EDA – ESP & CEP
... with Java

Complex Event Processing, or CEP, is technology to process events and discover complex patterns among multiple streams of event data. ESP stands for Event Stream Processing and deals with the task of processing multiple streams of event data with the goal of identifying the meaningful events within those streams, and deriving meaningful information from them. The Esper engine has been developed to address the requirements of applications that analyze and react to events.

Session topics

- Historical background
- Real time business
- What is EDA?
  - ESP, CEP
- EDA and Java
- Esper
  - Short case study

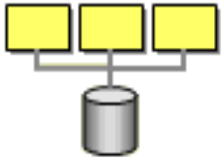
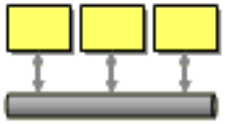
Historical background

- Integration projects as historical background for SOA und EDA

- Typical issues
  - Networks are not reliable
  - Networks are slow
  - Any two applications are different.
  - Changes are inevitable

EAI solutions

- File transfer
- Shared database
- Remote procedure calls
- Messaging

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

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

From the architecture perspective

- SOA
  - Services are RPC calls
    - Point-To-Point communication
  - Increased complexity and manageability
    - Service orchestration
    - Monitoring
- EDA
  - Message based, less coupling

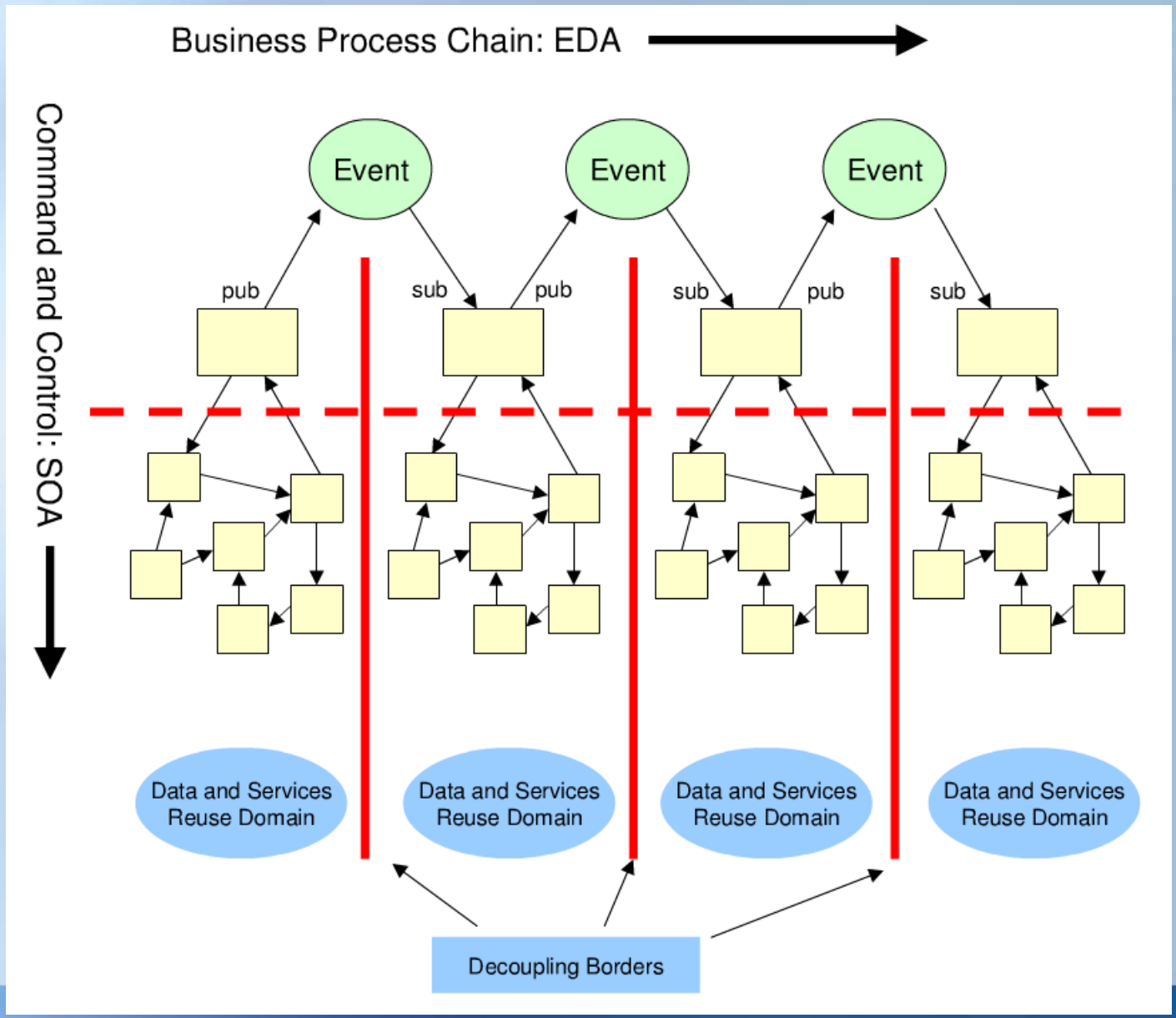
EDA vs. SOA

- EDA: “publish and subscribe”
  - React to (business) events using patterns
  - Usually less coupled than SOA
- SOA: “request and reply”
  - SOA allows to transmit (business) coarse grained Events

