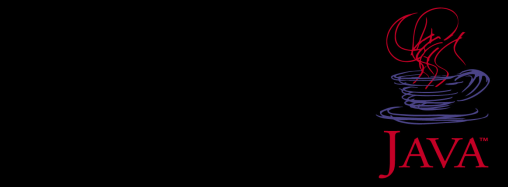
JAVA DAYS 98

JavaSpaces - neue Wege bei der Erstellung verteilter Anwendungen

Andreas Dörr
Java Technologe
Sun Microsystems GmbH

Sun microsystems

JavaSpaces - na und?



- Workflow Systeme
- Customer Management Systeme
- Lieferketten Management
- Handelssysteme
- Agentensysteme
- Publish und Subscribe Dienste

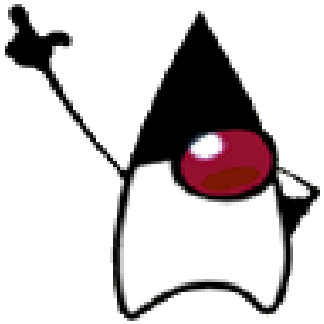
JavaSpaces - Ziele

- Einfach, einfach, einfach ...
- Viel Funktionalität, wenig Code
- Einheitlicher Mechanismus
- Objekte
- Asynchron
- Erweiterbar
- Transparent
- 100% Pure Java

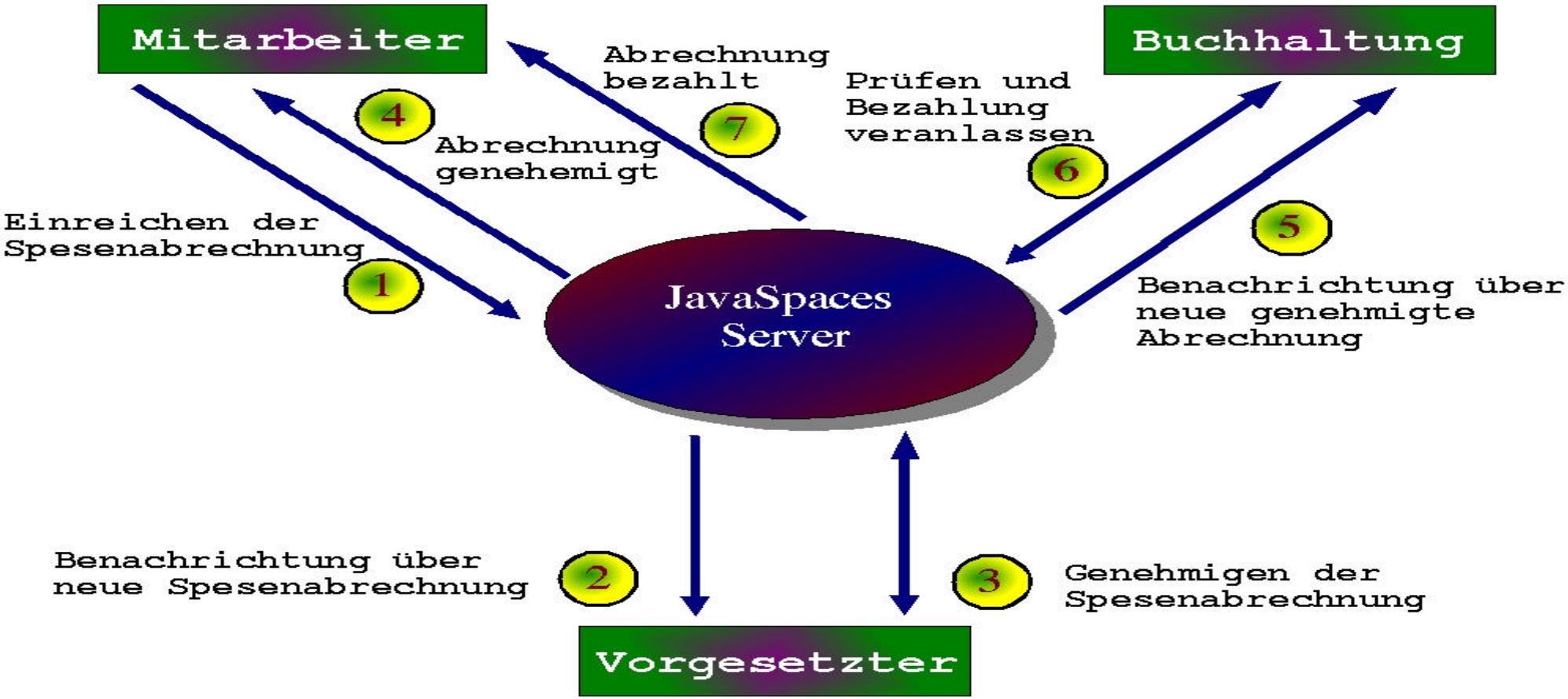
Sun microsystems

JavaSpaces - das Modell

- Entries
- Templates
- Operationen
- Transaktionen



JavaSpaces - ein Beispiel



JavaSpaces - Entry

- Implementiert das Interface Entry oder AbstractEntry.
- Nur public Felder werden beachtet.
- Felder müssen Referenzen auf Objekte sein, keine primitiven Datentypen (Integer statt int).
- Die Referenzen müssen auf serialisierbare Objekte verweisen.
- Public no-arg Konstruktor muß vorhanden sein.

