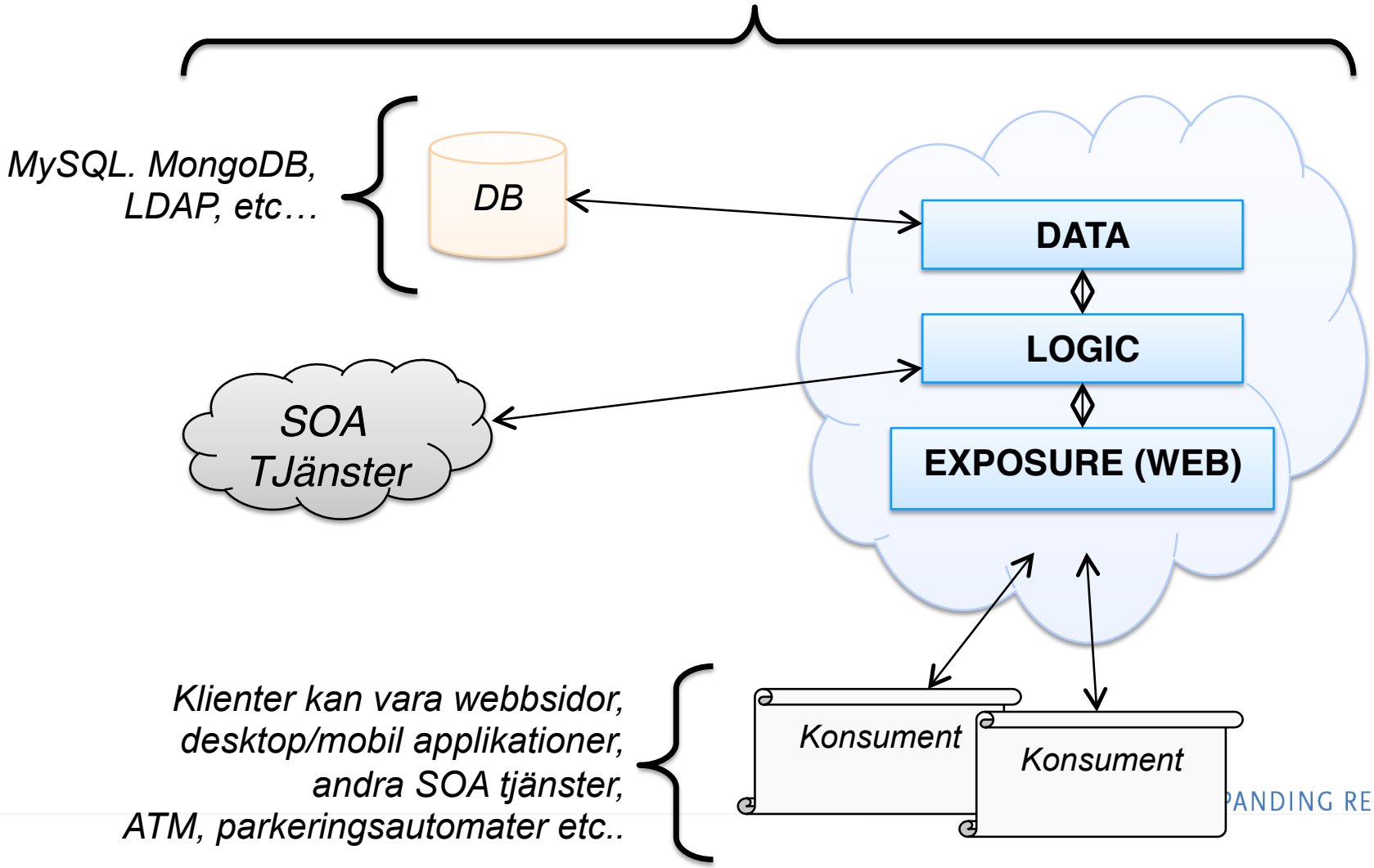
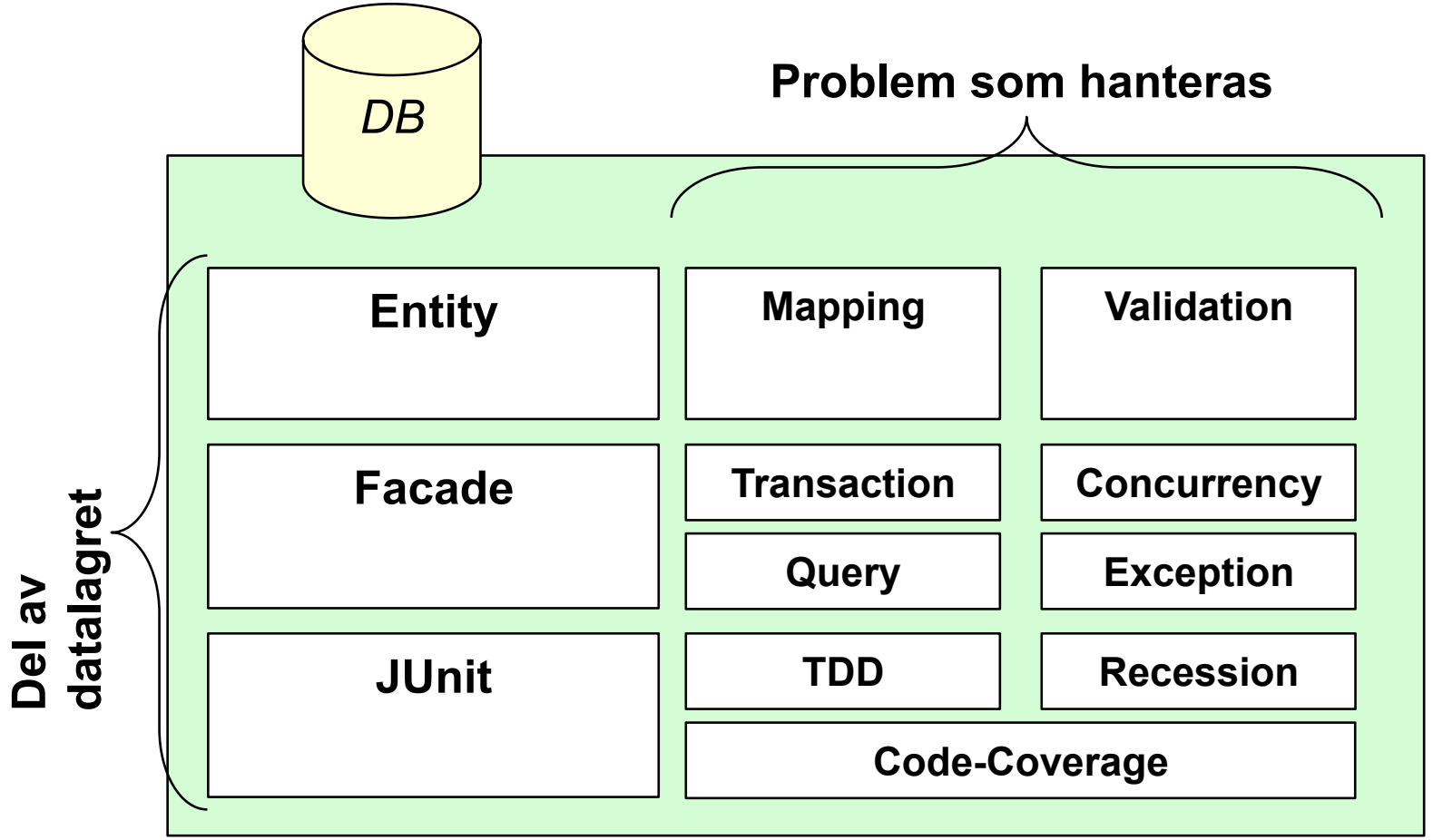
FÖRELÄSNING 4