EDA extends SOA…

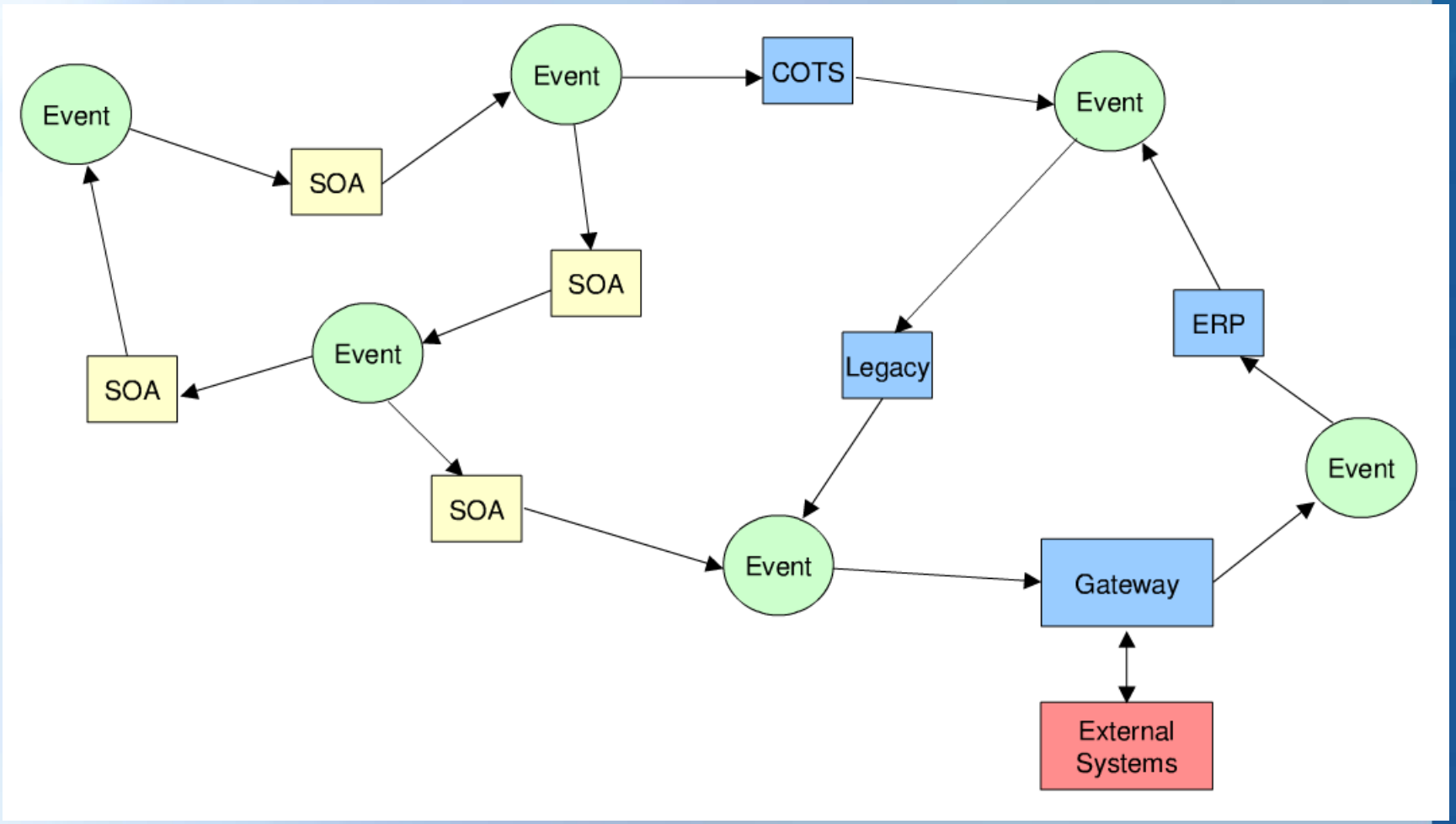
- Use SOA:
  - Vertical interaction, request and reply processes
- Use EDA
  - Horizontal communication between tiers in a process chain

http://soa-eda.blogspot.com/2006/11/how-eda-extends-soa-and-why-it-is.html



http://soa-eda.blogspot.com/2006/11/how-eda-extends-soa-and-why-it-is.html

EDA



http://soa-eda.blogspot.com/2006/11/how-eda-extends-soa-and-why-it-is.html

Session topics

- Historical background
- Real time business
- What is EDA?
  - ESP, CEP
- EDA and Java
- Esper
  - Short case study

Real Time Business

- **Also known as „on demand business“**
- **Real time business (intelligence)** is the process of delivering information about business operations without any latency. In this context, *real time* means delivering information in a range from milliseconds to a few seconds after the business event.

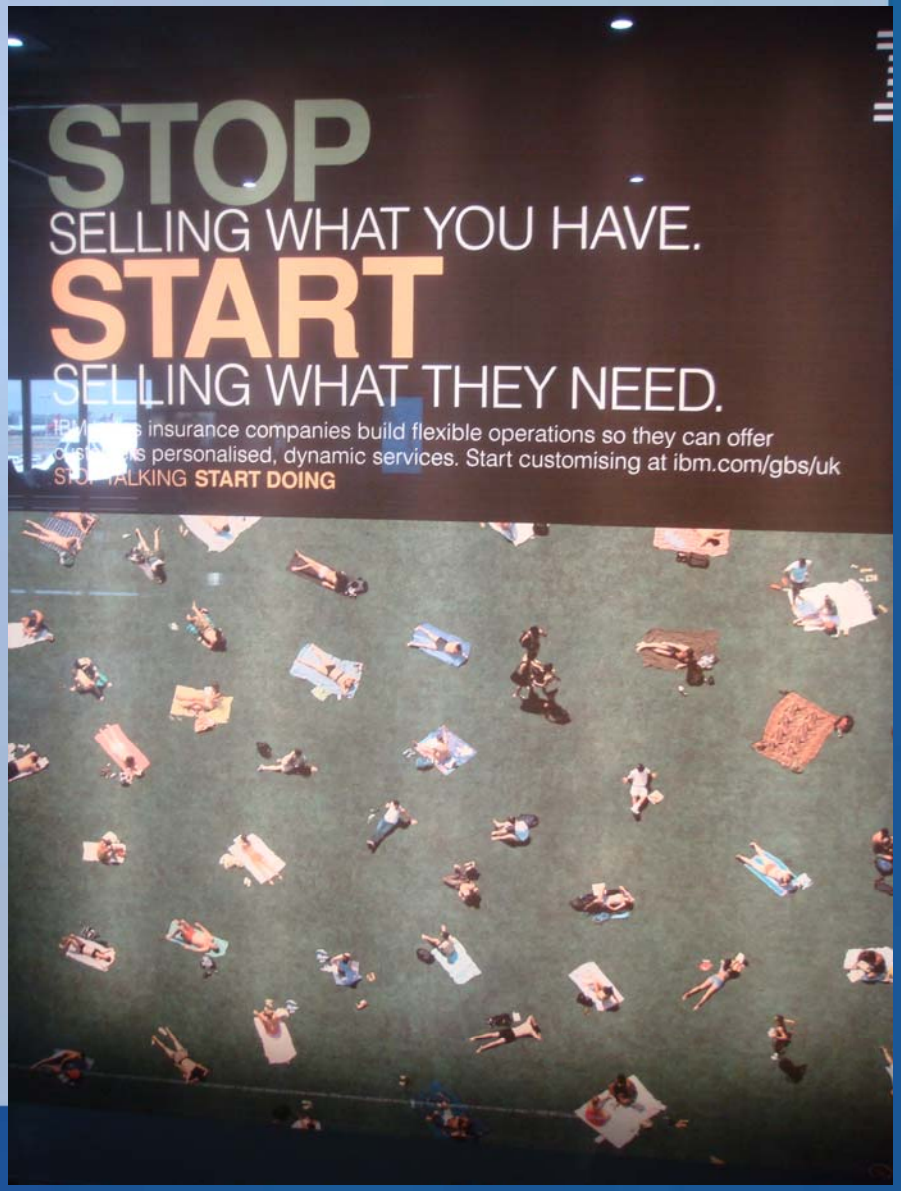
Real time business latency

- All *real time business intelligence* systems have some **latency**, but the goal is to **minimize the time** from the business event happening to a corrective action or notification being initiated
  - Information latency
  - Analysis latency
  - Action latency

Business Inteligence

- traditional business intelligence presents historical information to users for analysis
- real time business intelligence compares current business events with historical patterns to detect problems or opportunities automatically
  - enables corrective actions to be initiated and or business rules to be adjusted to optimize business processes

IBM ad, Heatrow Airport.



