SBB CFF FFS

Hazelcast bei der SBB
24.Mai 2016 - jug.ch

Korhan Gülseven
kgu@sbb.ch

Hazelcast ist

→ Ein Produkt

- In-Memory Data Grid in Java
- Open Source (Apache License 2.0) https://github.com/hazelcast
- 6 MB Jar ohne Dependencies
- Kommerzielle Versionen & Subscription Plans: Enterprise & HD

→ Eine Firma

- Gründung 2008
- Von Istanbul nach Palo Alto (HQ)
- Ca. 80 Angestellte
- Büros in Istanbul, Ankara, London, New York, Indien

Hazelcast ist

Mission Statement:

«Our main goal is to simplify development of scalable and highly available systems»

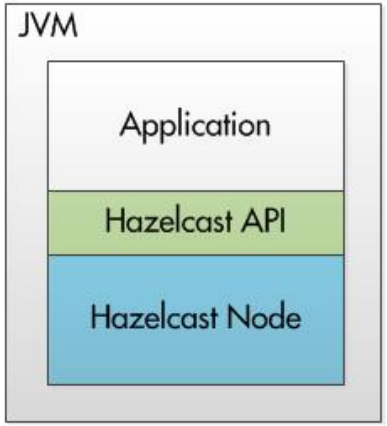