JavaSpaces - Entry Beispiel


```
class ExpenseReport implements Entry {
  public String m_LastName, m_FirstName;
  public String m_Department, m_Status;
  public Float  m_Total = new Float("0.0");
  public Vector m_Items = new Vector();

  private class Item {
    private String m_Type;
    private Float  m_Amount;

    public Item(String type, Float amount) {
      m_Type = type; m_Amount = amount;
    }
    public String getType() { return m_Type; }
    public Float getAmount() { return m_Amount; }
    public void setAmount(Float amount) { m_Amount = amount; }
  }

  public ExpenseReport() { /* EMPTY */}

  public void addItem(String type, Float amount) {
    m_Items.add(new Item(tpye,amount));
  }
}
```

JavaSpaces - Template

- Ein Template ist ein Entry mit *Werten* oder *Platzhaltern* (Wildcards).
- Felder mit Werten müssen *exakt* übereinstimmen.
- Felder mit Platzhaltern werden ignoriert.

```
ExpenseReport expRep = new ExpenseReport();
expRep.m_FirstName = expRep.m_LastName = null;
expRep.m_Department = "Java Zentrum";
expRep.m_Status     = "submitted";
```

JavaSpaces - Operationen

- write - schreibt einen Entry in den JavaSpace.
- read - liefert einen gesuchten Entry. Der Entry wird nicht aus dem JavaSpace entfernt.
- take - liefert einen gesuchten Entry. Der Entry wird aus dem JavaSpace entfernt.
- notify - Nachricht über einen neuen Entry im JavaSpace.
- Für alle Operationen gilt: Ein bestimmter Typ von Entry wird über *Templates* qualifiziert.

JavaSpaces - Transaktionen

- Gruppieren von mehreren Operationen, die atomar ausgeführt werden.
- Transaktionen können mehrere JavaSpaces umfassen.
- Transaktionen bestimmen die Sichtbarkeit von Entries.
- null - Transaktion: Transaktion mit genau einer Operation und automatischem Festschreiben (Auto-Commit).

Sun microsystems

JavaSpaces - write


```
Lease write(Entry e, Transaction txn, long lease)
   throws RemoteException, TransactionException
```

```
Transaction txn        = null;
long        timeToLive = Lease.FOREVER;
Lease       expire     = null;

try {
   expire = getSpace().write(expRep, txn, timeToLive);
} catch (Exception e) {
   e.printStackTrace(System.err);
}
```

JavaSpaces - read, take


```
Entry read(Entry tmpl, Transaction txn, long timeout)
Entry take(Entry tmpl, Transaction txn, long timeout)
   throws TransactionException, UnusableEntryException,
          RemoteException, InterruptedException
```

```
ExpenseReport tmpl = new ExpenseReport();
tmpl.m_FirstName = tmpl.m_LastName = null;
tmpl.m_Department = "Java Zentrum";
tmpl.m_Status     = "submitted";

long        timeToWait = 0L;
Transaction   txn       = null;
ExpenseReport res       = null;

try {
   res = (ExpenseReport)getSpace().read(tmpl, txn, timeToWait);
} catch (Exception e) {
   e.printStackTrace(System.err);
}
```