Future of Real Time Business

- Unveil, detect business opportunities
  - Faster response to market demands
  - Competitive advantage through early detection / corrections
- Positive trends
  - Increased availability of digital information
    - Fine grained data
    - Frequent updates (state changes)
  - Real time access to (reaction) services

Implementing Real Time Business

- High requirements
  - Huge amount of Events (>> 100.000 events per sec)
  - More than usual OLTP systems can handle
- „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

Actual OLTP benchmarks…

- „Transaction Processing Performance Council“
  - **68,000** TPS (TPC-C)
  - 64 Intel 1.6 GHz processors
  - 128 cores
- Server costs: 11 978 134 US$

Events / messages

- Some state change
  - Response to an authorization request
  - Actual response times at the web server
  - Data from sensors in building
  - Raw market data feeds
- Technical representation of events
  - Key/ value pairs in a HashMap
  - XML document
  - Any plain old Java object: POJO

Session topics

- Historical background
- Real time business
- What is EDA?
  - ESP, CEP
- EDA and Java
- Esper

EDA principles

- Loose coupling
- Event based
- Message queuing infrastructure
- Asynchronous communication
- Event processing models

EDA Requirements

- Event streams
  - Throughput
  - Availability
  - Low latency
  - Correlation between events
- Provide strong modelling language

EDA event processing

- SEP
  - **S**imple **E**vent **P**rocessing
- ESP
  - **S**tream **E**vent **P**rocessing
- CEP
  - **C**omplex **E**vent **P**rocessing

SEP – Simple Event Processing

- Single event based
- Single event triggers reaction
- Commonly adopted
  - *Java Messaging Service*
    - Point to point
    - Publish and Subscribe
  - EAI Patterns
    - Channels, Pipes, routers, etc.

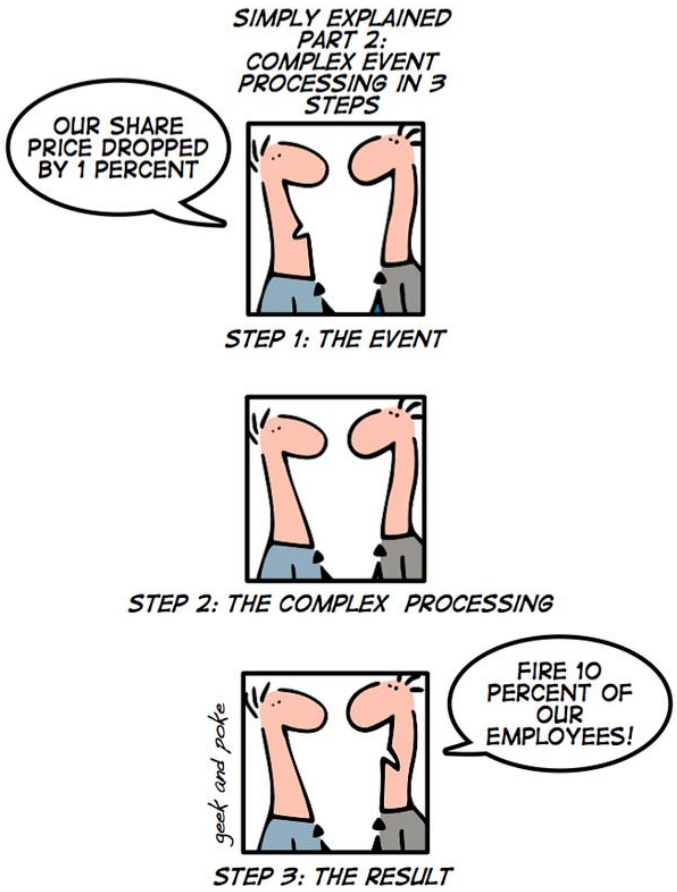
ESP – Event Stream Processing

- Stream based processing
- Single event does not trigger reaction, stream analysis is required
  - Sliding windows
    - Time based
    - Size based

CEP – Complex Event Processing

- Stream based processing
- Complex analysis of events in stream(s) is required
  - Definition of patterns
  - Events correlate
    - In time
    - In causality
    - In space

CEP simply explained



http://soa-eda.blogspot.com/2008/04/cep-simply-explained.html

Complex Event

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

CEP-Example

- Pattern matching events
  - Filter
  - Aggregate
  - Correlate

  - E.g. automated emergency call

    if weight on drivers seat decreases to zero
    in 200 ms and
    with negative accelaration > a and
    with loss of pressure in the tires
    then make emergency call

Session topics

- Historical background
- Real time business
- What is EDA?
  - ESP, CEP
- EDA and Java
- Esper
  - Short case study

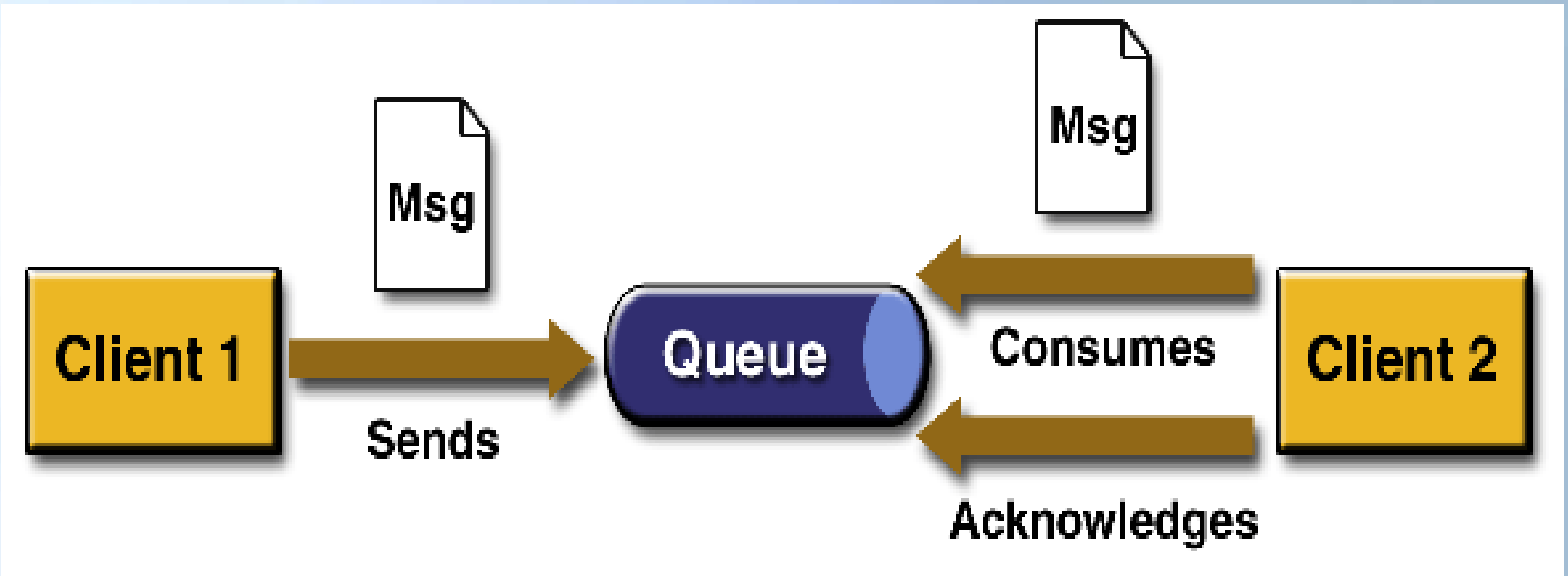
EDA with Java

- SEP
  - Java Messaging Service
  - Enterprise Service Bus
- ESP & CEP
  - No standard
  - Only products