→ Features

- Distributed Data-Grid
- Distributed Computing
- Messaging

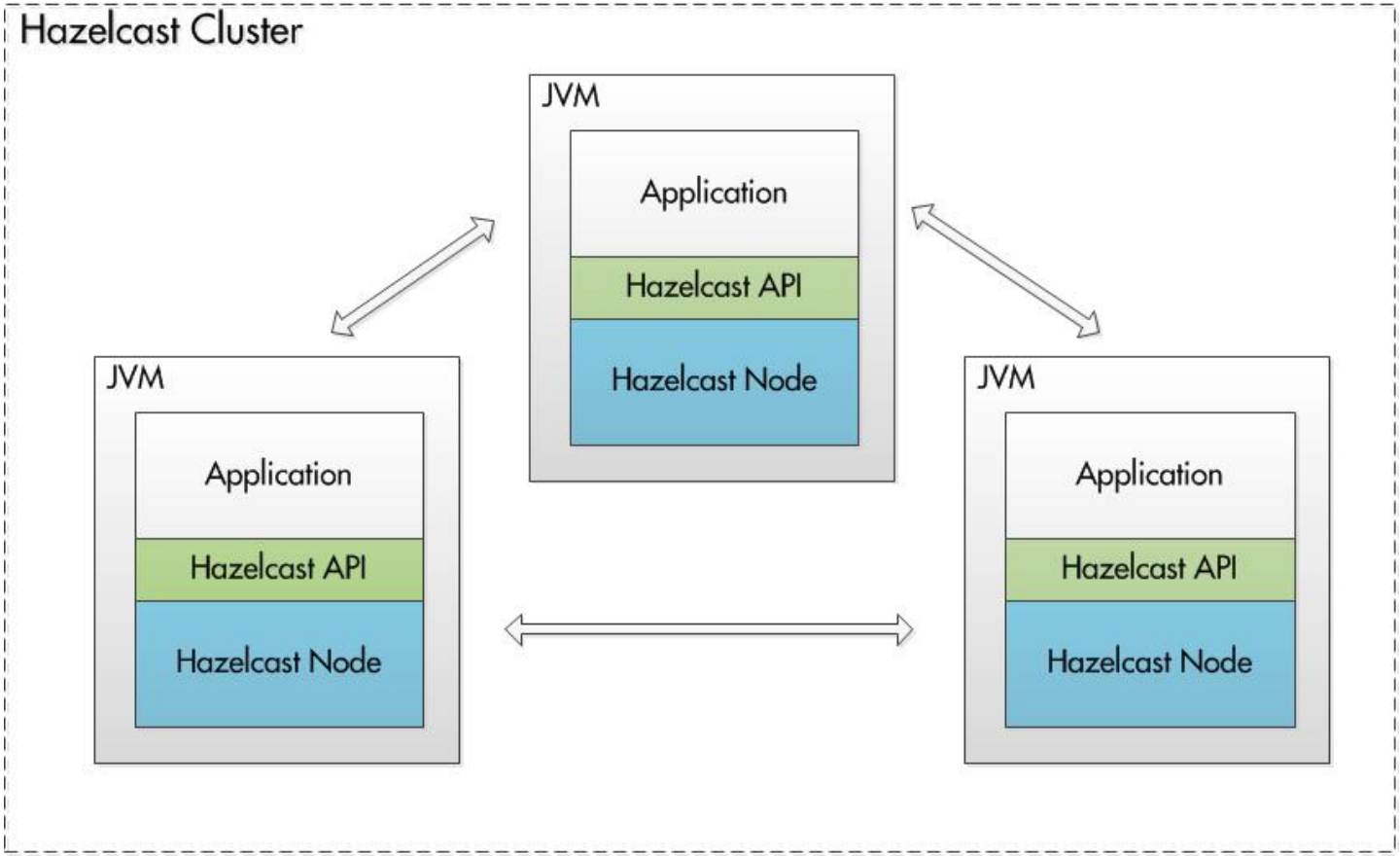
→ Aus Entwicklersicht

- Einfach, vielseitig, schnell
- API vom Feinsten

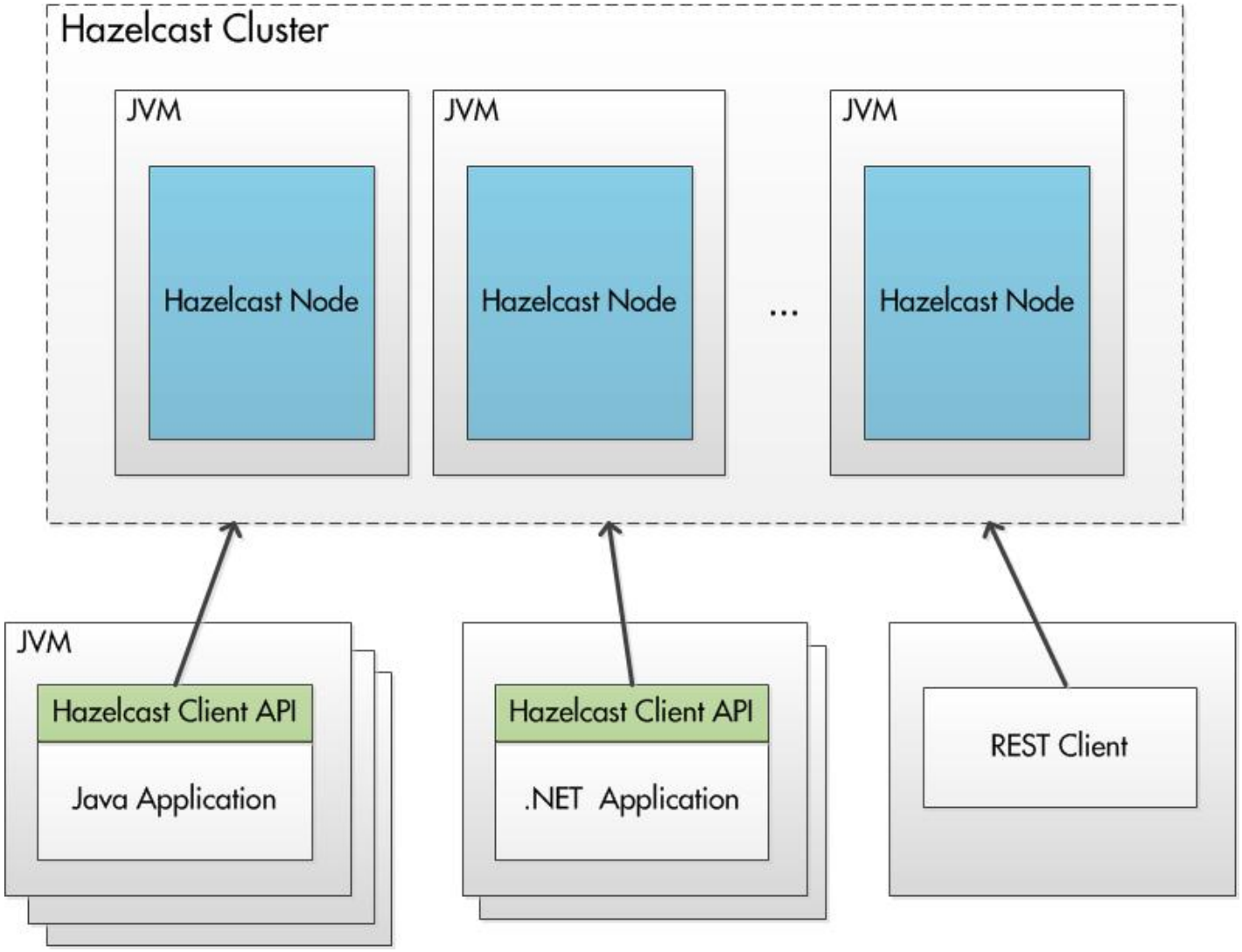
Topologie: Embedded



Topologie: Embedded



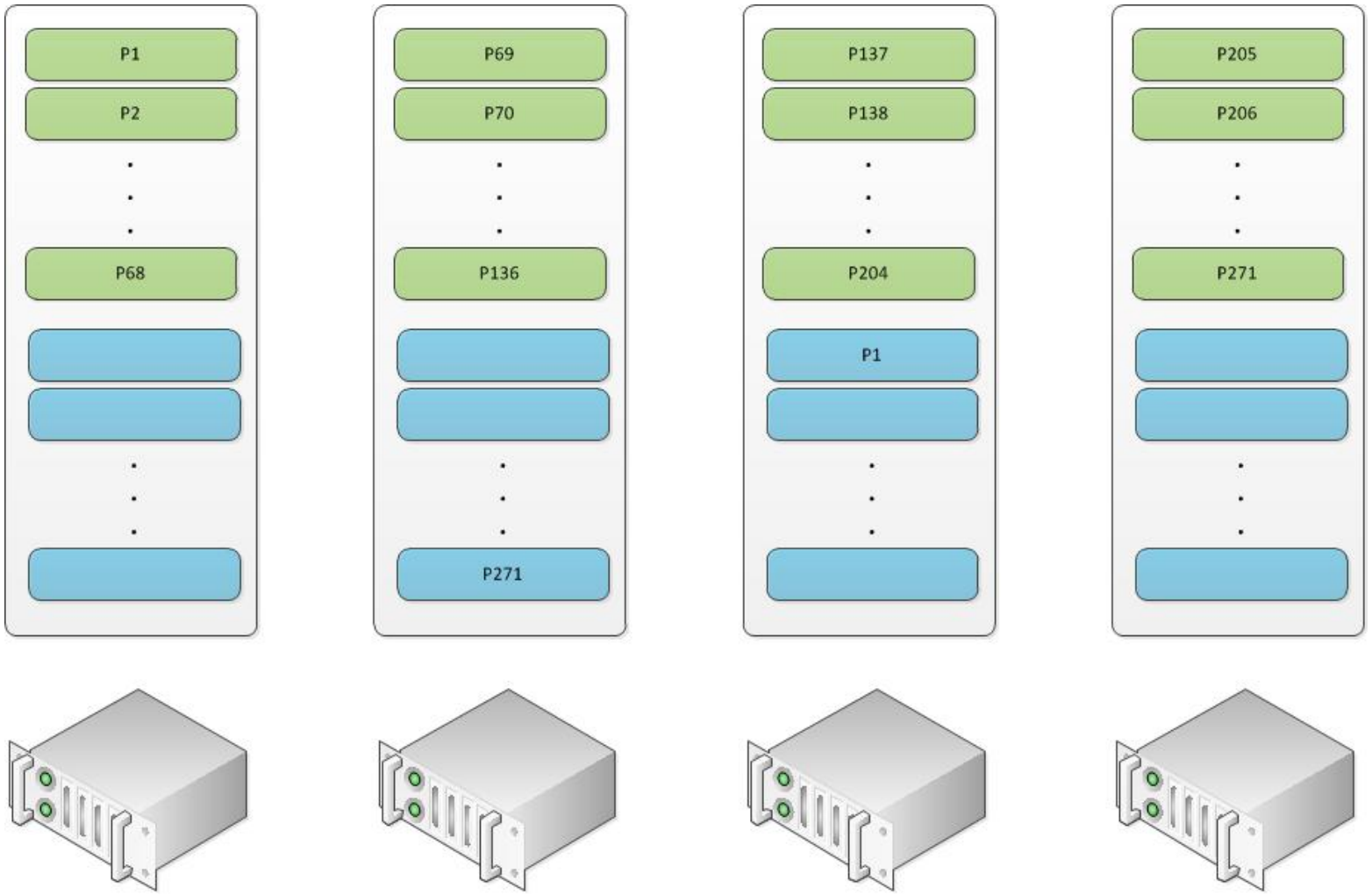
Topologie: Client/Server



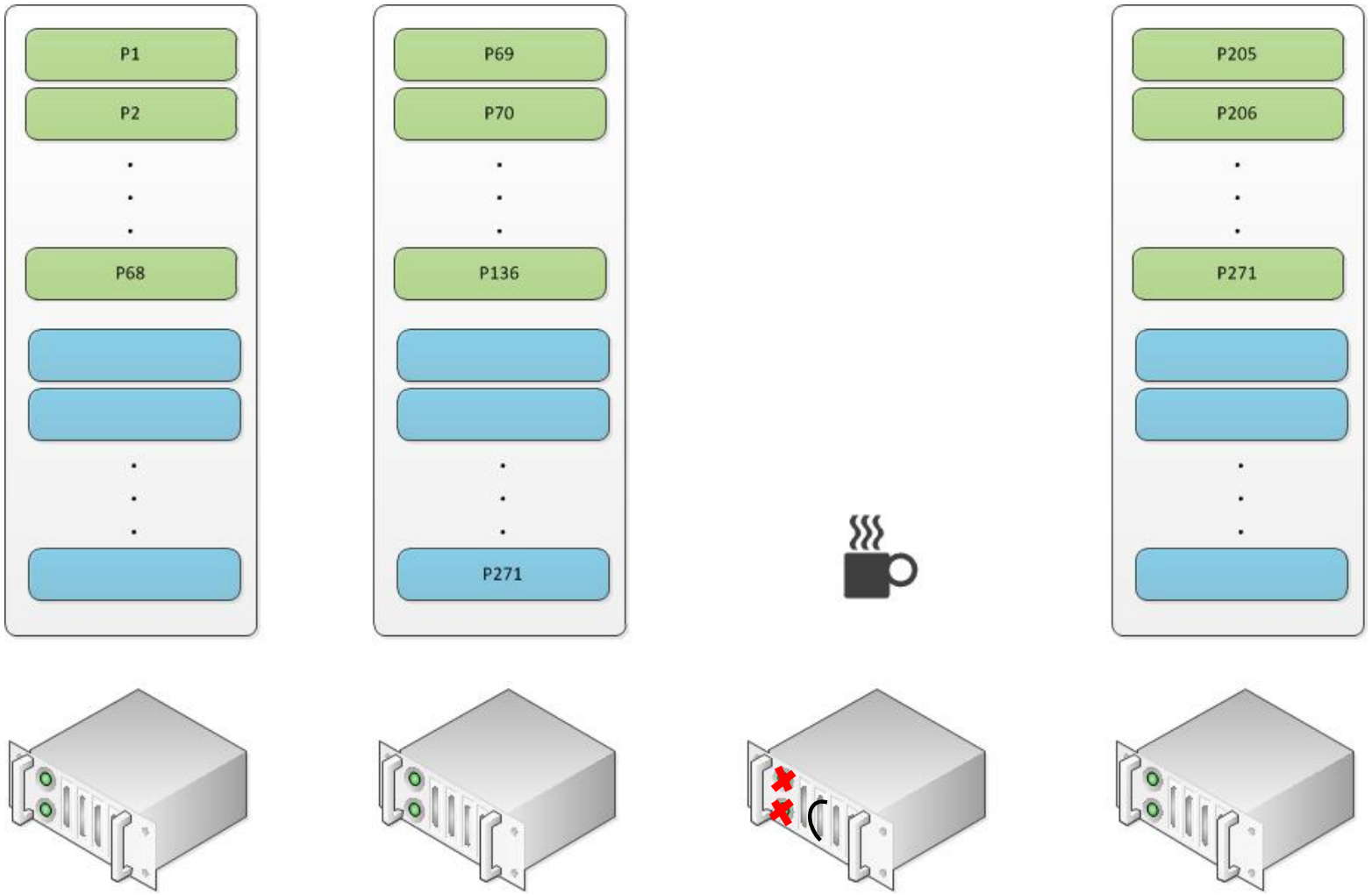
Partitionierung

- Hazelcast speichert die Daten in Partitionen
- Die Daten werden in Partitionen gleichmässig verteilt
- Bestimmung Partition: hash(key-data) % partition-count
- Replicas dienen der Ausfallsicherheit