Concurrency,Webblager, Exceptions,Kafka, och att bygga skiten bra

Backend – DB, DATA, LOGIC, EXPOSURE (WEB)

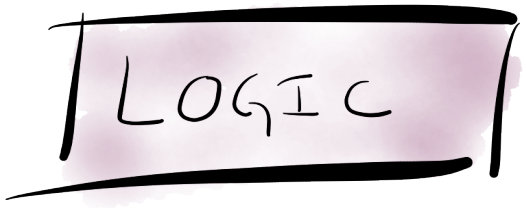


Datalager


DB
Problem som hanteras
Del av datalagret
Entity
Mapping
Validation
Facade
Transaction
Concurrency
Query
Exception
JUnit
TDD
Recession
Code-Coverage

Business Logic Layer

- Business Logic – Kan förklaras som de funktioner som får ”företaget” eller ”systemet” att fungera.
- T ex för en p-automat kan det vara – **printTicket**, **ejectChange** etc. För en webbplats kan det vara **registerUser**, **loginUser** etc.
- Det är de funktioner som är av intresse för omvärlden att använda sig av, som ger värde till företaget, därav "business logic"

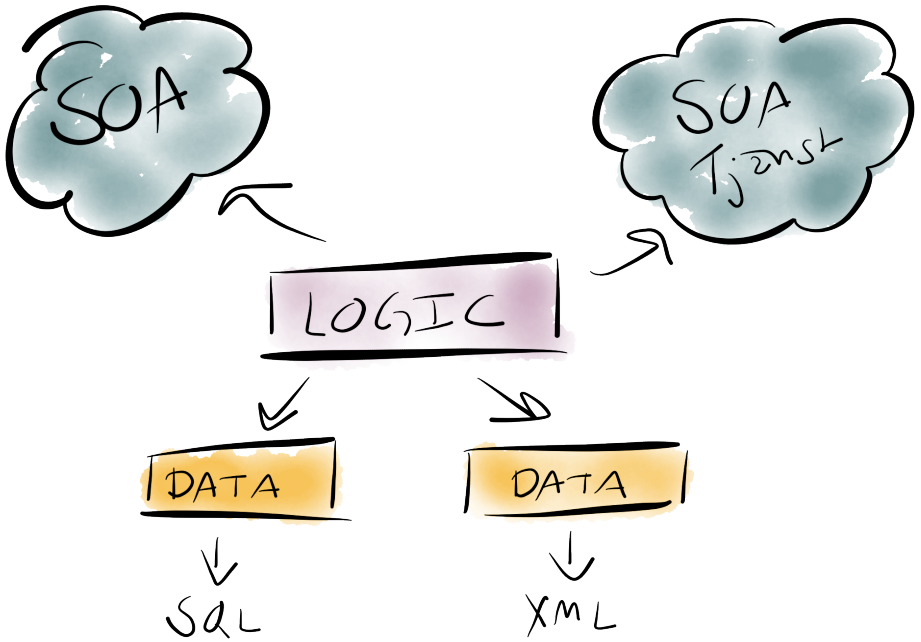


Business Logic Layer

- För att skapa värde till företaget behöver logiklagret göra två saker:

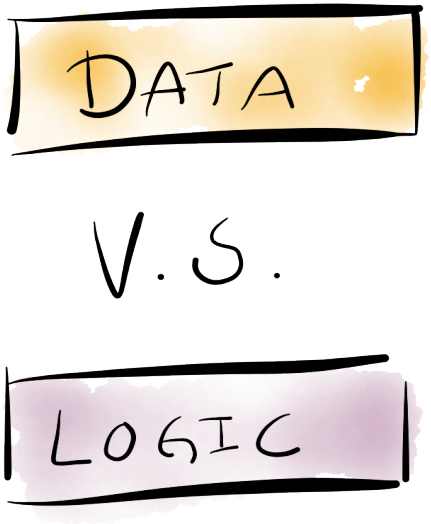
- 1. Anrop till externa tjänster inom och utom företaget. Man kan behöva anropa t ex openID för inloggning, skapa Git konton, skicka faktura, skriva till Twitter etc....

- 2. Kommunicera med datalagret (man skulle kunna ha fler datalager i samma SOA tjänst)



Data vs Logic

- I datalagret har man: create(...)
- I logiklagret har man: register(...)
- **register** kommer behöva **create**, men funktionen kanske också måste göra en hel del annat.
- Att skriva till databasen är ett steg i processen, men det kanske behövs ett Git konto, Apache server, etc. som måste skapas, det kan inte datalagret.
- Datalagret kan returnera data till logiklagret, men kan inte anropa logiklagret. Datalagret har inte ens tillgång till logiklagret vid kompilering, så man kommer inte så långt med ett sådant arbete.



Logic Layer – Externa tjänster

- Hur man anropar externa tjänster är oftast olika beroende på vad man anropar (Twitter, Facebook, Google, Amazon etc).
- I kodskelettet finns det en fil HTTPHelper som kan användas för att enklare göra GET anrop till servrar.
- Oftast får man titta i dokumentationen för tjänster hur man gör, men ibland så finns det kompilerade bibliotek som man bara inkluderar i sitt system och sedan kan man anropa tjänsterna som om de var vanliga Java funktioner (t ex Monlog).