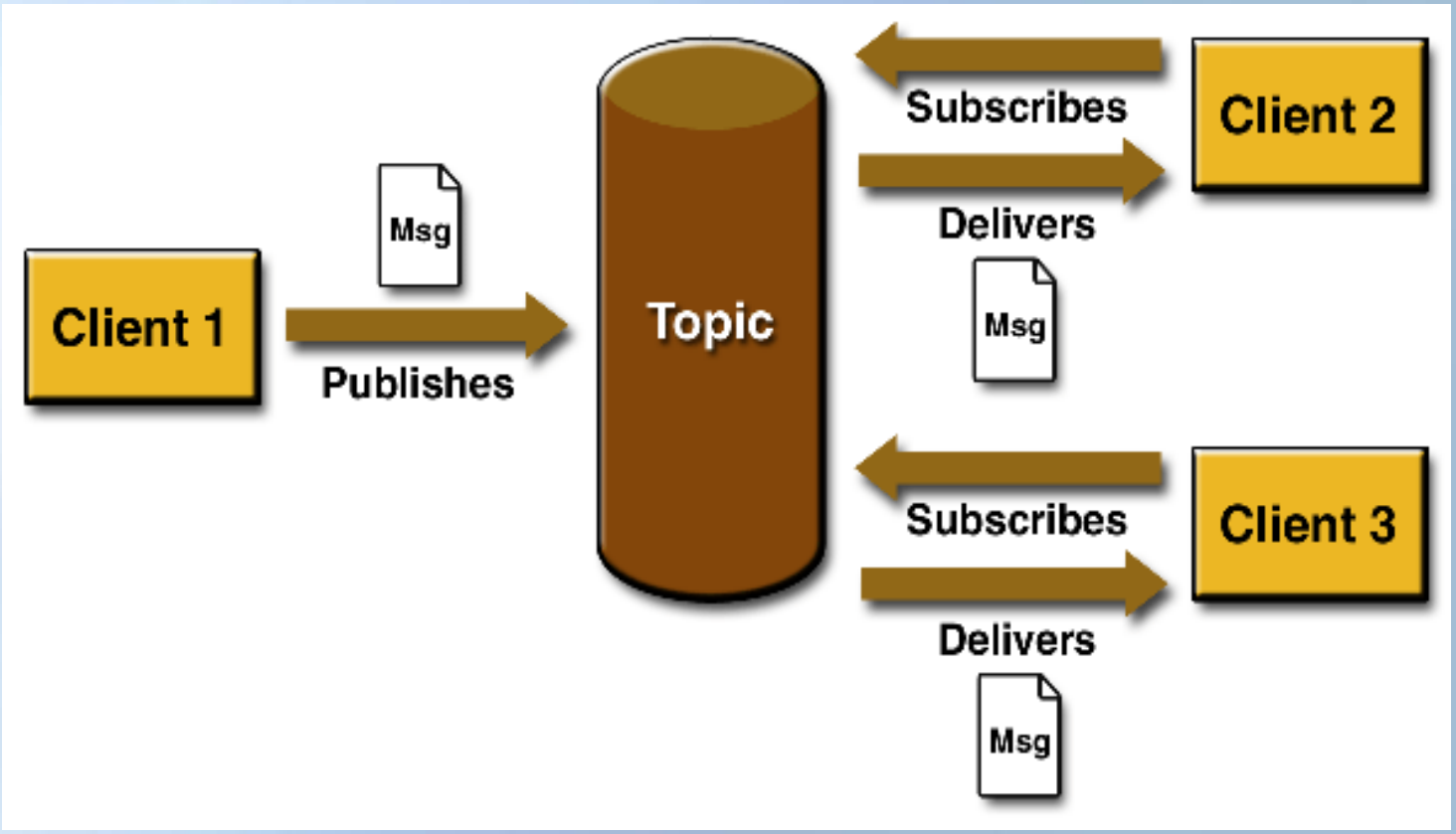
Java Messaging Service

- Well known technology in the Java EE stack
  - Security, transaction
- Many implementations
  - OSS
- JMS specific terms and definitions
- Two event processing models
  - Point-To-Point
  - Publish-and-Subscribe

Point-To-Point

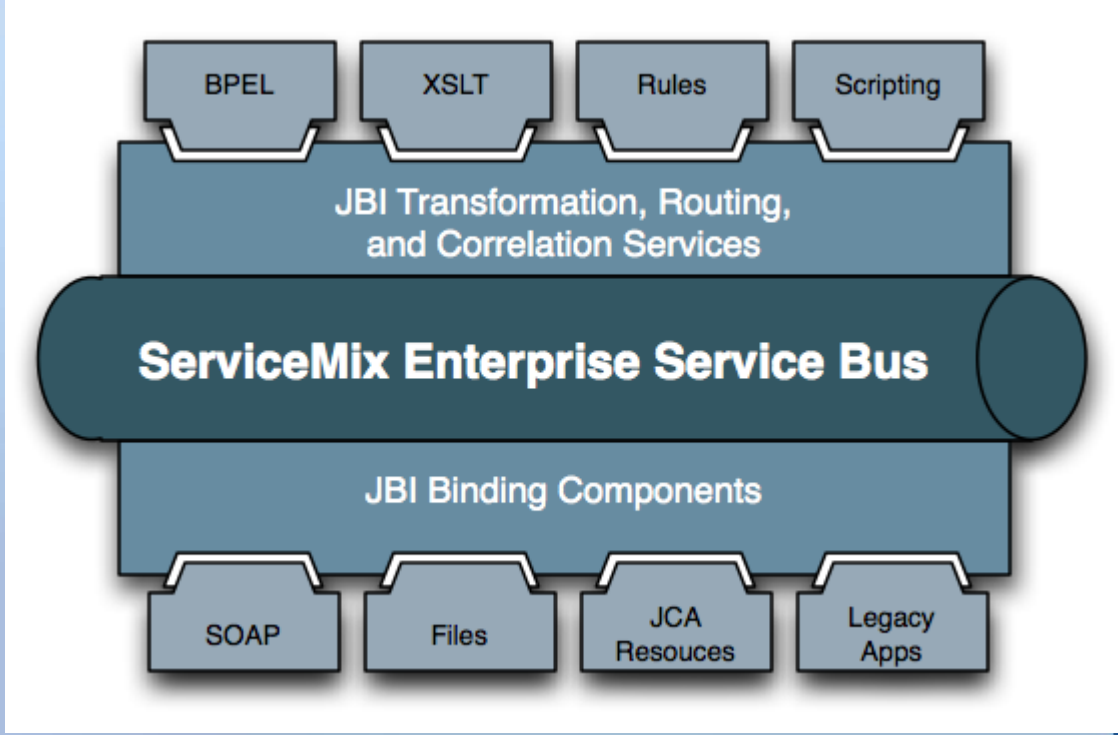


Publish-and-Subscribe



ESB – Enterprise Service Bus

- Apache ServiceMix
- OSS ESB Produkt



http://servicemix.apache.org/home.html

Esper

- Esper is a component for CEP and ESP applications, available for Java as Esper, and for .NET as NEsper.
- Esper and NEsper enable rapid development of applications that process large volumes of incoming messages or events. Esper and NEsper filter and analyze events in various ways, and respond to conditions of interest in real-time.