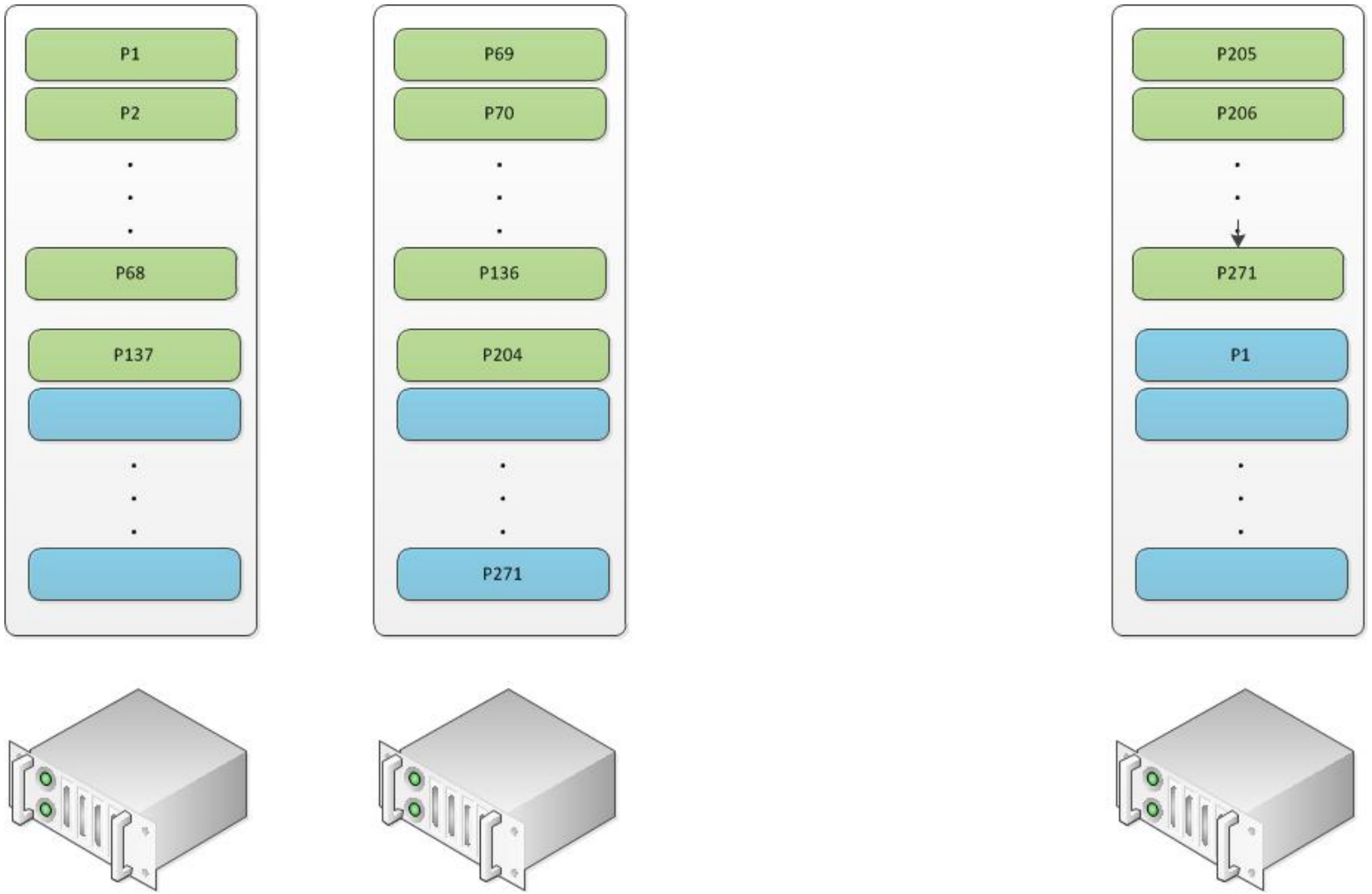
Partitionierung



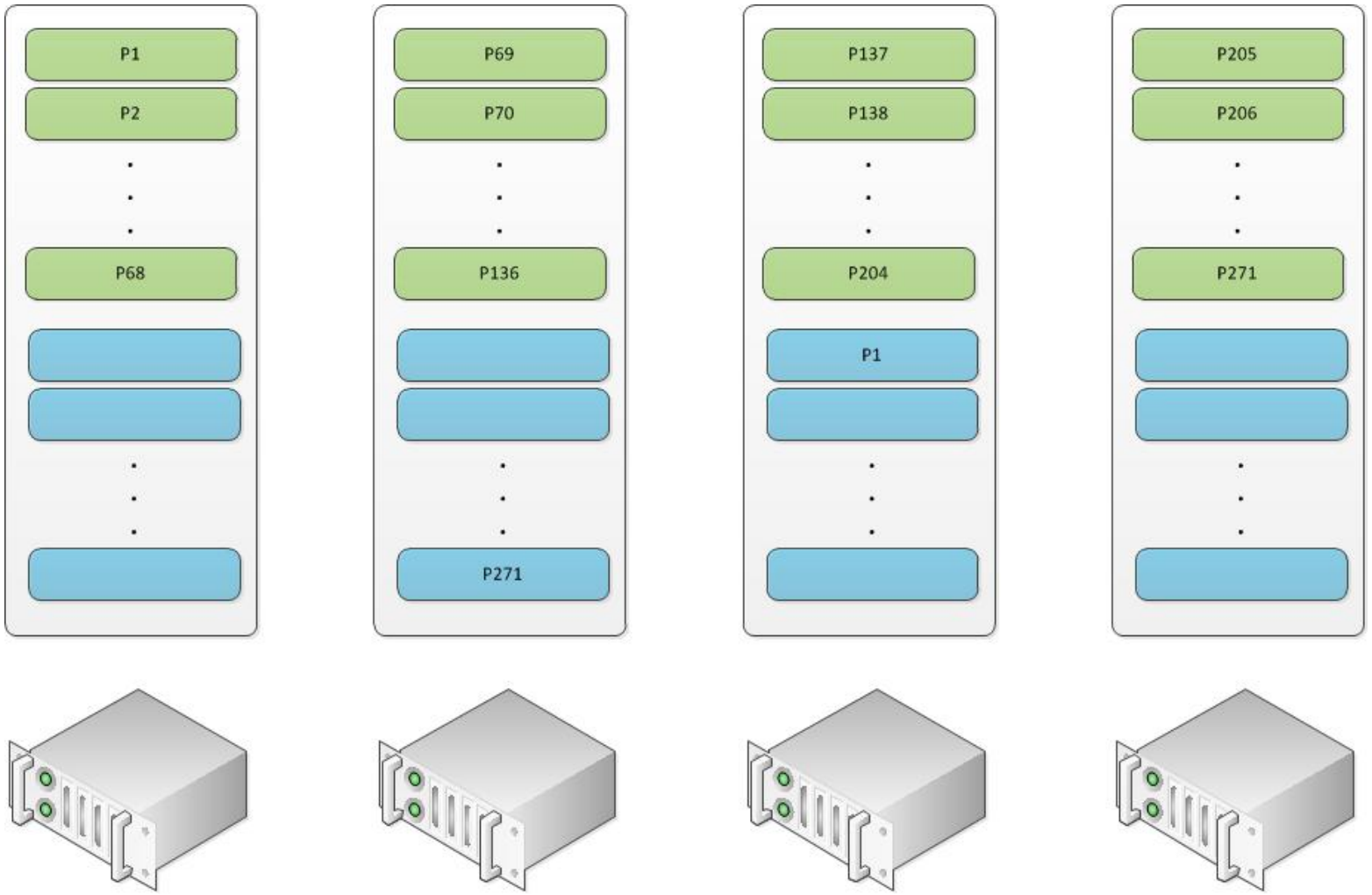
Partitionierung



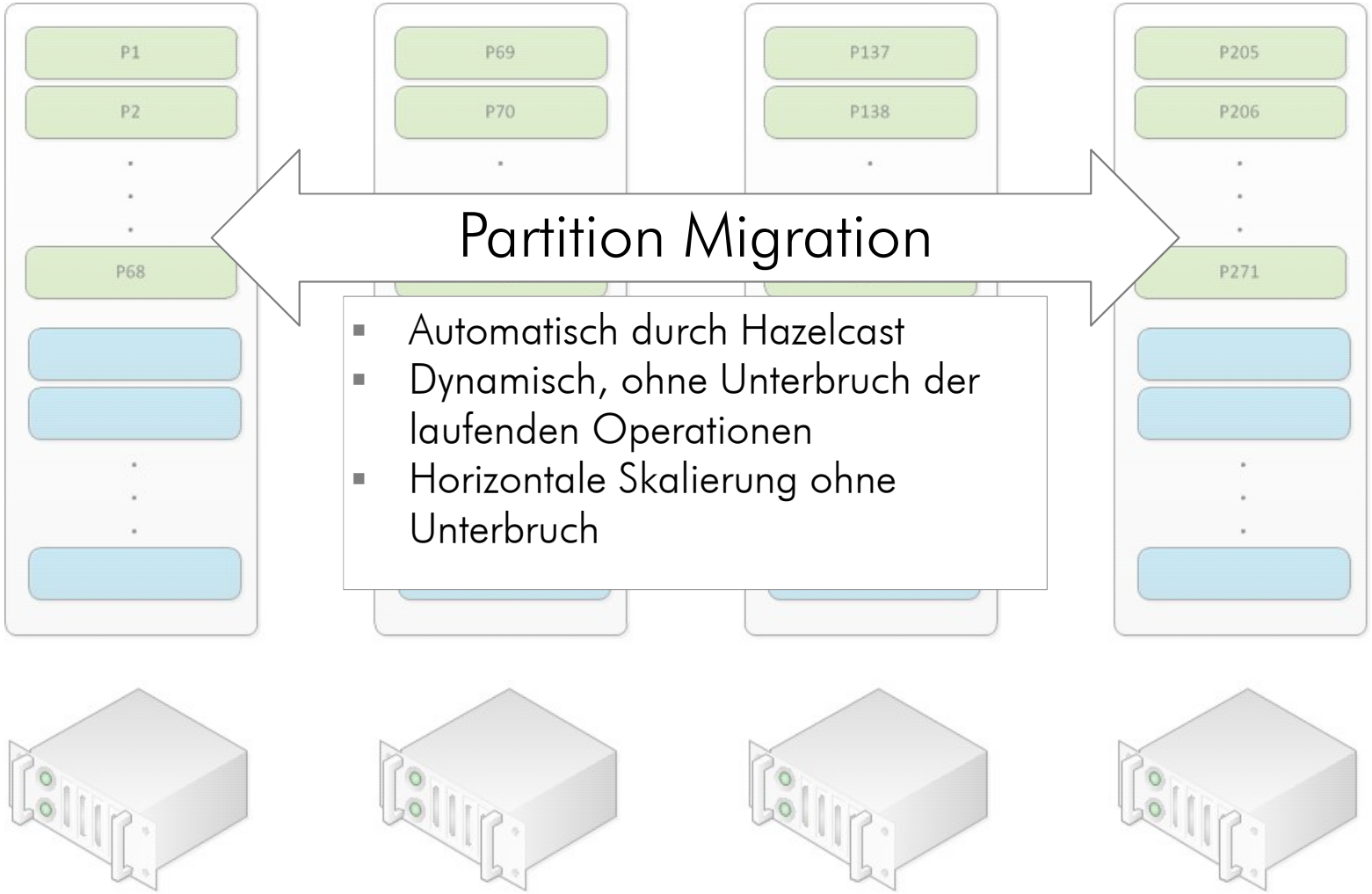
Partitionierung



Partitionierung



Partitionierung



Serialisierung

- Serialisierung ist wichtig
- In-Memory Format
  - Binary
  - Object
- Default: Serializable (jdk), Externalizable (jdk), DataSerializable IdentifiedDataSerializable, Portable
- Custom: StreamSerializer, ByteArraySerializer (pluggable z.B. Kyro)

Serialisierung

```
20 public class ZugfahrtKey implements Portable {
21
22   public static final String FIELD_TRASSEN_ID = "trassenId";
23
24   private String trassenId;
25
26   @Override
27   public int getClassId() {
28     return 1;
29   }
30
31   @Override
32   public int getFactoryId() {
33     return 1;
34   }
35
36   @Override
37   public void readPortable(PortableReader reader) throws IOException {
38     trassenId = reader.readUTF(FIELD_TRASSEN_ID);
39   }
40
41   @Override
42   public void writePortable(PortableWriter writer) throws IOException {
43     writer.writeUTF(FIELD_TRASSEN_ID, trassenId);
44   }
45 }
```

Data Affinity

```
14 public class ZugfahrtPunkt implements Portable, PartitionAware<ZugfahrtKey> {
15
16   private ZugfahrtPunktKey key;
17
18   public ZugfahrtPunktKey getKey() {
19     return key;
20   }
21
22   @Override
23   public ZugfahrtKey getPartitionKey() {
24     return key.getZugfahrtKey();
25   }
26
```

Distributed Data Structures

- Standard
  - Map, Multimap, Replicated Map
  - Set, List, Queue, Ringbuffer
- Messaging (pub/sub)
  - Topic
- Concurrency Utilities
  - Lock, Semaphore, IdGenerator, AtomicLong, CountDownLatch
- Persistence APIs für Maps und Queues
  - Optional
  - read-through, write-through (sync), write-behind (async)

Distributed Map

```
13    HazelcastInstance client = HazelcastClient.newHazelcastClient();
14    IMap<ZugfahrtKey, Zugfahrt> map = client.getMap("zugfahrtMap");
15
16    ZugfahrtKey key = zugfahrt.getKey();
17
18    map.lock(key);
19
20    try {
21      Zugfahrt existing = map.get(key);
22      if (zugfahrt.getVersion() > existing.getVersion()) {
23        map.set(key, zugfahrt);
24      }
25    } finally {
26      map.unlock(zugfahrt.getKey());
27    }
```