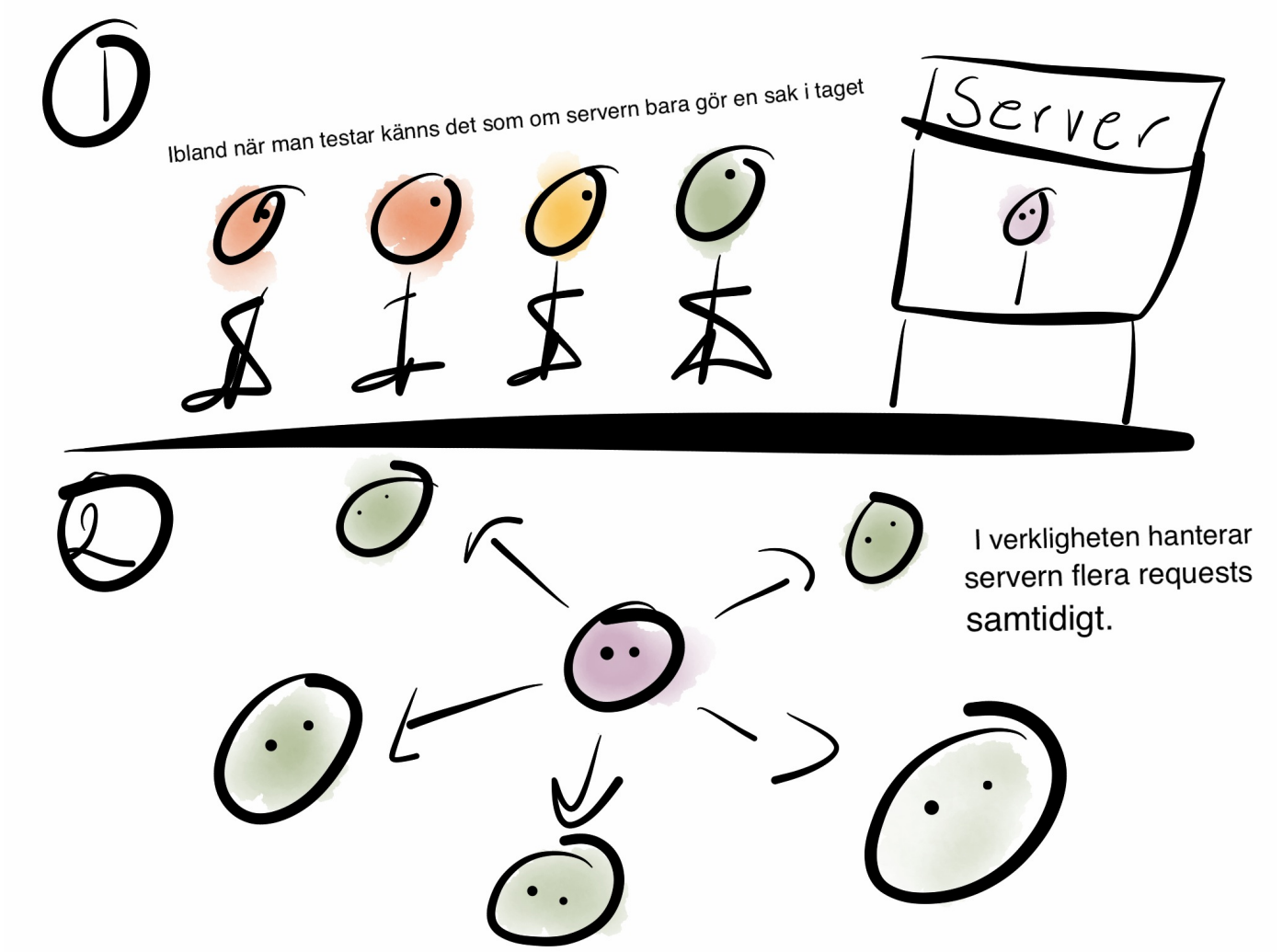
Logic Layer – Consistency och Rollback

- Något som är väldigt svårt att generalisera, eller även definiera är *rollbacks* i logik lagret.

- Antag att vi gör tre steg:
  - Lägg en beställning på vara i databasen.
  - Skicka faktura till kund.
  - Skriv till Twitter att vi sålt en vara.

- Om vi inte lyckas skicka fakturan, hur hanterar vi det?
- Om vi inte lyckas skriva till Twitter, hur hanterar vi det?

- Utanför ramen för denna kurs, men en viktigt aspekt.


1
Ibland när man testar känns det som om servern bara gör en sak i taget
Server
2
I verkligheten hanterar servern flera requests samtidigt.

JPA - Concurrency

- Våra tjänster har inte bara en tråd
- Räkna med att för varje ny användare skapas en ny tråd, och då ofta en ny EntityManager
- Vad händer om man försöker köra samma funktion samtidigt, t ex uppdatera en Todo ?
- ***Pessimistisk****:* Lås hela raden som skall ändras, ingen annan för röra den tills tråden är klar, lås upp när vi är klara
- ***Optimistisk****:* Hämta objekt och ändra som vanligt, men om objektet har ändrats i databasen när vi försöker skriva så fallerar funktionen

JPA – Concurrency Exempel

```
…
public long update(long id, String title, String body) {

   …
      Todo todo = em.find(TodoDB.class, id, LockModeType.PESSIMISTIC_WRITE);
   …
}
```

För andra varianter av LockModeType kolla länkar under kurslitteratur på hemsidan

JPA – Concurrency – Klassiskt exempel

```
(OBS: Pseudo kod)

function debit(amount) {

   Account account = findAccount();

   if(account.credit >= amount) {
      account.credit = account.credit - amount;
      persistToDatabase(account);
      return true;
   } else {
      return false;
   }

}
```

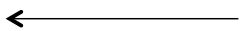
Antag att Account bara har 100kr på sitt konto, och
vi får in två requests samtidigt med en debitering på 100kr

JPA – Concurrency – Klassiskt exempel

(OBS: Pseudo kod)

Tråd 2 väntar….

```
function debit(amount) {

   Account account = findAccount();

   if(account.credit >= amount) {
      account.credit = account.credit - amount;
      persistToDatabase(account);
      return true;
   } else {
      return false;
   }

}
```



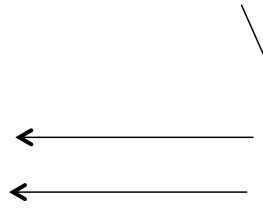
Tråd 1

Antag att Account bara har 100kr på sitt konto, och
vi får in två requests samtidigt med en debitering på 100kr

JPA – Concurrency – Klassiskt exempel

```
(OBS: Pseudo kod)

function debit(amount) {

   Account account = findAccount();

   if(account.credit >= amount) {
      account.credit = account.credit - amount;
      persistToDatabase(account);
      return true;
   } else {
      return false;
   }

}
```

CPU'n har valt att arbeta med tråd 2, tråd 1 står därmed still.



Tråd 2 debiterar och skriver till, databasen. Account har nu 0kr på sitt konto.

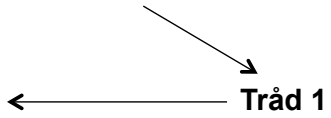
```
 Antag att Account bara har 100kr på sitt konto, och
 vi får in två requests samtidigt med en debitering på 100kr
```

JPA – Concurrency – Klassiskt exempel

```
(OBS: Pseudo kod)

function debit(amount) {

   Account account = findAccount();

   if(account.credit >= amount) {
      account.credit = account.credit - amount;
      persistToDatabase(account);
      return true;
   } else {
      return false;
   }

}
```

Tråd 2 är klar…

Tråd 1 har redan kollat att account hade >= 100kr, så den kör på.

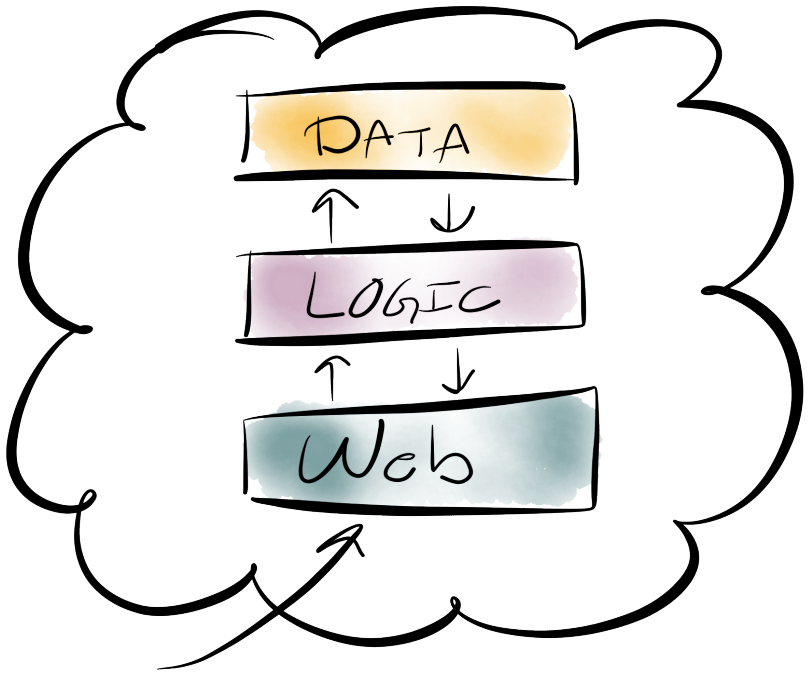


När Tråd 1 är klar har vi debiterat, 200 kr från account (som bara hade 100kr) och account.credit == 0

```
 Antag att Account bara har 100kr på sitt konto, och
 vi får in två requests samtidigt med en debitering på 100kr
```