Projekt Überblick

GPL => problem !?!

- Dual licensing model:
  - Open Source License: GPL 2.0
  - EsperTech
    - Commercial licenses
    - Q2/2007 BEA partner, BEA Event Server

OHLOH

- Summary:
  - Mostly written in Java
  - Small development team
  - Few source code comments

Java	61%
XML	39%
Other	
HTML	<1%
CSS	<1%
DOS batch script	<1%
Perl	<1%
shell script	<1%
SQL	<1%

Filter on: Filter [4 total]

Name	Activity (5 years)	Commits	Person Years	Lines Modified	Comment Ratio
bernhardttom	5 year commit count	306	1.4	395,839	10.6%
srdan	5 year commit count	56	0.2	3,438	12.7%
avasseur	5 year commit count	28	0.4	2,224	12.7%
bwalding	5 year commit count	4	0.0	0	n/a

Good documentation!

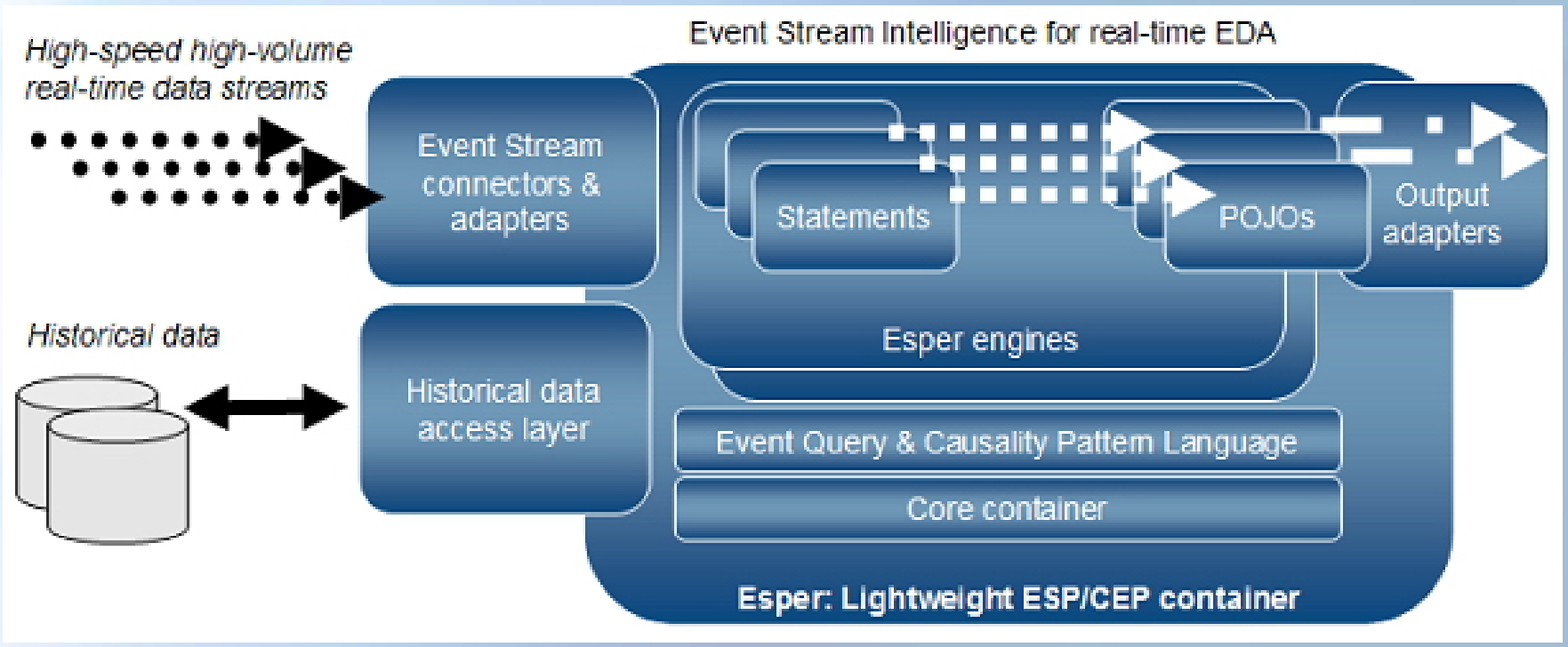
- Impressive reference documentation
  - PDF document has > 130 pages
- Many examples
  - Benchmark kit
  - Many tests
  - Documented sample applications
- Pattern catalogue online

History

Version 2.0.0 Major Release
Released February 17, 2008

Version 1.12.0 Update
Released December 15, 2007
Version 1.11.0 Update
Released September 15, 2007
Version 1.10.0 Update
Released July 18, 2007
Version 1.9.0 Update
Released June 6, 2007
Version 1.8.0 Update
Released April 22, 2007
[…]
Version 0.7.0 (Alpha Release)
Released January 16, 2006

Big picture



Performance

- Esper Benchmark
  - http://esper.codehaus.org/esper-2.0.0/doc/reference/en/html_single/index.html#performance-kit
  - The benchmark application is basically an Esper event server build with Esper that listens to remote clients over TCP. Remote clients send MarketData(ticker, price, volume) streams to the event server.

- Esper exceeds over **500 000 event/s on a dual CPU 2GHz** Intel based hardware, with **engine latency below 3 microseconds** average (below 10us with more than 99% predictability) on a VWAP benchmark with 1000 statements registered in the system - this tops at 70 Mbit/s at 85% CPU usage.
- Esper also demonstrates **linear scalability** from 100 000 to 500 000 event/s on this hardware, with consistent results accross different statements.