Distributed Query

- Criteria API

```
16    Predicate zugnummerPredicate = Predicates.between("nummer", 1000, 1100);
17    Predicate namePredicate = Predicates.like("name", "Alp%");
18    Predicate predicate = Predicates.and(zugnummerPredicate, namePredicate);
19    Collection<Zugfahrt> zugfahrten = map.values(predicate);
```

- Distributed SQL

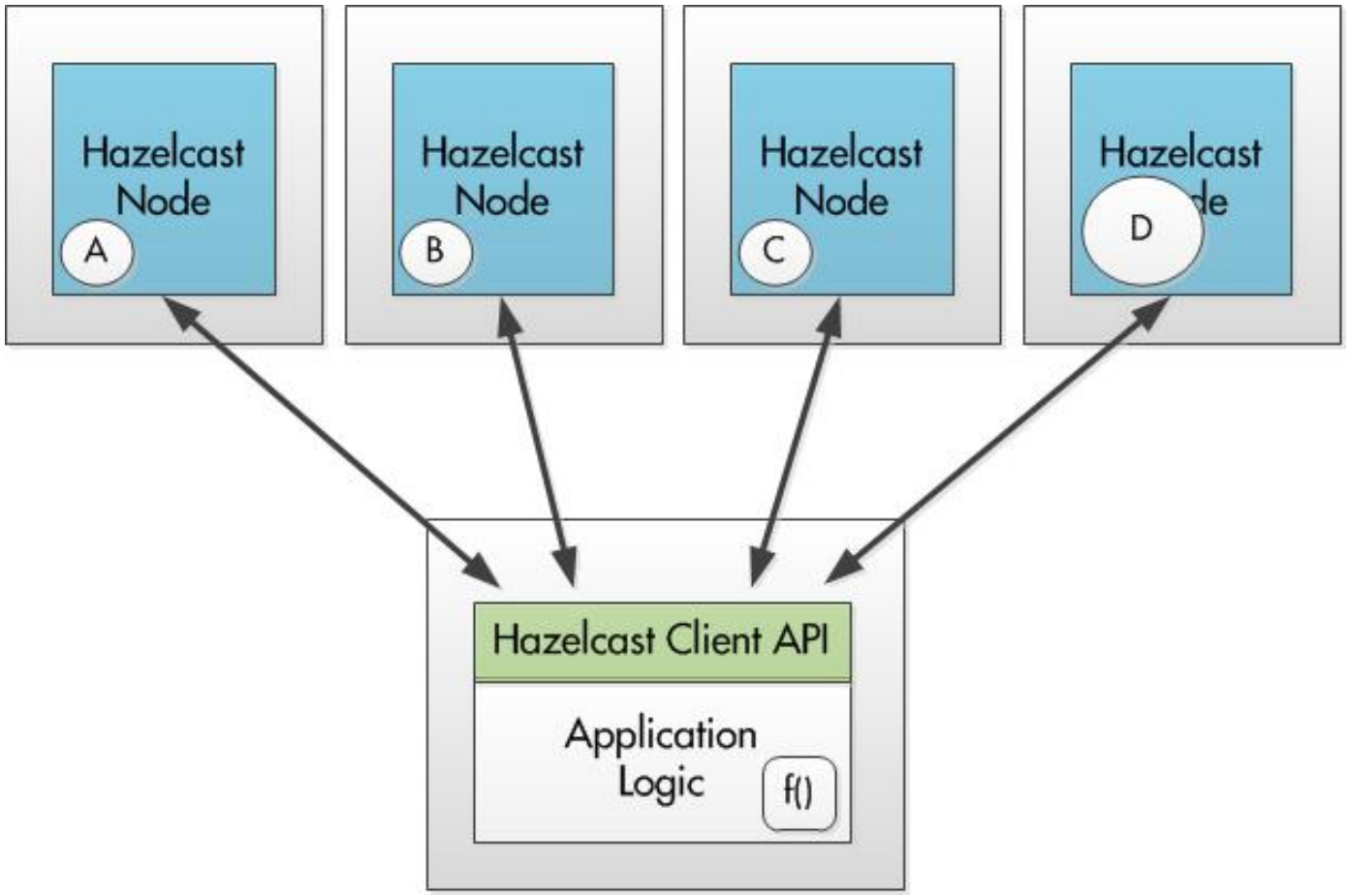
```
22    zugfahrten = map.values(new SqlPredicate(
23        "(nummer BETWEEN 1000 AND 1100) AND name LIKE 'Alp%')"));
```

- Indizes

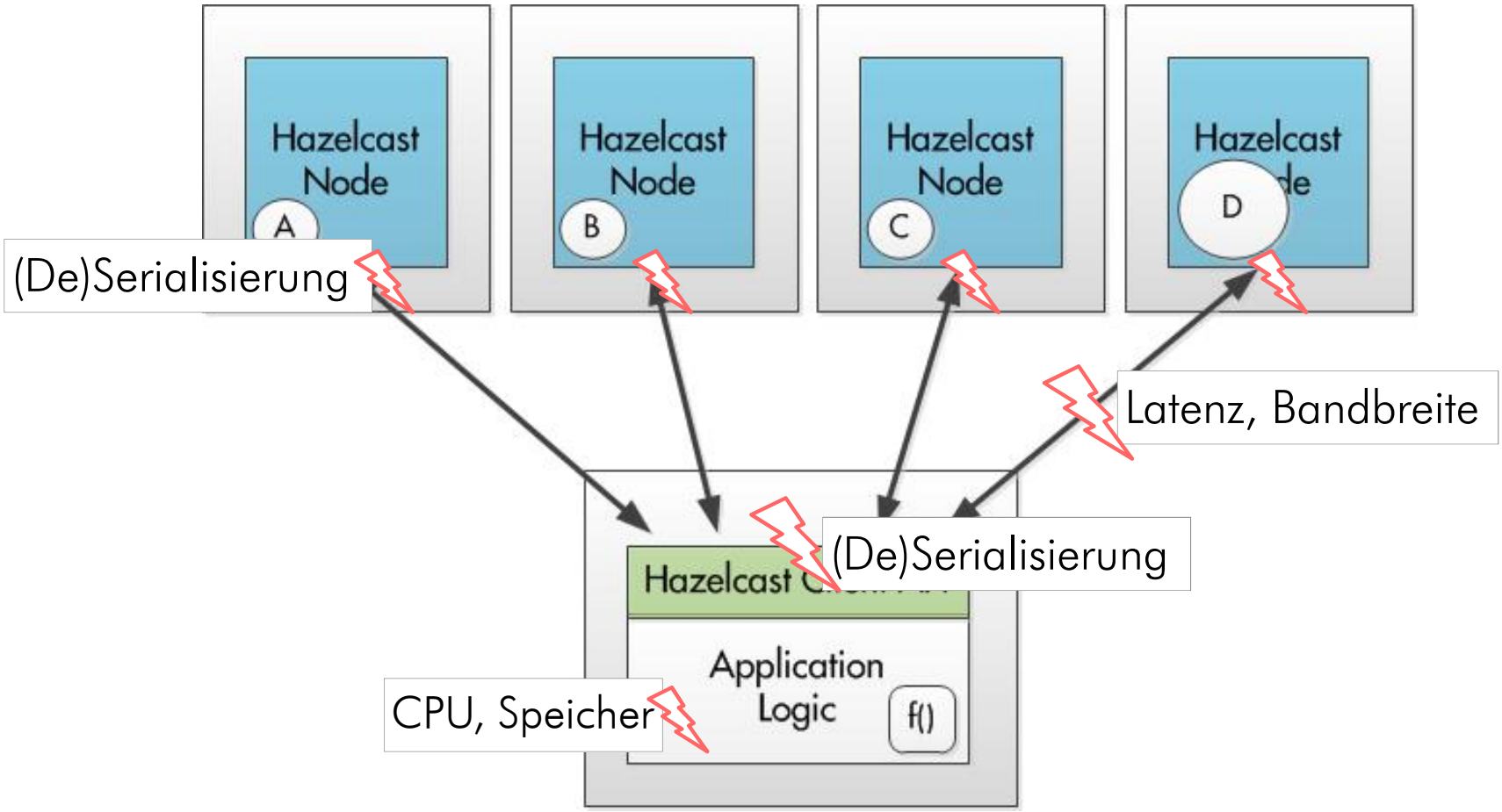
```
24 <map name="zugfahrtMap">
25     <indexes>
26         <index ordered="true">nummer</index>
27         <index ordered="false">typ</index>
28     </indexes>
29 </map>
```

- MapReduce & Aggregators (avg, sum, min, max, eigene..)

