processing“ versteht man die flussartige Bearbeitung der eingehenden Nachrichten.
- Bei der Abarbeitung der Nachrichten ist nicht jede einzelne Nachricht ausschlaggebend, sondern der Fluss als solches. So wird zum Beispiel erst reagiert, wenn ein Temperatursensor für einen bestimmten Zeitraum durchgehend eine bestimmte Temperatur über- oder unterschritten hat.

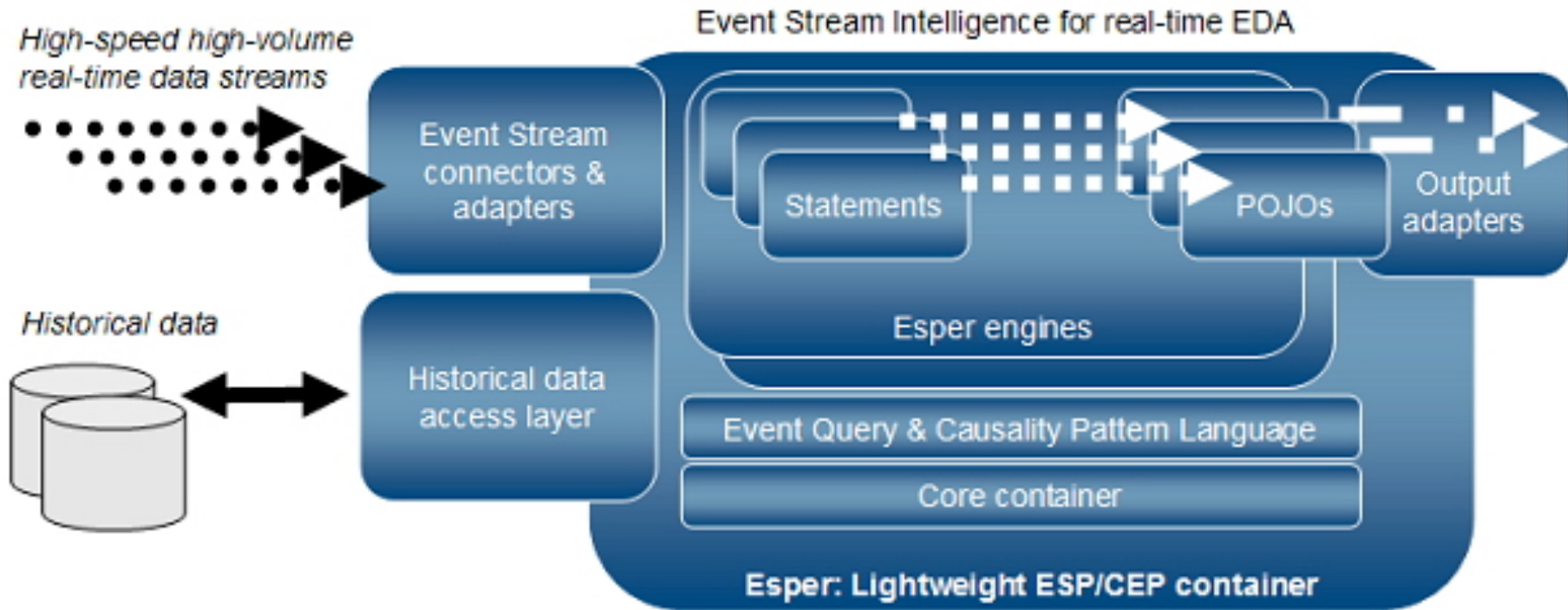
- Unter CEP versteht man die Erkennung von komplexen Mustern innerhalb eines oder sogar zwischen verschiedenen Ereignisströmen. So haben eventuell Ereignisse wenig Relevanz im einzelnen, sind aber in bestimmten Zusammenhängen Indiz eines besonderen Ereignisses.
- Ereignisse können in kausaler, temporaler oder räumlicher Korrelation auftreten. Die einzelnen Ereignisströme können ein sehr hohes Aufkommen von Ereignissen aufweisen.