EDA alternatives

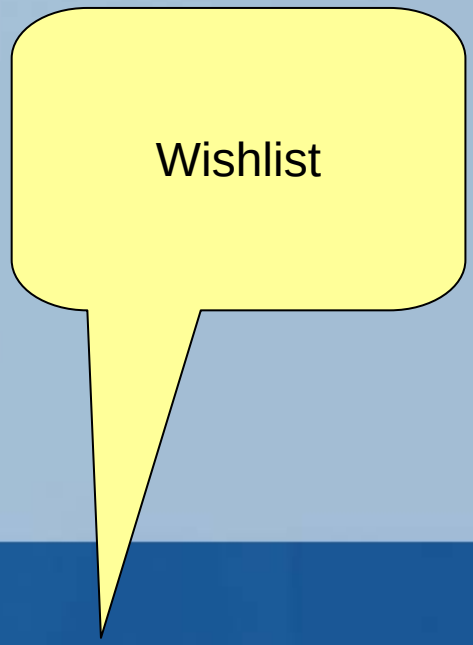
- Database
  - Polling
  - Wasteful use of resources
  - SQL does not support temporal and causality statements
- Rule engines + JMS
  - No optimizations for event streams
  - No continuous processing of events/ streaming
- Distributed Caches bzw. JINI spaces
  - Offer Listener, but nothing similar to EQL

EDA infrastructure

- EXtreme Transaction Processing (XTP)
  - Term introduced 2003 by Gartner
  - Rate above 100 000 Events/s
  - Correlation < 2%
  - Combination of Events and historical data
- Event repositories and ontologies

Event Driven Applicationservers

- Middleware providing
  - Deployment,
  - Runtime and management for EDA applications
  - Open, standards
  - Java/.Net support



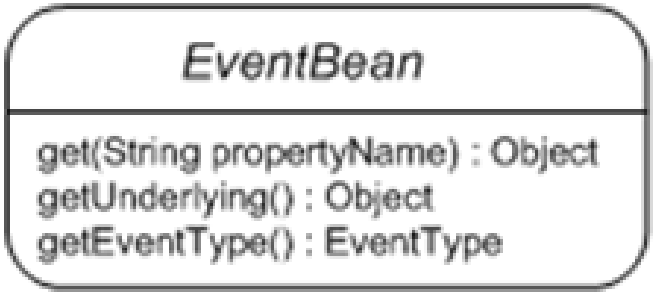
Session topics

- Historical background
- Real time business
- What is EDA?
  - ESP, CEP
- EDA and Java
- Esper
  - Short case study

API Overview

- EPServiceProvider
  - Engine
  - Threads, time, streams
- EPStatement
  - Statements / Queries
  - In EQL (Event Query Language)
- UpdateListener
  - Listener
  - POJI

Events



- Flexible approach
- Esper sends EventBeans

- EventBeans contains
  - POJOs
  - Key value pairs (java.util.Map)
  - XML documents (org.w3c.dom.Node)

Esper Events - POJOS

```
public class EmployeeEvent {
    public String getFirstName();
    public Address getAddress(String type);
    public Employee getSubordinate(int index);
    public Employee[] getAllSubordinates();
}
```

```
every EmployeeEvent(firstName='myName')
every EmployeeEvent(address('home').streetName='Park Avenue')
every EmployeeEvent(subordinate[0].name='anotherName')
every EmployeeEvent(allSubordinates[1].name='thatName')
every EmployeeEvent(subordinate[0].address('home')
          .streetName='Water Street')
```

Esper Events - XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Sensor>
    <ID>urn:epc:1:4.16.36<ID>
        <Observation Command="READ_PALLET_TAGS_ONLY">
            <ID>00000001<ID>
            <Tag> <ID>urn:epc:1:2.24.400<ID> </Tag>
            <Tag> <ID>urn:epc:1:2.24.401<ID> </Tag>
        </Observation>
</Sensor>
```

```
select ID,
    Observation.ID,
    Observation.Command,
    Observation.Tag[0]
from
    SensorEvent.win:time(30 sec)
```

Esper Events – java.util.Map

```
Map event = new HashMap();
event.put("txn", txn);
event.put("account", account);
epRuntime.sendEvent(event, "TxnEvent");
```

```
select account.id, account.rate * txn.amount
    from TxnEvent.win:time(60 sec)
    group by account.id
```

Esper 1x1