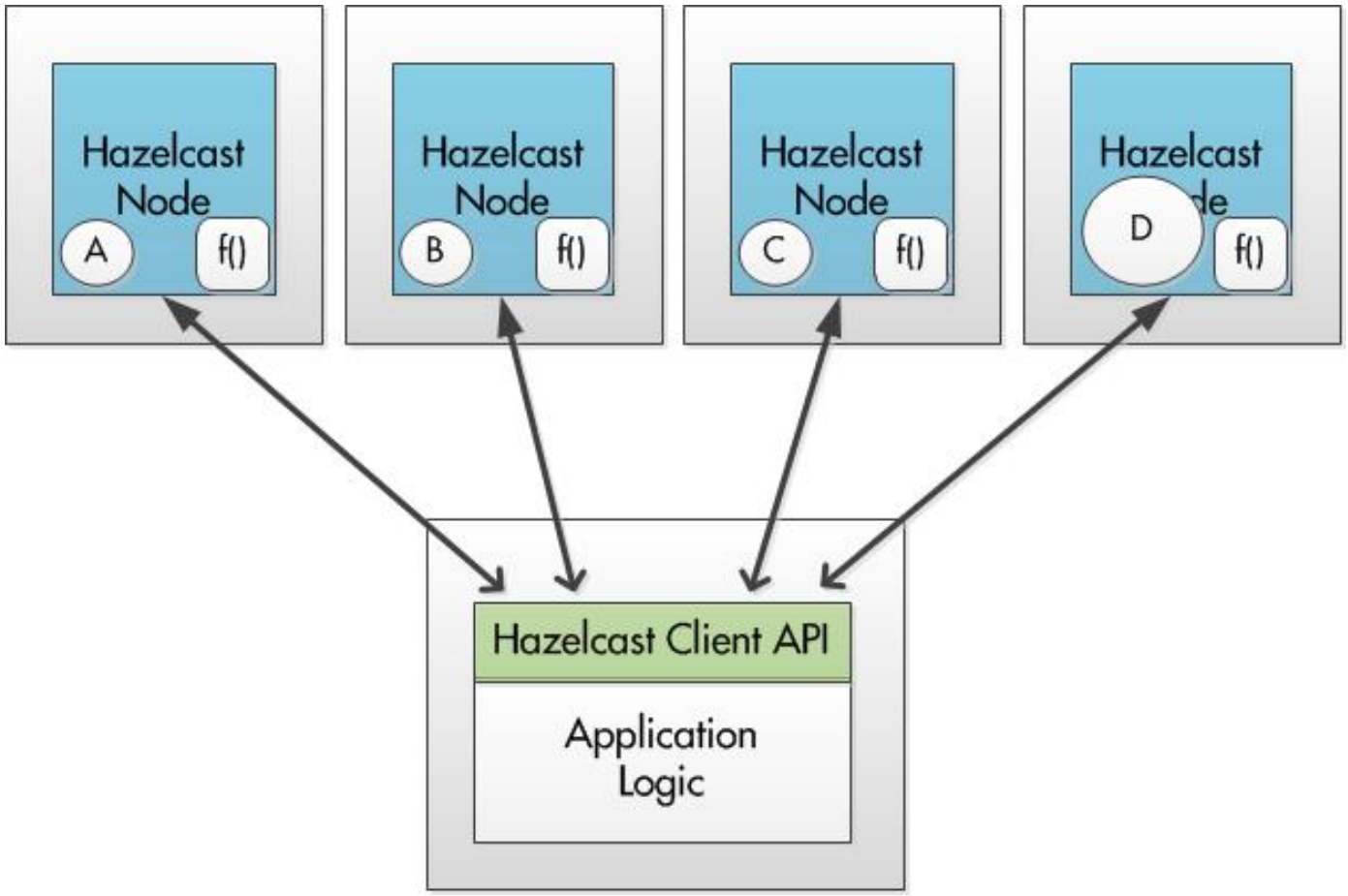
Distributed Computing: Motivation



Distributed Computing: Motivation



Distributed Computing: Motivation



Distributed Computing

- Implementierung von ExecutorService (`java.util.concurrent`) für verteilte Umgebungen (`Callable`, `Runnable`)

```
15    IExecutorService executorService = client.getExecutorService("executor");
16    Future<Boolean> future = executorService.submitToKeyOwner(
17        new ZugfahrtPunktAusfallTask(key), key);
```

- Entry Processor, um Daten lokal und atomar zu modifizieren.

```
37  Map<ZugfahrtKey, Object> oldStatus = map.executeOnEntries(
38      new ZugfahrtStatusUpdater("cruising"), predicate);
```

Distributed Computing: EntryProcessor

```
 9 public class ZugfahrtStatusUpdater extends AbstractEntryProcessor<ZugfahrtKey, Zugfahrt> {
10
11   private final String status;
12
13   public ZugfahrtStatusUpdater(String status) {
14     this.status = status;
15   }
16
17   @Override
18   public Object process(Entry<ZugfahrtKey, Zugfahrt> entry) {
19     Zugfahrt value = entry.getValue();
20     String oldStatus = value.getStatus();
21     value.setStatus(status);
22     entry.setValue(value);
23     return oldStatus;
24   }
25 }
```

```
35   EntryObject entry = new PredicateBuilder().getEntryObject();
36   Predicate predicate = entry.get("duration").greaterThan(HOURS.toSeconds(1));
37   Map<ZugfahrtKey, Object> oldStatus = map.executeOnEntries(
38       new ZugfahrtStatusUpdater("cruising"), predicate);
```

Distributed Computing: ExecutorService

```
13 public class ZugfahrtPunktAusfallTask implements Callable<Boolean>,
14   HazelcastInstanceAware, Serializable {
15
16   @Override
17   public Boolean call() throws Exception {
18
19     Map<ZugfahrtPunktKey, ZugfahrtPunkt> zugfahrtPunktMap = hazelcastInstance.getMap("zugfahrt
20     Map<ZugfahrtKey, Zugfahrt> zugfahrtMap = hazelcastInstance.getMap("zugfahrtMap");
21     ITopic<Zugfahrt> updateTopic = hazelcastInstance.getTopic("zugfahrtAusfallTopic");
22
23     ZugfahrtPunkt zugfahrtPunkt = zugfahrtPunktMap.get(zugfahrtPunktKey);
24     zugfahrtPunkt.setAusfall(true);
25     zugfahrtPunktMap.put(zugfahrtPunktKey, zugfahrtPunkt);
26
27     Zugfahrt zugfahrt = zugfahrtMap.get(zugfahrtPunktKey.getZugfahrtKey());
28     if(zugfahrt.hasAusfall()) {
29       return false;
30     }
31     else {
32       zugfahrt.setAusfall(true);
33       updateTopic.publish(zugfahrt);
34       zugfahrtMap.put(zugfahrt.getKey(), zugfahrt);
35       return true;
36     }
37   }
```

Distributed Computing: ExecutorService

```
15    IExecutorService executorService = client.getExecutorService("executor");
16    Future<Boolean> future = executorService.submitToKeyOwner(
17        new ZugfahrtPunktAusfallTask(key), key);
18    System.out.println(future.get());
```

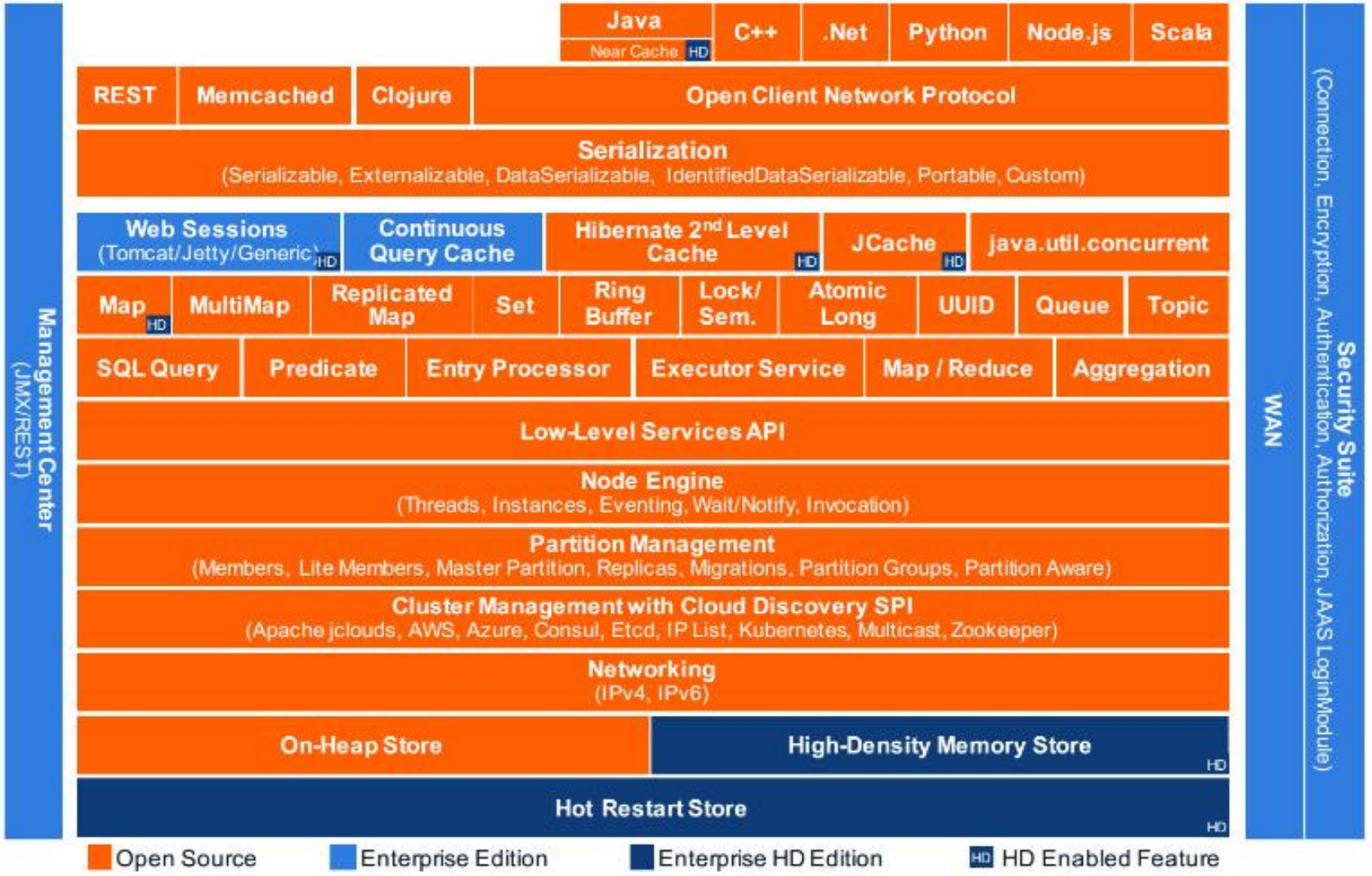
Messaging

- publish/subscribe à la JMS, aber kein JMS-Provider
- Topics können dynamisch angelegt werden
- Leichtgewichtig, da keine dedizierte Messaging-Middleware

Messaging

```
18  @Test
19  public void testTopic() {
20    ITopic<String> topic1 = Hazelcast.newHazelcastInstance().getTopic("zugfahrtTopic");
21    ITopic<String> topic2 = Hazelcast.newHazelcastInstance().getTopic("zugfahrtTopic");
22
23    CountDownLatch latch1 = new CountDownLatch(1);
24    topic1.addMessageListener(new MessageListener<String>() {
25      @Override
26      public void onMessage(Message<String> message) {
27        assertEquals("Test", message.getMessageObject());
28        latch1.countDown();
29      }
30    });
31
32    CountDownLatch latch2 = new CountDownLatch(1);
33    topic2.addMessageListener(new MessageListener<String>() {
34      @Override
35      public void onMessage(Message<String> message) {
36        assertEquals("Test", message.getMessageObject());
37        latch2.countDown();
38      }
39    });
40
41    topic1.publish("Test");
42
43    try {
44      assertTrue(latch1.await(1, TimeUnit.SECONDS));
45      assertTrue(latch2.await(1, TimeUnit.SECONDS));
46    } catch (InterruptedException ignored) {
47    }
48  }
```