JavaSpaces - notify

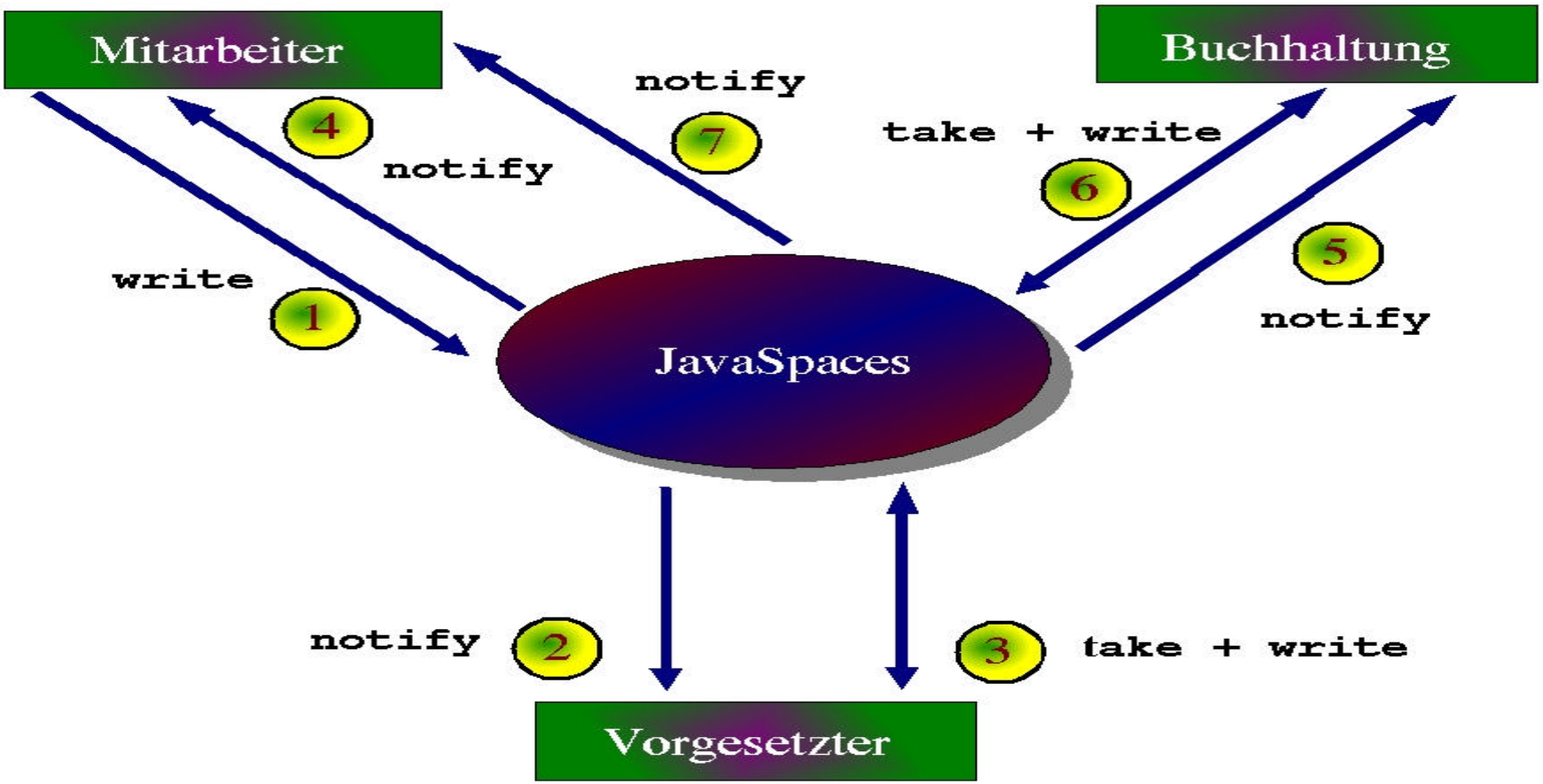

```
EventRegistration notify(Entry tmpl, Transaction txn,
                         RemoteEventListener listener,
                         long lease, MarshalledObject hndb)
  throws RemoteException, TransactionException
```

```
ExpenseReport tmpl = new ExpenseReport();
tmpl.m_FirstName = tmpl.m_LastName = null;
tmpl.m_Department = "Java Zentrum";
tmpl.m_Status     = "submitted";

long              timeout = Lease.FOREVER;
Transaction       txn     = null;
EventRegistration  reg    = null;

try {
   reg = getSpace().notify(tmpl, txn, this, timeout, null);
} catch (Exception e) {
   e.printStackTrace(System.err);
}
```

JavaSpaces - wo sind sie?

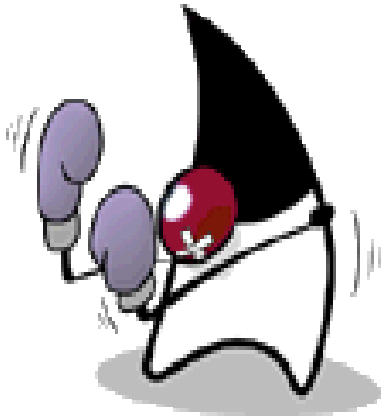

```
JavaSpace space      = null;
Locator   locator    = null;
Finder    finder     = null;
String    locatorName = "net.jini.impl.outrigger.RegistryLocator";
String    finderName  = "net.jini.impl.outrigger.RegistryFinder";

try {
   locator = (Locator)(Class.forName(locatorName)).newInstance();
   finder  = (Finder)(Class.forName(finderName)).newInstance();

   space   = (JavaSpace)finder.find(locator, "SpesenSpace");
} catch (Exception e) {
   e.printStackTrace(System.err);
}
```

JavaSpaces - ein Beispiel (2)



JavaSpaces - warum?

- Für ein RPC basiertes System muß man:
  - Protokolle und Schnittstellen entwerfen
  - Client und Server implementieren
- Mit JavaSpaces muß man:
  - Abläufe und Einträge entwerfen
  - Den Client implementieren
  - (der Server ist schon fertig!)
- Lose gekoppelte Systeme skalieren gut!
- Haben Sie etwas gegen Einfachheit?



JavaSpaces - Weisheiten ...

... every key step in software history has been a step away from the computer, toward *forgetting* about the machine and its physical structure and limitations ...

David Gelernter

JavaSpaces - The End

Vielen Dank für Ihre Aufmerksamkeit !!