Web Layer

- Sista delen i vår SOA tjänst är webblagret
- Här exponerar vi vår logik med hjälp av web tjänster enligt REST principen
- Lagret kommunicerar endast direkt med logiklagret, men får inte ändra något i logiken av systemet
- För att förstå styrkan av detta lager måste man föreställa sig en situation där man byter ut HTTP mot t ex CORBA eller liknande



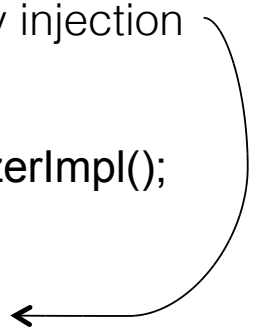
LiU EXPANDING REALITY

Web Service – REST

- När man utvecklar REST tjänster så vill man definiera olika URL'er för olika funktioner
- Java EE har ett bibliotek för detta som kallas JAX-RS eller ibland det vänligare *Jersey*
- Jersey är en förlängning av Java EEs vanliga bibliotek för HTTP hantering som kallas *Servlets*
- Alla filer som tillhör tjänsten (inklusive data och logik biblioteken) paketeras i en sk. ”war” fil
- **war** filen placeras sedan i Glassfish och tjänsterna startas och finns tillgängliga
- war = Web Application Archive

JAX-RS - POJO - Exempel

```
public class TodoService {

   private TodoJsonSerializer jsonSerializer = new TodoJsonSerializerImpl();
   private TodoLogicFacade todoLogicFacade =
           new TodoLogicFacadeImpl(new TodoEntityFacadeImpl());

   public Response find(long id) {
           Todo todo = todoLogicFacade.find(id);
           String json = jsonSerializer(todo);
           return Response.ok().entity(json).build();
    }
}
```



LiU EXPANDING REALITY

JAX-RS - POJO - Exempel

```
@Path(”/todo”)
public class TodoService {
…

   @GET
   @Path(”find”)
   public Response find(@QueryParam(”id”) long id) {
         Todo todo = todoLogicFacade.find(id);
         String json = jsonSerializer(todo);
         return Response.ok().entity(json).build();
    }
}
```


LiU EXPANDING REALITY

Vi kör med Spring ja eller Spring Boot

Spring **Boot**®

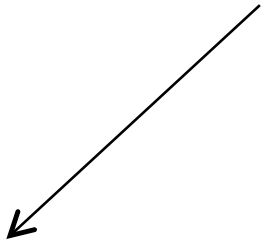
JAX-RS - Exceptions

- Om något gått fel i anropen till logiklagret (och därmed datalagret) så vill vi rapportera detta till den som gjorde anropet
- Med REST görs detta ofta med HTTP felkoder
- I laboration 1 nöjer vi oss med att helt enkelt rapportera att något gått fel, och inte i detalj rapportera vad som har gått fel.
- Det finns möjlighet för studenter i laboration 2 att fördjupa sig just i denna problematik, hur man förmedlar information om vad som gått fel och avancerade exceptions, ändringar behöver ske hela vägen ner i datalagret.

JAX-RS – Junit

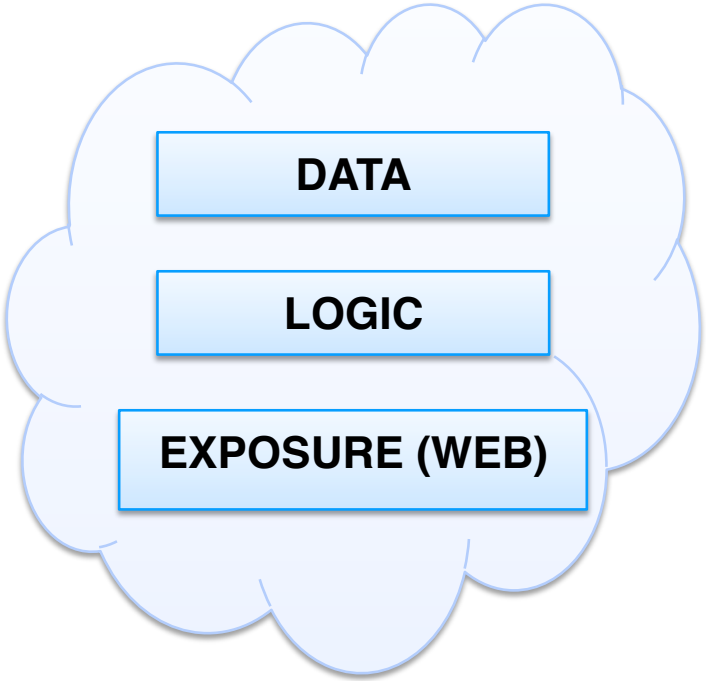
- Eftersom våra tjänster är rena Java klasser (POJO) så kan vi skapa instanser av dessa med Java kod och testa funktionerna, vi behöver alltså inte skapa anrop till URL'erna över HTTP (även om det kan vara smart att även testa detta)

Vi kollar att vi returnerar korrekt JSON



```
@Test
public void test() {

    TodoService service = new TodoService();
    Response response = service.find(1);
    Assert.assertEquals("{'id':1, 'title': ….", (String)response.getEntity());
    ….
}
```

SOA Tjänst – DONE!



Det finns ett antal saker kvar att studera innan vi kan förlita oss på tjänsten, dessa saker kan man, blanda andra, titta på i scenario 2

Det vi i huvudsak saknar är:

1. Detaljerad felhantering - nu säger vi bara att nått gått fel, inte vad.

2. Datavalidering – Hur ser vi till att vi verkligen bara sparar kontokortsnummer i den kolumn i databasen där det skall finnas ?

3. Säkerhet – Våra tjänster är vidöppna till omvärlden, och detta är inte alltid önskvärt.

Felhantering

- Vi har tidigare nämnt att vi inte sköter felhantering på ett tillfredställande sätt.

- Vi kommer använda oss utav tre klassificeringar av fel *(teoretiskt sätt så kan man skapa hur många klassificeringar man önskar, beroende på hur detaljerad man måste vara)*.

- EntityNotFoundException
- InputParameterException
- ServiceConfigurationException

LiU EXPANDING REALITY

Felhantering

- **EntityNotFoundException** – Detta fel uppstår om den som anropar koden försöker utföra något på en entity som inte finns, t ex om man skickat in id = 4 men det finns ingen entity med id 4.

- **InputParameterException** – Detta fel uppstår om man skickar in en parameter som inte är ok, t ex om man har *null* som namnet på något.

- **ServiceConfigurationException** – Detta fel uppstår när något går fel som inte är den som anropars fel, t ex. att nätverket är nere, man når inte databasen.

LiU EXPANDING REALITY

Felhantering

- Man definierar exception klasser i API:et för data lagret.
  (Det finns föreläsningskod med exceptions på kurshemsidan).