- What is a complex event?
 - It is an event that could only happen if lots of other events happend.
 - The Power of Events
An introduction to complex event processing in distributed enterprise systems

by David Luckham

- Esper ist ein Softwarebaustein für CEP und ESP Anwendungen, der in Java als Esper und .NET als Nesper bereitgestellt wird.
- Esper erlaubt die zeitnahe Entwicklung von Anwendungen mit hohem Aufkommen eingehender Ereignisse bzw. Nachrichten. Dabei können Ereignisse auf verschiedene Arten in Echtzeit analysiert, gefiltert und konsumiert werden.
 - Esper Homepage: <http://esper.codehaus.org>
 - Lizenz: dual licensing, GPL + commercial license through Espertech
 - Umfangreiche Dokumentation
 - Viele Beispiele
 - Abfragemuster auf der Homepage
 - Aktuelle Version: 2.0

Big picture



© EsperTech

- Kontinuierliche Bearbeitung
- Listener werden benachrichtigt, wenn sich die Ereignismenge verändert
- Datenbank „inside-out“
 - Nicht Anfragen gegen die Daten ausführen sondern Daten durch die Anfragen schicken.

- Analogie zu SQL
 - Ereignisströme sind Tabellen
 - Ein Ereignis entspringt ein Datensatz
 - Eigenschaften im Ereignis entsprechen Datenfelder
- EQL Queries lassen sich grob in
 - ESP Queries und
 - CEP Queries
 - einteilen.

```
String stmtText =
    "insert into ThroughputPerFeed " +
    " select feed, count(*) as cnt " +
    "from " + FeedEvent.class.getName() + ".win:time_batch(1 sec) " +
    "group by feed";

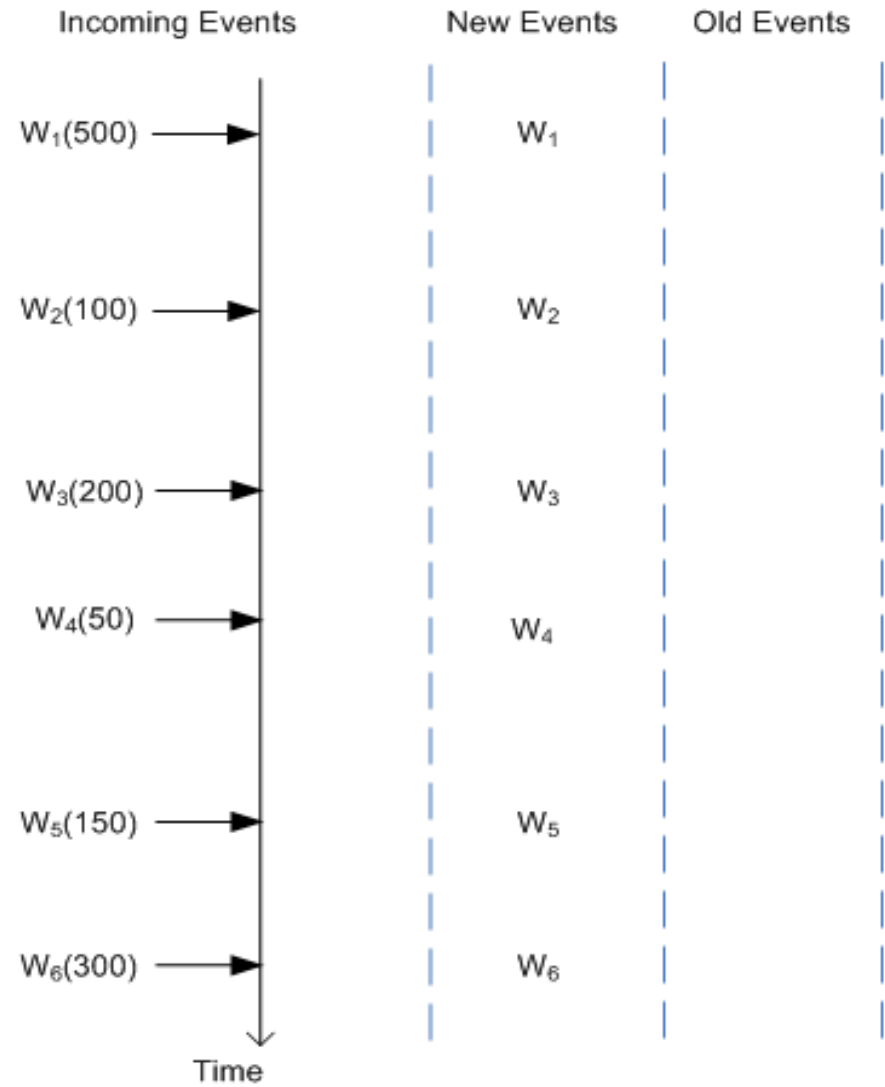
EPServiceProvider engine = EPServiceProviderManager.getDefaultProvider();
EPStatement stmt = engine.getEPAdministrator().createEQL(stmtText);

stmt.addListener(new MyListener());

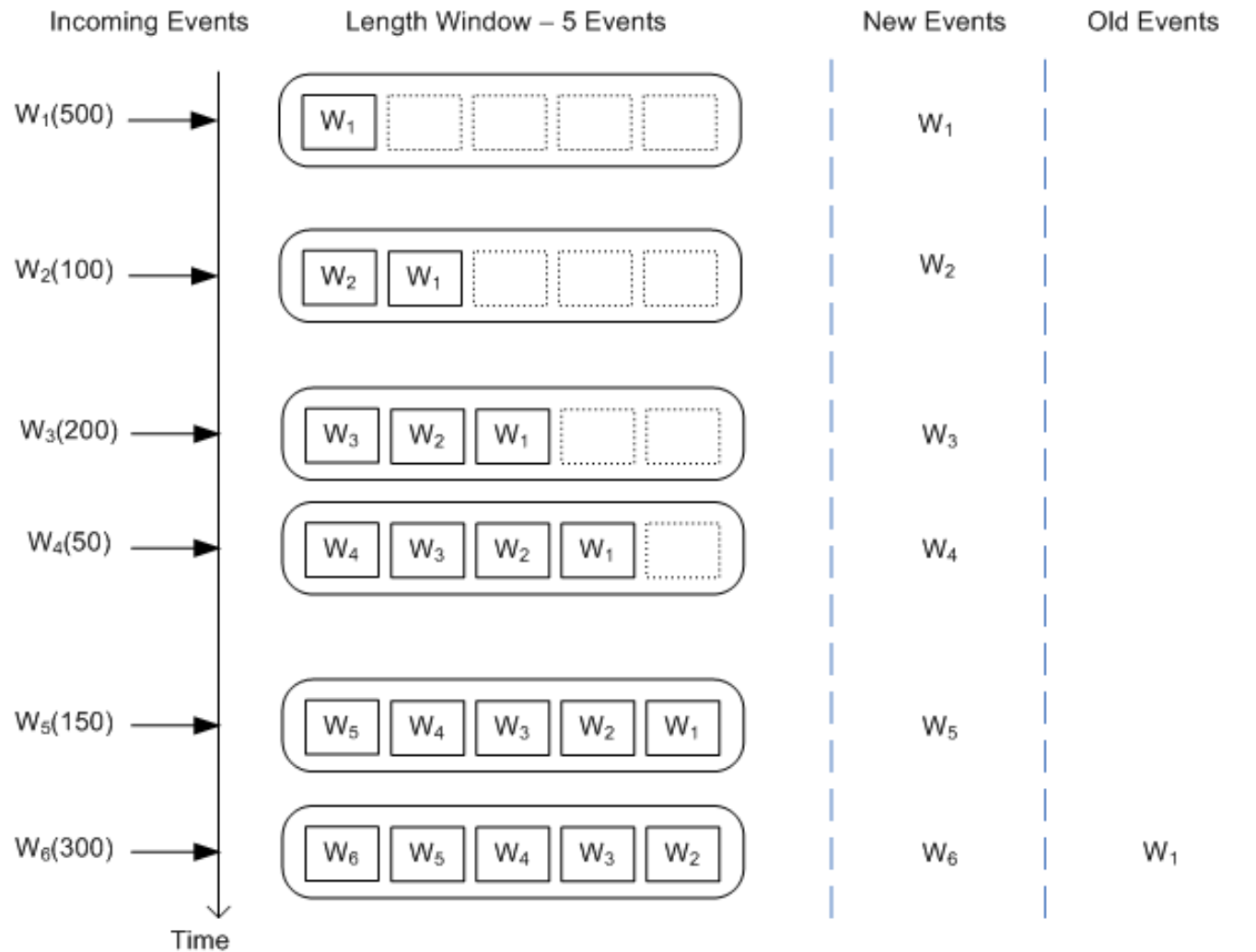
while(true)
{
    FeedEvent event;
    event = new FeedEvent(FeedEnum.FEED_A, "IBM", 70);
    engine.getEPRuntime().sendEvent(event);
    event = new FeedEvent(FeedEnum.FEED_B, "IBM", 70);
    engine.getEPRuntime().sendEvent(event);
}
```