Features / Editions

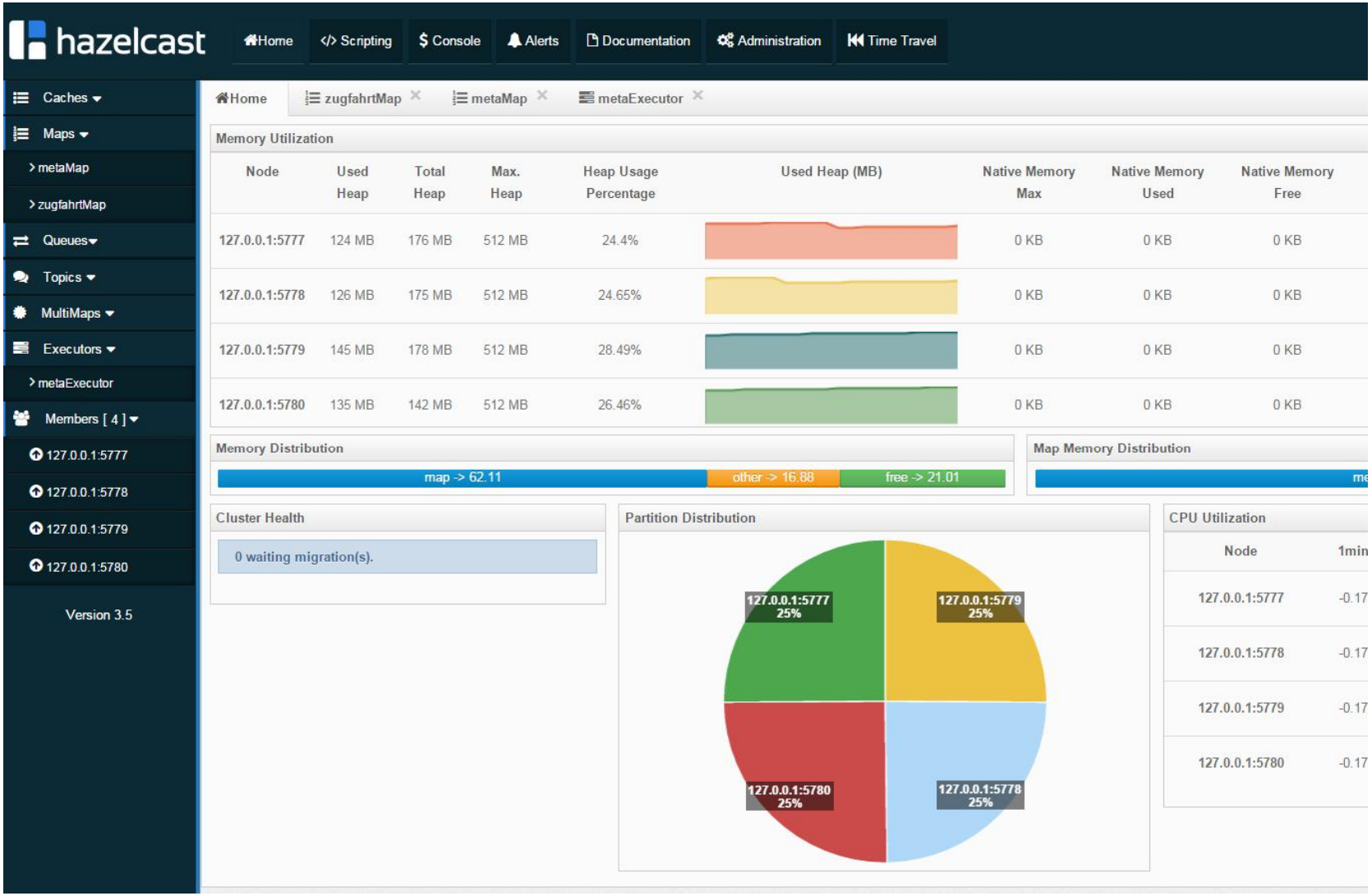


http://docs.hazelcast.org/docs/3.6/manual/html-single/index.html

Features: Was sonst

- Event Listeners (Member, Client, Entry..)
- Clients (Java, C++, .NET, REST)
- JCache (JSR-107) Implementierung
- Integration: Hibernate (2nd Level Cache), Web Session Replication, Spring Framework
- Enterpriseyness:
  - Transactions (XA, JEE Integration über JCA)
  - WAN Replication
  - Security
  - ..

Management Center



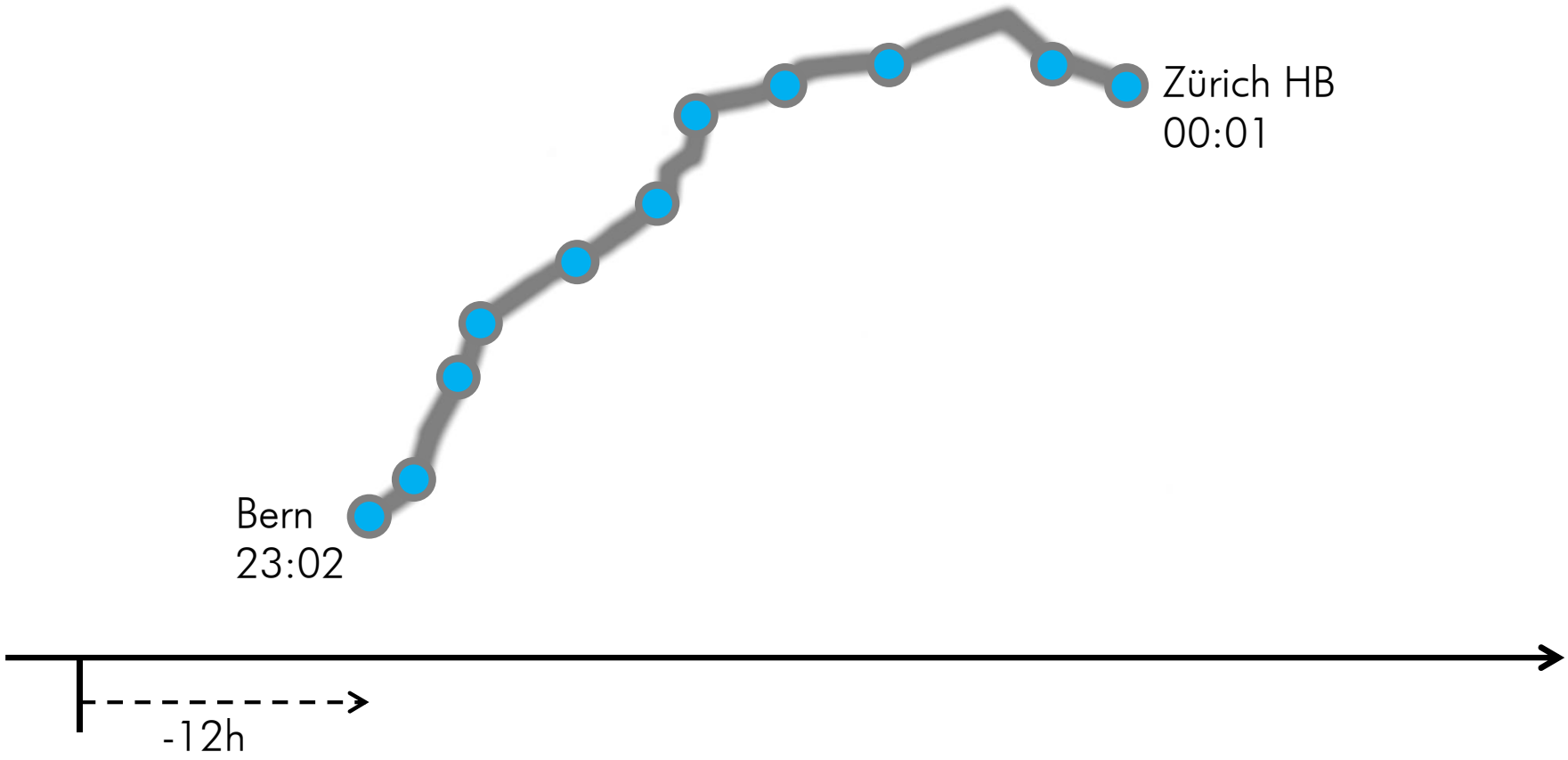
Cargo Information System – CIS Infra

«Ein System zum Planen, Verwalten und Durchführen des Güterverkehrs. Liefert aktuelle, lückenlose Informationen vor, während und nach dem Transport, nicht nur innerhalb der SBB, sondern auch gegenüber Kunden und anderen Bahnen können mit diesem Informatik-System Informationen gegeben werden.»