- Sedan skriver man in i kontraktet vilka fel som kan uppstå:

```
public interface TodoEntityFacade {

    public long create(String title, String body)
            throws
            TodoInputParameterException,
            TodoServiceConfigurationException;

...
```


LiU EXPANDING REALITY

Felhantering

- Sedan så hanterar vi fel i vår implementation.

```
@Override
public long create(String title, String body)
        throws
        TodoInputParameterException,
        TodoServiceConfigurationException {

if (title == null) {
 throw new TodoInputParameterException(”Title can not be null");
}

...
```


LiU EXPANDING REALITY

Felhantering

- Det blir dock lite svårare för andra typer av fel.

```
  em.getTransaction().commit();

  return todo.getId();

} catch (RollbackException e) {

    /* We have moved the rollback into the catch of rollback exception */
    if (em.getTransaction().isActive()) {
       em.getTransaction().rollback();
    }

   throw new TodoServiceConfigurationException("Creating the Todo failed due to service errors.
Please contact your database administrator.");

} finally {
    em.close();
}
```

Vilka fel kan uppstå här?

Kolla dokumentationen för JPA för att ta reda på vilka fel som kan uppstå.

T ex. kan detta orsaka ett RollbackException som vi måste ta hand om.

Men vi har lovat logik lagret att kasta ett TodoServiceConfigurationException om något går fel som de är ansvariga för, så därför gör vi det.

Felhantering

- Det blir dock lite svårare för andra typer av fel.

Vi får nu aldrig returnera 0, null, -1 eller liknande när något går fel, det skall alltid kastas exceptions.

```
  em.getTransaction().commit();

  return todo.getId();

} catch (RollbackException e) {

    /* We have moved the rollback into the catch of rollback exception */
    if (em.getTransaction().isActive()) {
       em.getTransaction().rollback();
    }

   throw new TodoServiceConfigurationException("Creating the Todo failed due to service errors.
Please contact your database administrator.");

} finally {
    em.close();
}
```


LiU EXPANDING REALITY

Vad gör logiklagret med felet?

Notera att logik lagrets kontrakt har ändrats.

```
@Override
  public long create(
        String title,
        String body)
        throws
        TodoInputParameterException,
        TodoServiceConfigurationException {
    try {
        return todoEntityFacade.create(title, body);
    } catch (TodoServiceConfigurationException e) {
        todoLogger.log(e);
        throw e;
    }

  }
```

I de flesta fall vill logiklagret bara skriva till loggen att det har hänt, och sedan kasta vidare felet till webblagret.

LiU EXPANDING REALITY

Vad gör webblagret med felet?

```
 try {

    long id = todoLogicFacde.create(title, body);
    return Response.ok().entity(id + "").build();

} catch (TodoInputParameterException e) {

    /* Något gick fel som användaren kan lösa */
    return Response.status(Response.Status.BAD_REQUEST).entity("Something is wrong with you
input").build();

} catch (TodoServiceConfigurationException e) {

   /* Något gick fel som inte användaren kan lösa */
   return Response.status(Response.Status.INTERNAL_SERVER_ERROR).entity("The service could
not handle your request, please contant your administrator").build();

}
```


LiU EXPANDING REALITY

Vad gör webblagret med felet?

Vad händer om det sker ett fel i data lagret som vi inte visste kunde hända?

Det kommer då att leta sig upp genom logik lagret till webblagret.

Som sista anhalt är det viktigt att webblagret tar hand om allt, och om ett fel uppstår som man inte kände till så ska ingen information ges till anroparen.

Att inte ta hand om alla fel i slutet kommer läcka information, detta kan vara en säkerhetsbrist.

```
} catch (Throwable e) {

   return Response.status(Response.Status.SERVICE_UNAVAILABLE).build();

}
```


LiU EXPANDING REALITY

Betyg 4- Felhantering

- Titta i koden som finns på kurshemsidan på hur det i detalj att ni skall/kan lösa uppgiften. (koden finns under föreläsningar (exception i .zip)

JPA - Logging

- Att logga alla anrop och fel gör att det blir enklare att hitta buggar när systemet väl är igång
- Genom att skriva till något persistent, istället för bara System.out, så kan vi gå tillbaka i tiden och även visualisera med HTML eller annat format
- Loggning kan också ske i form av mejl
- Men det finns ett problem med den modell vi visat här
- Vad händer om vår tjänst ligger bakom en load-balancer och är skalad till 20 servrar ?
- Alla anrop hamnar på olika servrar, så då har vi 20 filer som alla har loggat delar av en användares väg genom systemet

APACHE KAFKA: THE BACKBONE OF REAL-TIME DATA

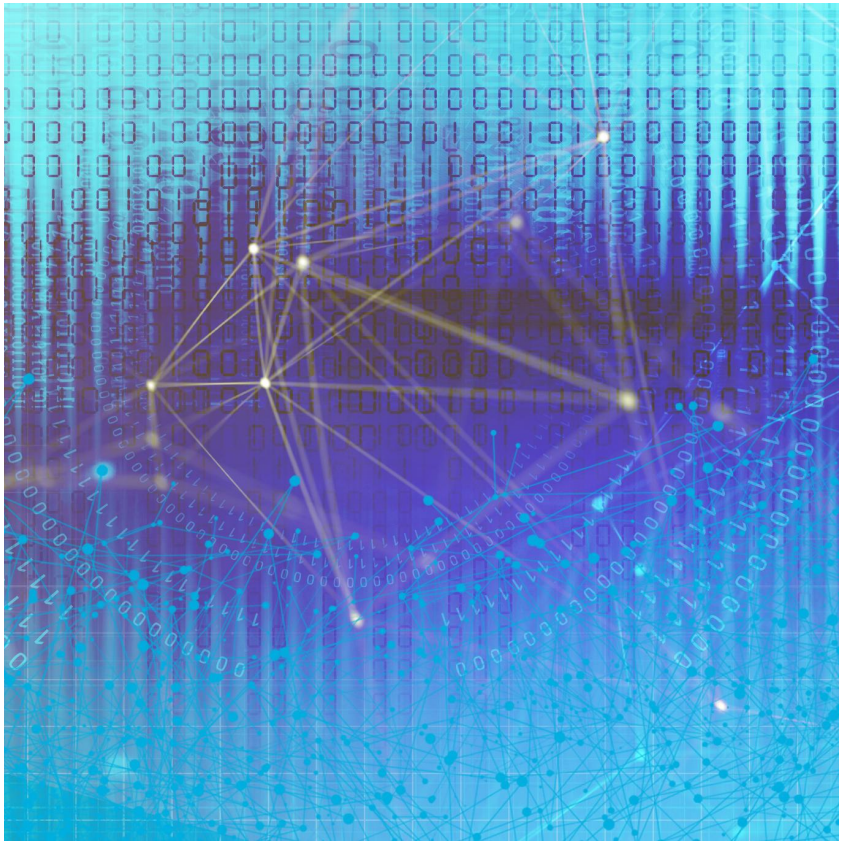
ENABLING SEAMLESS STREAMING AND PROCESSING OF DATA

Problem

Think about Twitter

Overview

- Kafka is a “publish-subscribe messaging rethought as a distributed commit log”
- Fast
- Scalable
- Durable
- Distributed



Overview of Apache Kafka

Kafka is a distributed event streaming platform crucial for modern data infrastructure and real-time data handling.

Industry Adoption

Companies like Netflix, Uber, and Spotify use Kafka for managing massive volumes of real-time data efficiently.

Importance for Developers

Understanding Kafka is essential for software engineers involved in data pipelines, microservices, and cloud-native architectures.