- Abfragen über
 - Einzelne Ereignisse
 - Ereignisse in einem Zeitfenster
 - Ereignisse in einem Mengen-Fenster

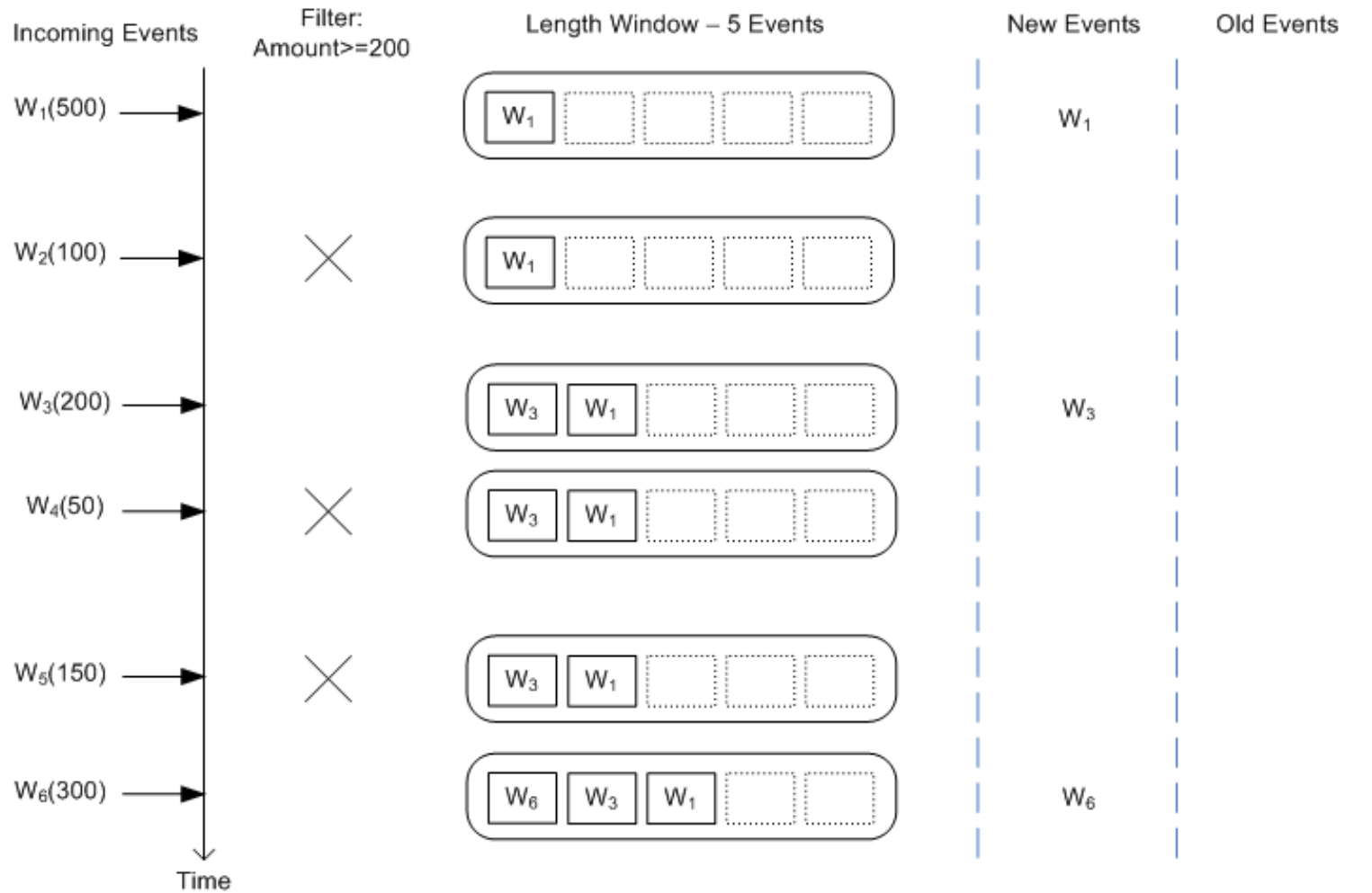
select * from Withdrawal



select * from Withdrawal.win:length(5)



**select * from
Withdrawal(amount>=200).win:length(5)**



- Abfragen über
 - Muster zwischen Ereignisse/ Ereignisstöme
 - Verwendeter Begriff: „Pattern“