```
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);
      }
```

Processing model

- Continuous processing
- Listeners get notified if resultset changes
  - New events come in
  - Old events move out of resultset/ scope
- Database „inside-out“
  - Don’t send queries against data, send the data through the queries

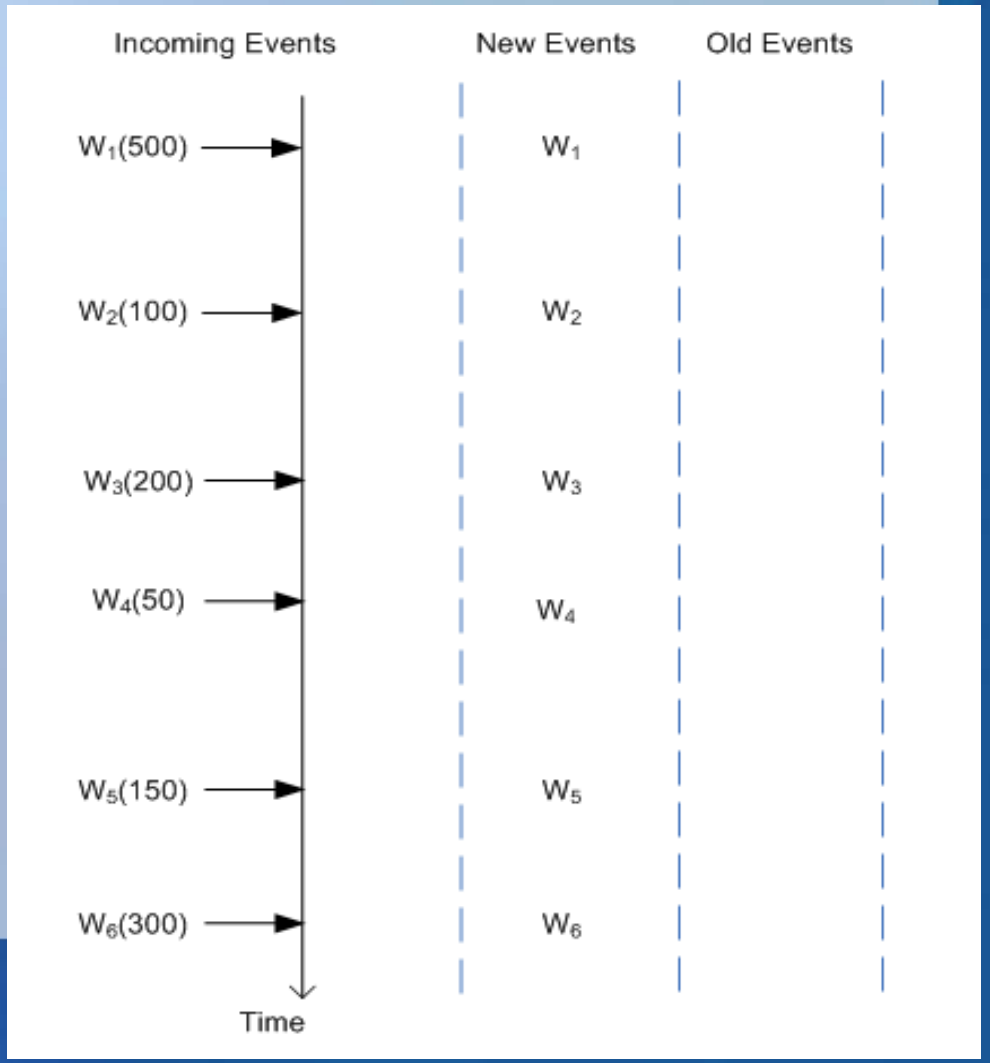
Introducing EQL (EPL)

- SQL analogy
  - Streams: tables
  - Event: record
  - Event attributes: fields in a record
- Esper queries:
  - ESP Queries
  - CEP Queries

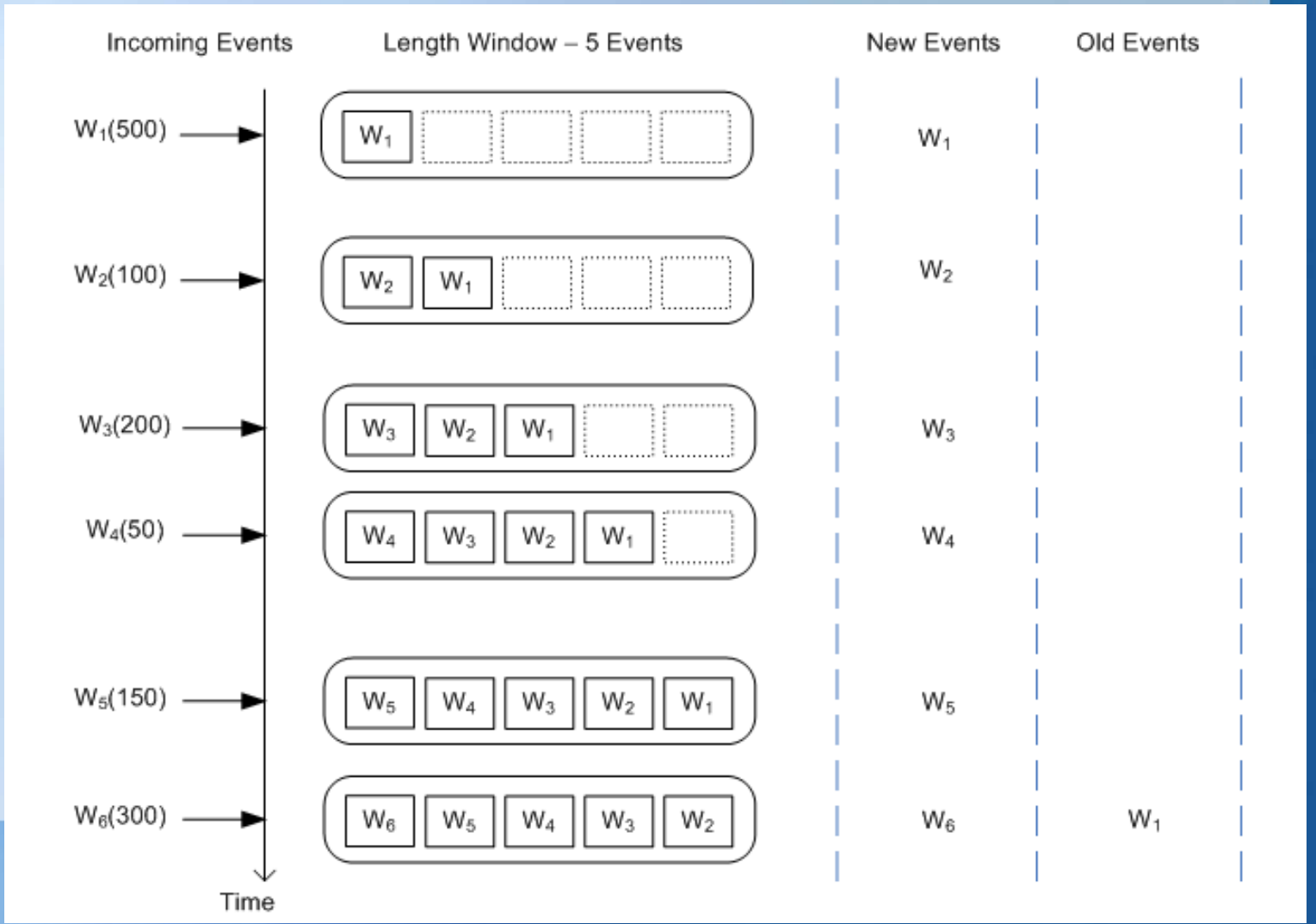
ESP Queries

- Single events
- Events in a sliding time window
- Events in a sliding sized window

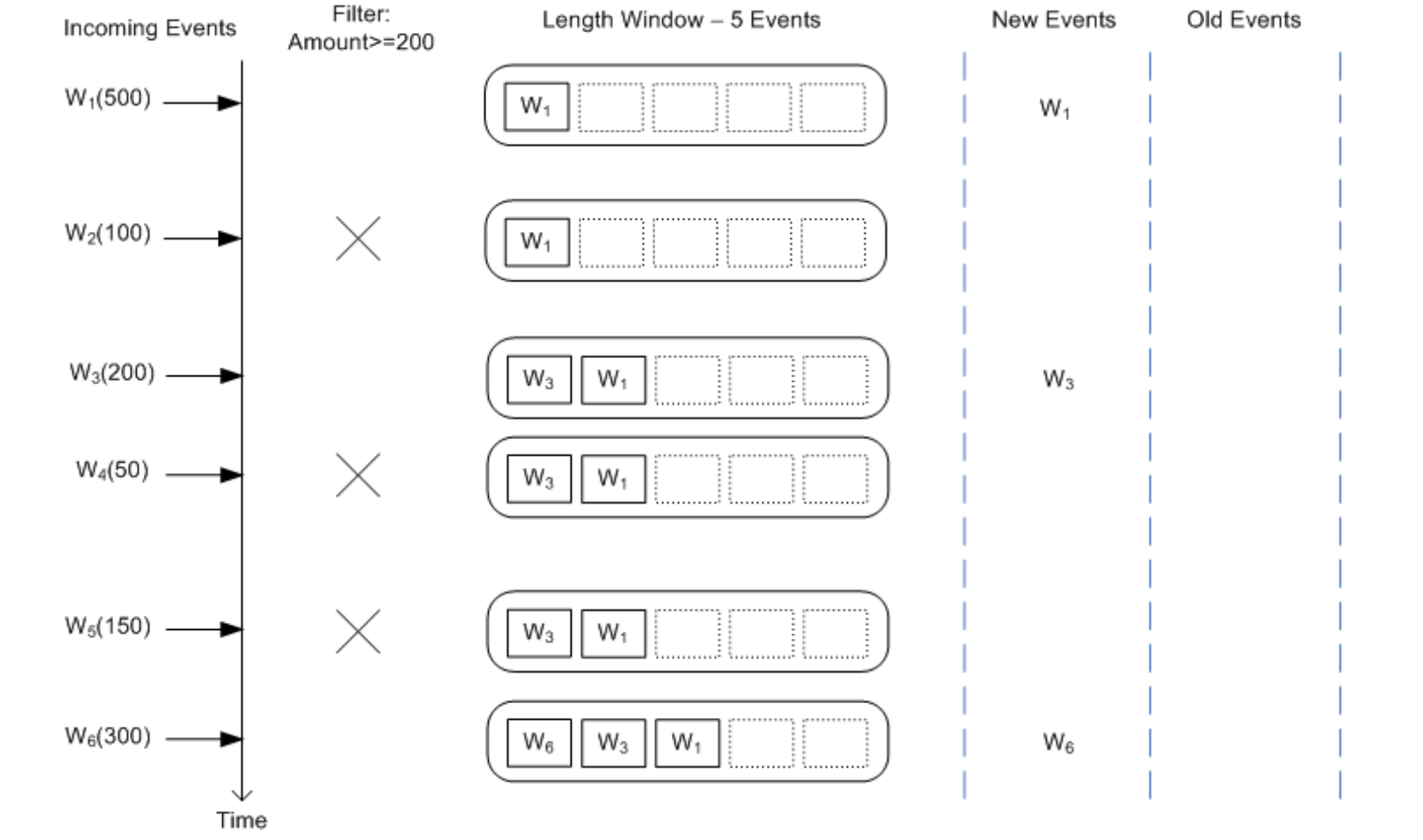
select * from Withdrawal



select * from Withdrawal.win:length(5)



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



Insert into…