Core Features

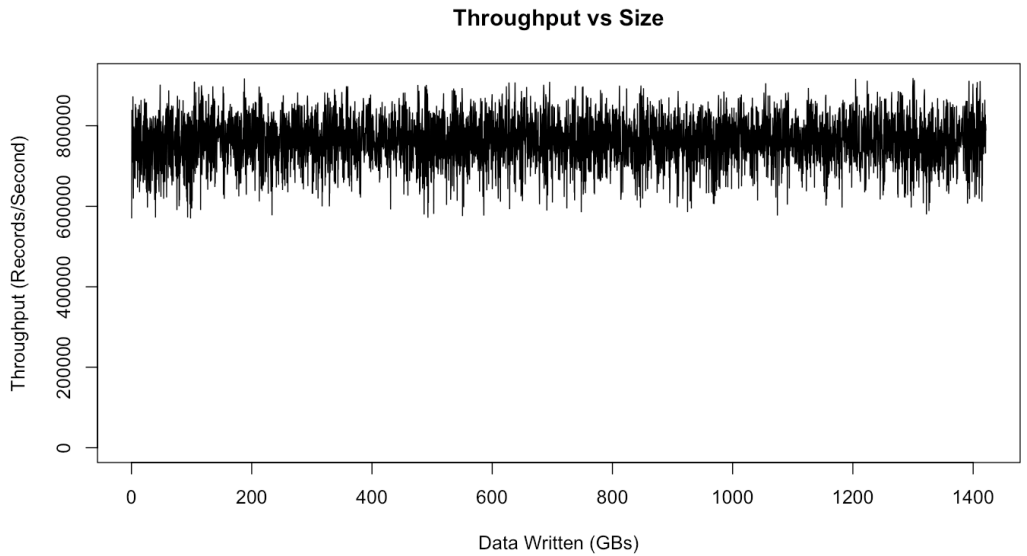
Kafka enables scalable, fault-tolerant data movement ideal for real-time analytics and event-driven applications.

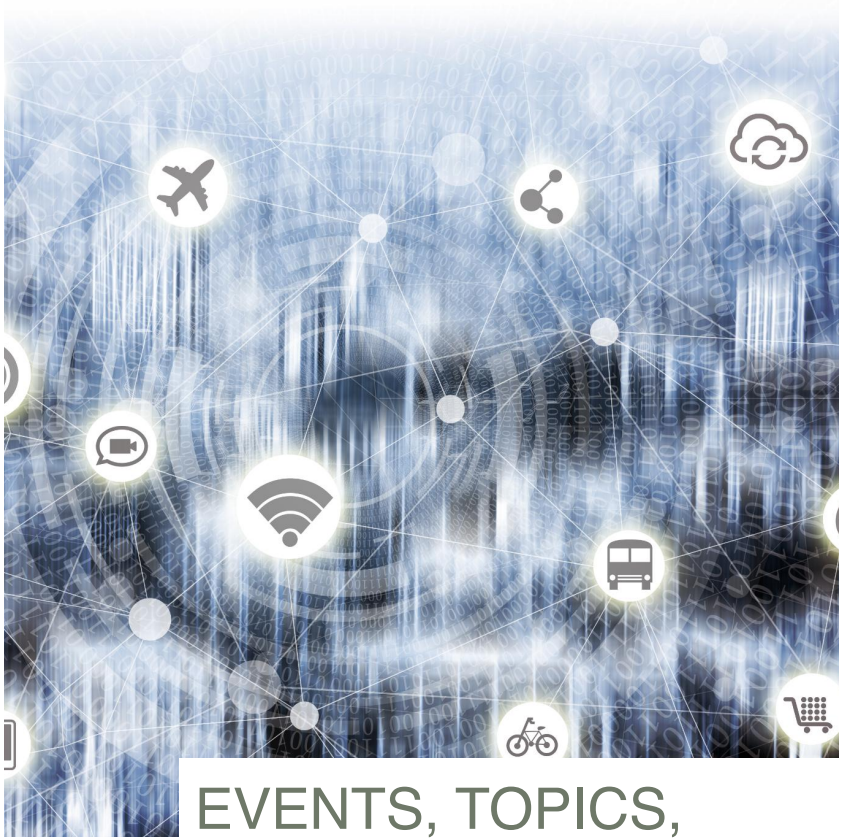
Kafka adoption and use cases

- **LinkedIn:** activity streams, operational metrics, data bus
  - 400 nodes, 18k topics, 220B msg/day (peak 3.2M msg/s), May 2014
- **Netflix**: real-time monitoring and event processing
- **Twitter**: as part of their Storm real-time data pipelines
- **Spotify**: log delivery (from 4h down to 10s), Hadoop
- **Loggly**: log collection and processing
- **Mozilla**: telemetry data
- Airbnb, Cisco, Gnip, InfoChimps, Ooyala, Square, Uber, …

How fast is Kafka?

- **“Up to 2 million writes/sec on 3 cheap machines”**
  - Using 3 producers on 3 different machines, 3x async replication
    - Only 1 producer/machine because NIC already saturated
- **Sustained throughput as stored data grows**
  - Slightly different test config than 2M writes/sec above.





EVENTS, TOPICS, PRODUCERS, CONSUMERS

Events as Data Units

Events are structured key-value pairs with timestamps, forming the basic data units in Kafka.

Role of Topics

Topics organize events into logical categories or feeds for efficient data stream management.

Producers Generating Events

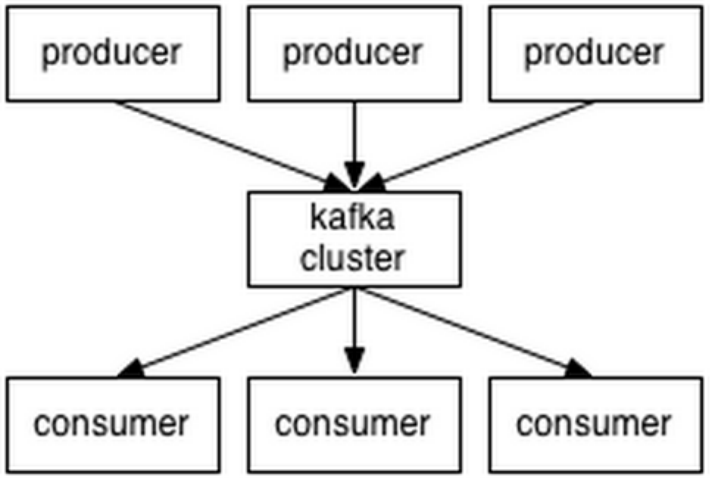
Producers are applications that send events to topics, similar to journalists submitting articles.

Consumers Processing Data

Consumers subscribe to topics and process events independently, tracking progress with offsets.

A first look

- The who is who
  - **Producers** write data to **brokers**.
  - **Consumers** read data from **brokers**.
  - All this is distributed.
- The data
  - Data is stored in **topics**.
  - **Topics** are split into **partitions**, which are **replicated**.



SCALABILITY & DURABILITY: PARTITIONS, BROKERS, REPLICATION

Clustered Brokers and Topics

Kafka clusters consist of multiple brokers managing data topics to enable scalable and distributed data handling.