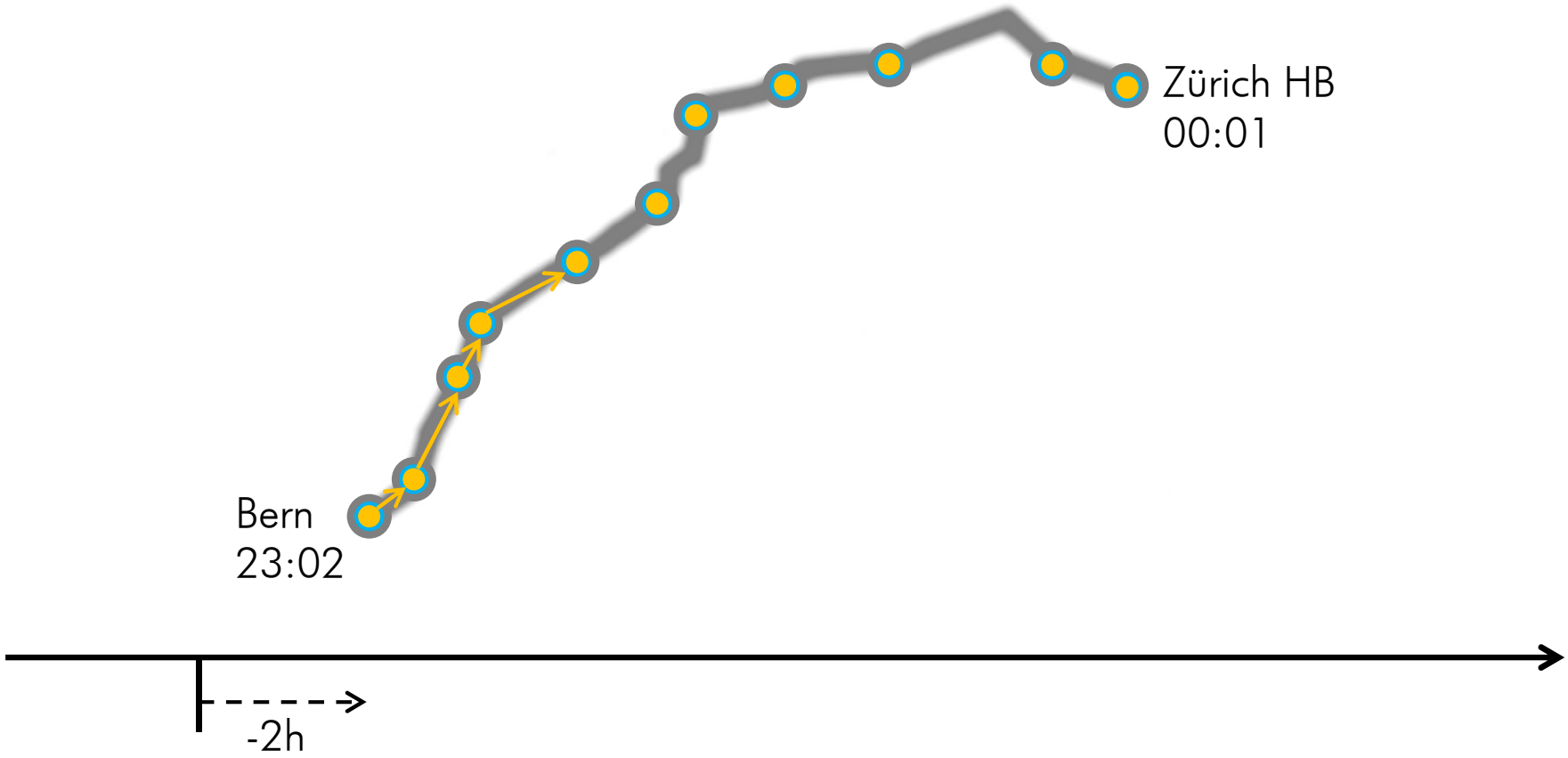
- → Laufende Migrationsprojekte Mainframe → Java
- → Viele Umsysteme, viele Schnittstellen
- → Performance-hungrig

InterCity 839

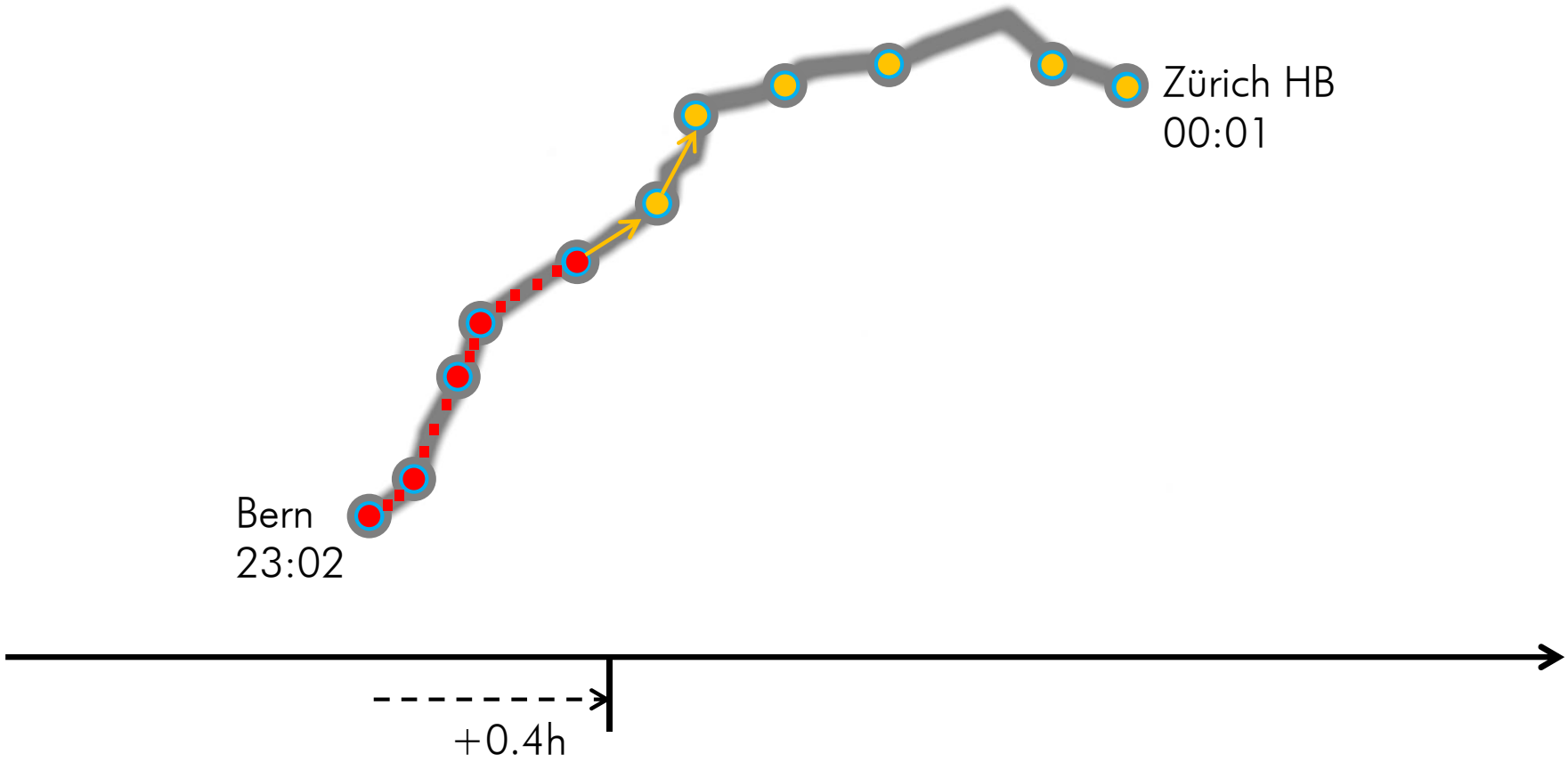
Soll-Zeiten (Rail Control System)