```
insert into
   WithdrawalFiltered
        select * from Withdrawal
        where Math.ceil(amount) >= 200

select * from WithdrawalFiltered
```

Batch Modus

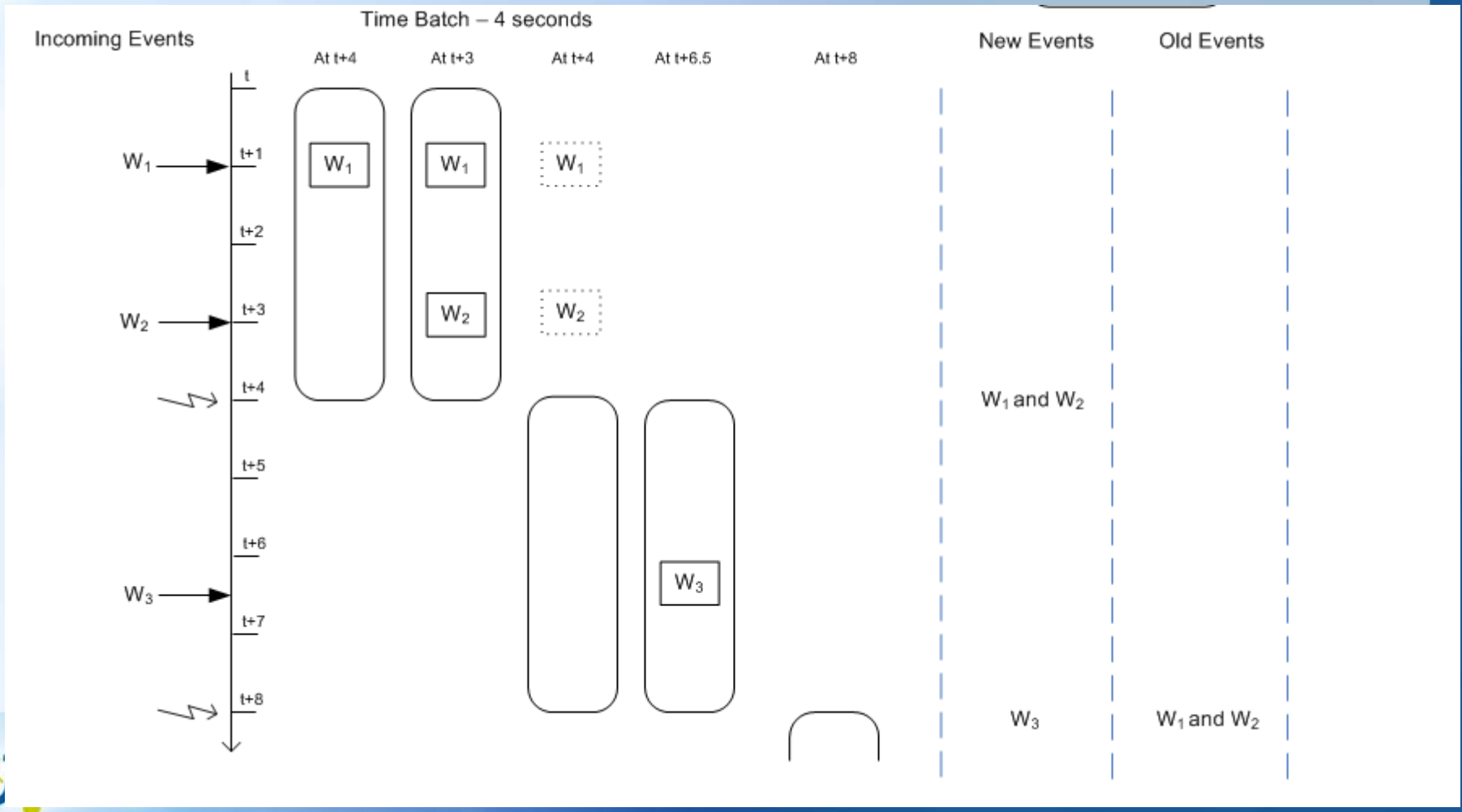
- Accumulate events before updating listener
  - **Accumulate in time**

    select * from Withdrawal.win:time_batch(4 sec)

  - **Accumulate give number of events**

    select * from Withdrawal.win:length_batch(5)

select * from Withdrawal.win:time_batch(4 sec)



CEP queries

- Define patterns to be matched against streams of events
- Used to identify complex events
- Pattern keyword

Pattern overview

- Define iteration/ repetition using „every“
- Logical operators (and, or, not)
- Followed by operator “->”
- 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

Simple sample patterns #1

- Events:
  - 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}

Simple sample patterns #2

- Events:
  - 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}

Guards und Observer

- (A or B) where timer:within (5 sec)
  - One A oder B in the next 5 seconds
- (every A) where timer:within(10 sec)
  - All A Events in the next 10 seconds
- A -> timer:interval(10 seconds)
  - After A, wait of 10 seconds
- every timer:at(5, *, *, *, *)
  - Every 5 minutes

Simple temperature monitoring?

- A temperature sensor delivers events
  - Sample
    - sensor
    - temp
- Please tell me when it gets too hot,
  - We need to trigger an alarm, if…
    - In 90 seconds
    - 3 consequent events
    - Temperature always > 50
    - Events originate from the same sensor

Complex Event Processing…

```
every sample= Sample(temp > 50)
    -> (
          (
         Sample(sensor=sample.sensor, temp > 50)
          and not
         Sample(sensor=sample.sensor, temp <= 50)
        )
         ->
          (
            Sample(sensor=sample.sensor, temp > 50)
          and not
            Sample(sensor=sample.sensor, temp <= 50)
        )
       ) where timer:within(90 seconds)
```

Session topics

- Historical background
- Real time business
- What is EDA?
  - ESP, CEP
- EDA and Java
- Esper
  - Short case study

Short case study

- **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.

http://esper.codehaus.org/tutorials/tutorial/feedmonitor_casestudy.html

Input

- 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

The future of EDA?

„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).“

Gartner RAS Core Research Note G00146107 Q1/2007

Questions?

Thank you!