- Patterns werden über den Konstrukt „pattern [...]“ definiert
 - Wiederholungsangabe mit „every“
 - Logische Operatoren
 - **and, or, not**
 - Zeitliche Operatoren
 - „->“ (**followed-by**)
 - Guards are where-conditions that control the lifecycle of subexpressions.
 - **timer:within**
 - Observers observe time events as well as other events.
 - **timer:interval**
 - **timer:at**

- Ereignisse:
 - A1 B1 C1 B2 A2 D1 A3 B3 E1 A4 F1 B4

- pattern [every A -> B]
 - {A1,B1}, {A2,B3}, {A3,B3}, {A4,B4}

- pattern [every (A -> B)]
 - {A1,B1}, {A2,B3}, {A4,B4}

„Subexpression“

- Ereignisse:
 - A1 B1 C1 B2 A2 D1 A3 B3 E1 A4 F1 B4
- pattern [A -> every B]
 - {A1,B1}, {A1,B2}, {A1,B3}, {A1,B4},
- pattern [every A -> every B]
 - {A1, B1}, {A1, B2},
 - {A1, B3}, {A2, B3}, {A3, B3},
 - {A1, B4}, {A2, B4}, {A3, B4} and {A4, B4}

- (A or B) where timer:within (5 sec)
 - Ein A oder B in den nächsten 5 Sekunden
- (every A) where timer:within(10 sec)
 - Alle A Ereignisse in den nächsten 5 Sekunden
- A -> timer:interval(10 seconds)
 - Nach A 10 Sekunden warten
- every timer:at(5, *, *, *, *)
 - Alle 5 Minuten

- Feed Monitor Case Study
 - In this tutorial we will learn to process a raw market data feed. We will learn to report throughput statistics and to detect when the data rate of a feed falls off unexpectedly. A rate fall-off may mean that the data is stale and we want to alert when there is a possible problem with the feed.

- Our input stream consists of 1 event stream that contains 2 simulated market data feeds. Each individual event in the stream indicates the feed that supplies the market data, the security symbol and some pricing information
 - String symbol;
FeedEnum feed;
double bidPrice;
double askPrice;

Computing Rates Per Feed



- insert into TicksPerSecond
select feed, count(*) as cnt
from
 MarketDataEvent.win:time_batch(1 sec)
group by feed

Rapid fall-off



- We define a rapid fall-off by alerting when the number of ticks per second for any second falls below 75% of the average number of ticks per second over the last 10 seconds.

Detecting a fall-off

- select
 feed,
 avg(cnt) as avgCnt,
 cnt as feedCnt
from
 TicksPerSecond.win:time(10 seconds)
group by feed having cnt < avg(cnt) * 0.75

- Esper Benchmark
 - RFID tracking sample
 - 1000 gruppen zu drei Elementen , 20 Zonen
 - 2000 registrierte Statements
 - ~ 110 000 Events/s

- Business/ System Monitoring
 - Ausführliche Fallstudie bei Esper zu Terminalüberwachung
 - Überwachung von Informationen aus Trackingdevices
 - Echtzeitauswertung von Finanzdaten
 -
- Intelligente Steuerungssysteme
 - Automobil
 - Haussteuerung/ Hausautomatisierung

Zeit für Demo?



- Hello World?
- RFID example?
- Rates per second?

- EDA Architekturparadigma für „Real Time Business“- Anwendung
 - Anwendungen mit Wahrnehmungsmustern
 - komplementär zu SOA
- CEP/ESP sind
 - zur Mustererkennung werden Eventdatenströme gefiltert, aggregiert und korreliert
 - besitzen Zeit und Kausalitätsanforderungen an eine Programmiersystem

- Esper
 - ESP/CEP Engine
 - Leichtgewichtig/integrierbar/open Source
 - Eigene EQL
 - **Ausdruckstark**
 - **Zeitfenster und Zeitabhängigkeiten modellierbar**

- Esper at Codehaus <http://esper.codehaus.org>
- NEsper for .NET
- EsperTech Inc. <http://www.esperotech.com>
 - support, services, and training
- Literatur —David Luckham: The Power of Events
- CEP portal—<http://www.complexevents.com>
- Forum—CEP-Interest@yahoogroups.com



**Vielen Dank für Ihre
Aufmerksamkeit !**



- Annotationsbasiert und Typisiert

```
@EventEngine
public class RFIDQuietTag {
    @When("every LR(zone=1) ->
        (timer:interval(1min) and not LR(zone!=1))")

    public void onEvent(LR assetEvent) {
        //action handler
    }
}
```