Partitions for Scalability

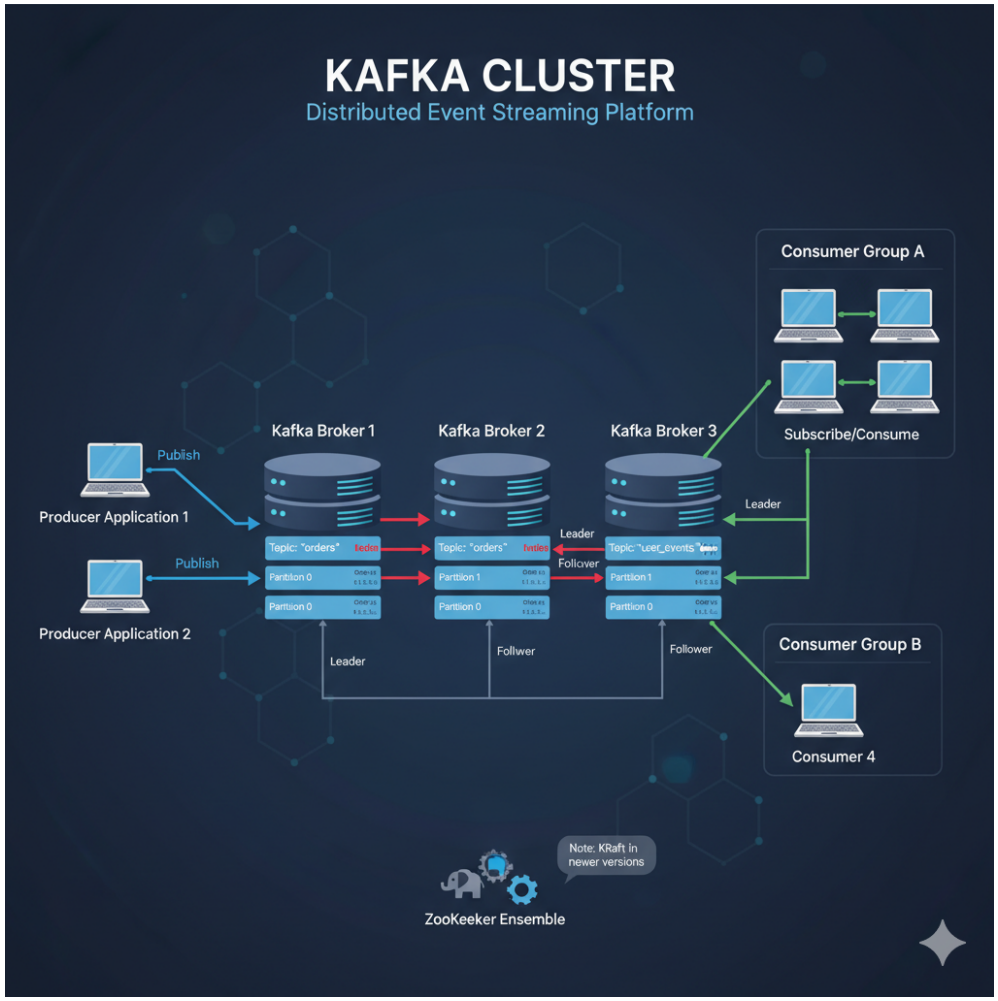
Topics are divided into partitions, allowing parallel processing and improved throughput via simultaneous reads and writes.

Replication for Durability

Partitions are replicated across brokers with leader and follower replicas to ensure fault tolerance and data integrity.

Automatic Failover

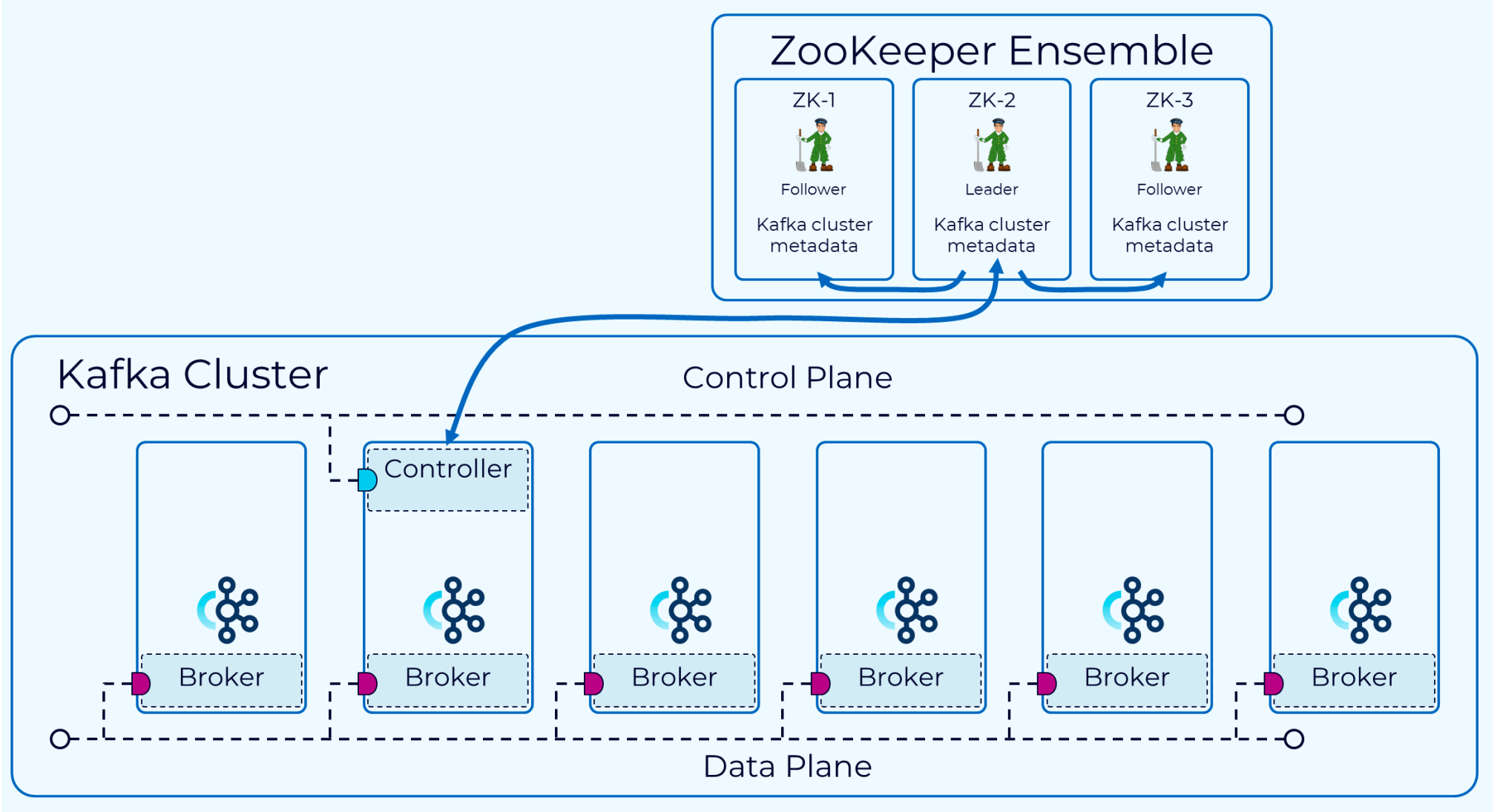
Kafka promotes follower replicas automatically if a leader fails, ensuring continuous service availability.