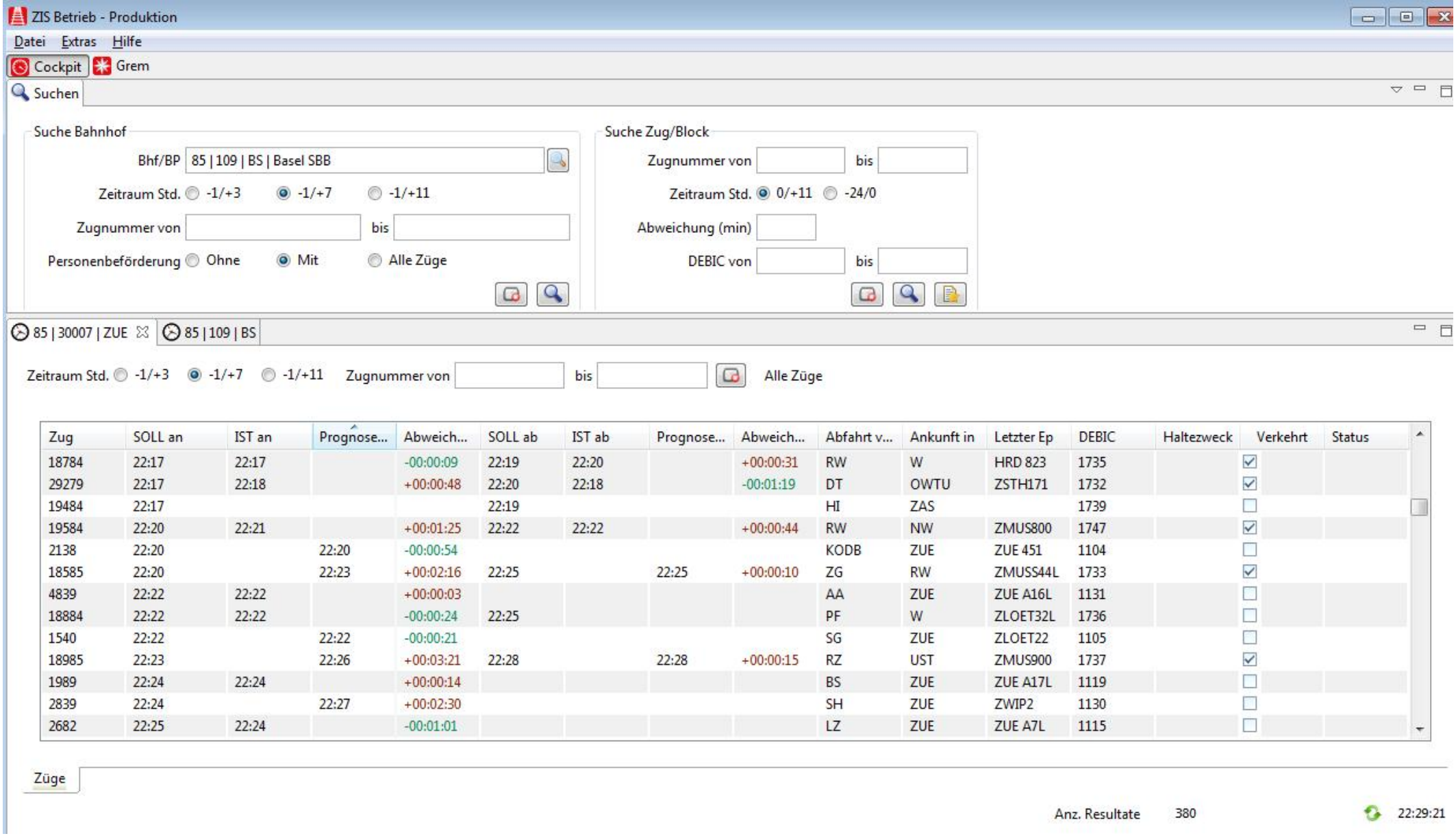


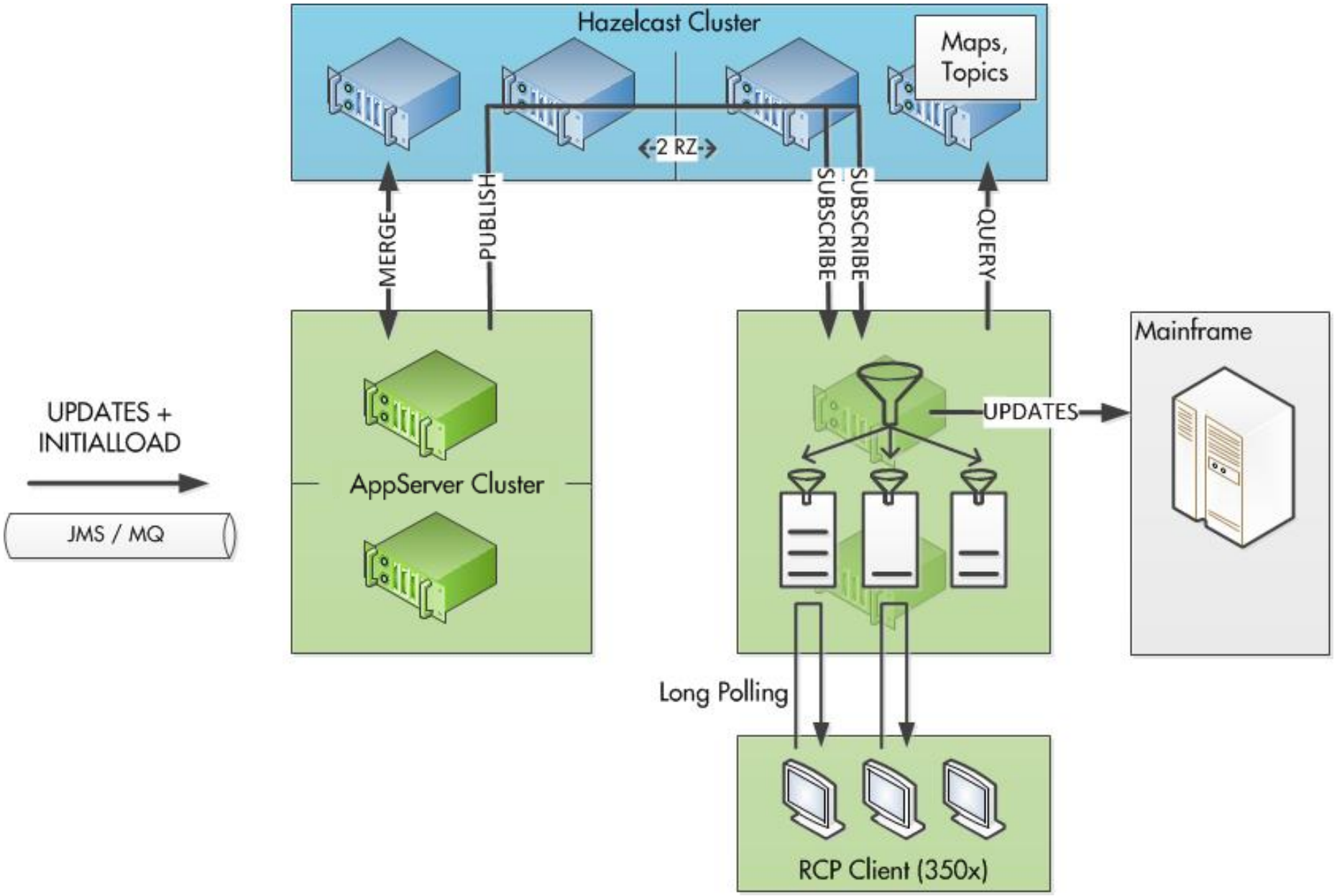
Prognose-Zeiten (RCS)

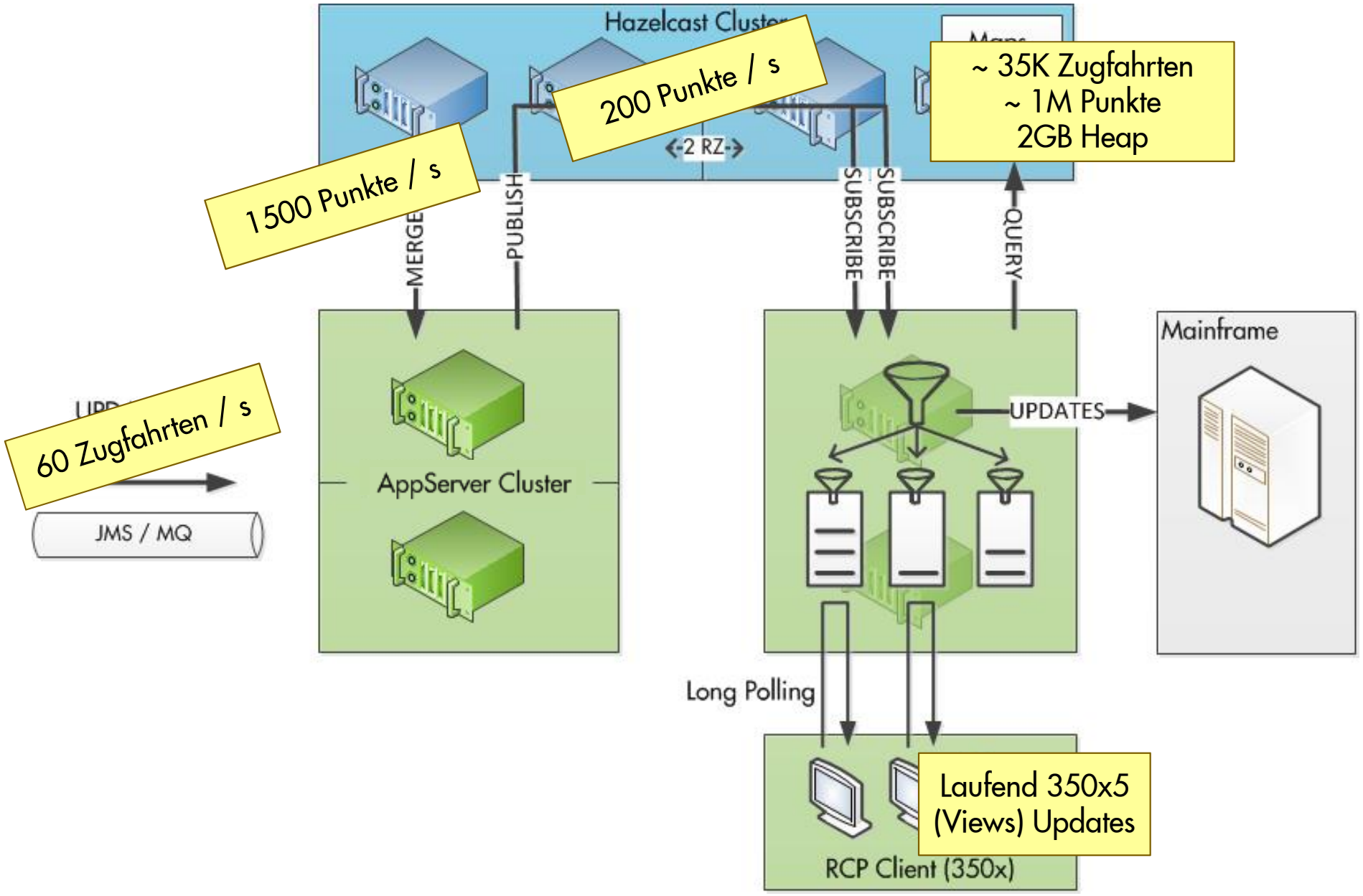


Ist-Zeiten (RCS)




ZIS Betrieb - Produktion
Datei Extras Hilfe
Cockpit Grem
Suchen
Suche Bahnhof
Bhf/BP 85 | 109 | BS | Basel SBB
Zeitraum Std. -1/+3 -1/+7 -1/+11
Zugnummer von bis
Personenbeförderung Ohne Mit Alle Züge
Suche Zug/Block
Zugnummer von bis
Zeitraum Std. 0/+11 -24/0
Abweichung (min)
DEBIC von bis
85 | 30007 | ZUE
85 | 109 | BS
Zeitraum Std. -1/+3 -1/+7 -1/+11 Zugnummer von bis Alle Züge
Zug SOLL an IST an Prognose... Abweich... SOLL ab IST ab Prognose... Abweich... Abfahrt v... Ankunft in Letzter Ep DEBIC Haltezweck Verkehrt Status
18784 22:17 22:17 -00:00:09 22:19 22:20 +00:00:31 RW W HRD 823 1735
29279 22:17 22:18 +00:00:48 22:20 22:18 -00:01:19 DT OWTU ZSTH171 1732
19484 22:17 22:19 HI ZAS 1739
19584 22:20 22:21 +00:01:25 22:22 22:22 +00:00:44 RW NW ZMUS800 1747
2138 22:20 22:20 -00:00:54 KODB ZUE ZUE 451 1104
18585 22:20 22:23 +00:02:16 22:25 22:25 +00:00:10 ZG RW ZMUSS44L 1733
4839 22:22 22:22 +00:00:03 AA ZUE ZUE A16L 1131
18884 22:22 22:22 -00:00:24 22:25 PF W ZLOET32L 1736
1540 22:22 22:22 -00:00:21 SG ZUE ZLOET22 1105
18985 22:23 22:26 +00:03:21 22:28 22:28 +00:00:15 RZ UST ZMUS900 1737
1989 22:24 22:24 +00:00:14 BS ZUE ZUE A17L 1119
2839 22:24 22:27 +00:02:30 SH ZUE ZWIP2 1130
2682 22:25 22:24 -00:01:01 LZ ZUE ZUE A7L 1115
Züge
Anz. Resultate 380
22:29:21


Hazelcast Cluster
Maps,
Topics
2 RZ
MERGE
PUBLISH
SUBSCRIBE
SUBSCRIBE
QUERY
UPDATES +
INITIALLOAD
JMS / MQ
AppServer Cluster
UPDATES
Mainframe
Long Polling
RCP Client (350x)


Hazelcast Cluster
<2 RZ>
200 Punkte / s
1500 Punkte / s
~ 35K Zugfahrten
~ 1M Punkte
2GB Heap
MERGE
PUBLISH
SUBSCRIBE
SUBSCRIBE
QUERY
60 Zugfahrten / s
JMS / MQ
AppServer Cluster
UPDATES
Mainframe
Long Polling
Laufend 350x5
(Views) Updates
RCP Client (350x)


http://lazysundaymag.com/?p=322