- Producers send data (records) to specific topics on a Kafka broker.
- The broker appends these records to a partition within the topic.
- Consumers read records from partitions they are subscribed to.
- The distributed nature with multiple brokers and replicated partitions ensures that the system remains available and data is not lost even if some brokers fail

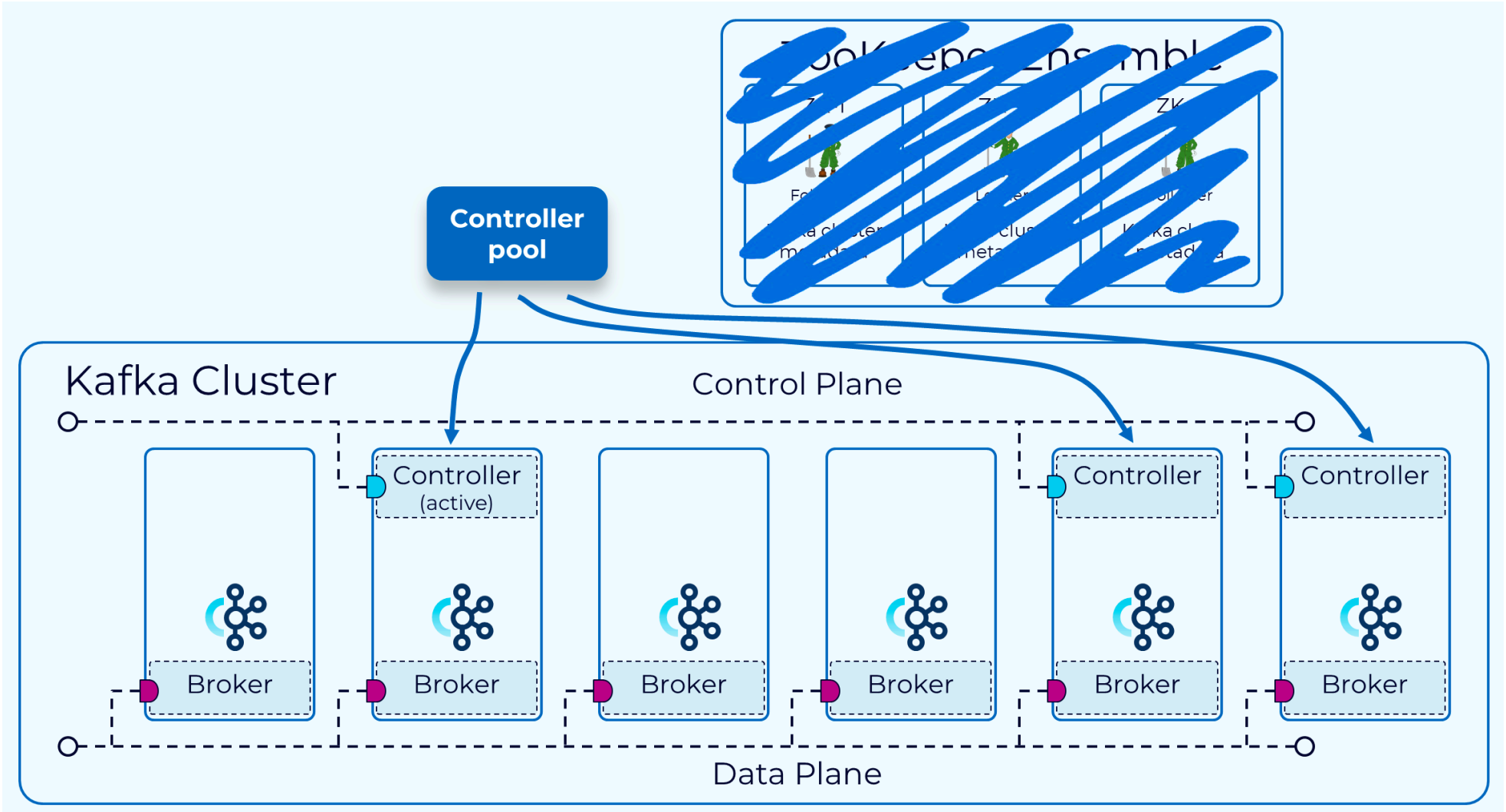
KAFKA CLUSTER

Kluster överblick



Kluster överblick

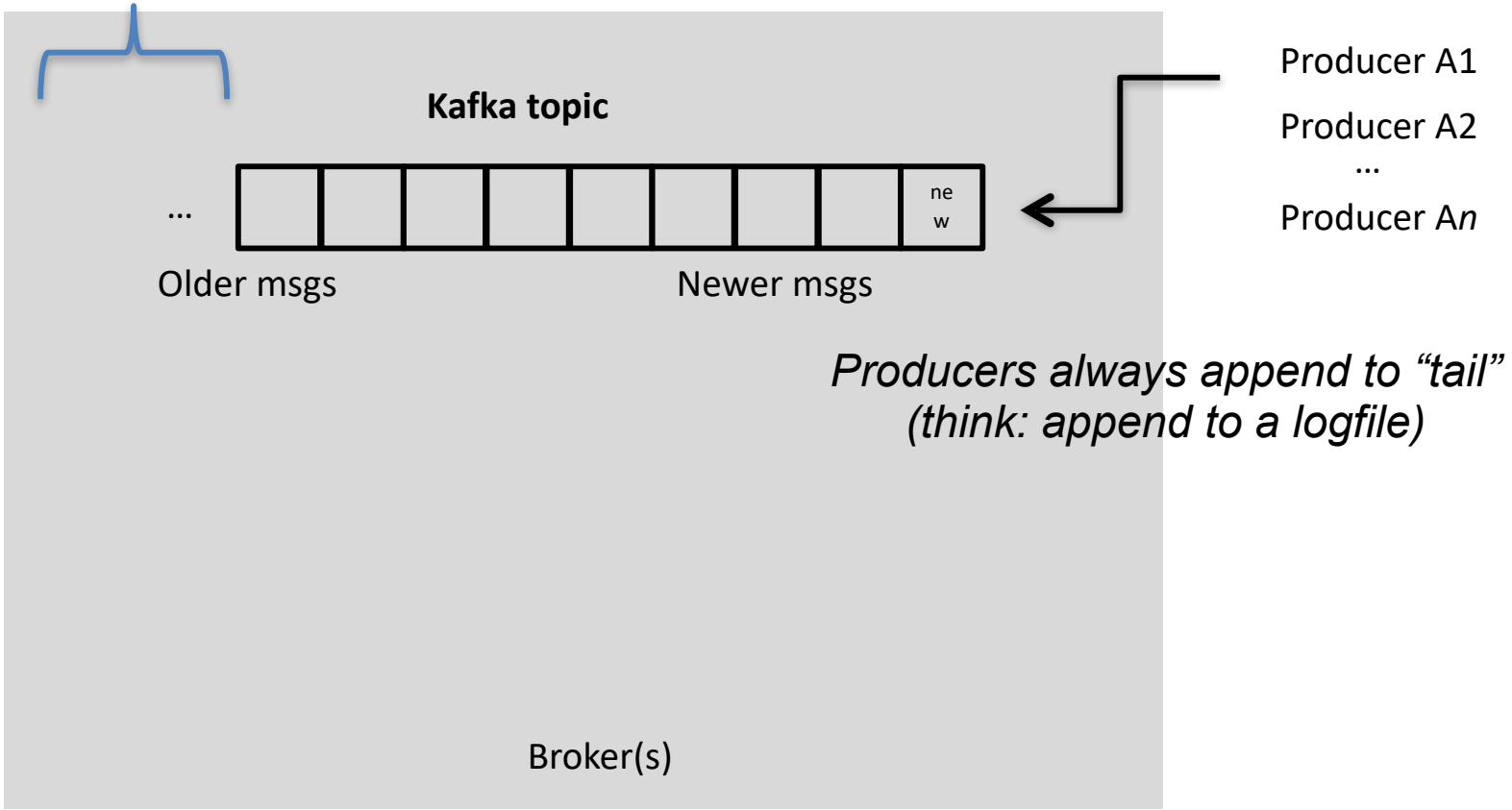
- KRaft Mode



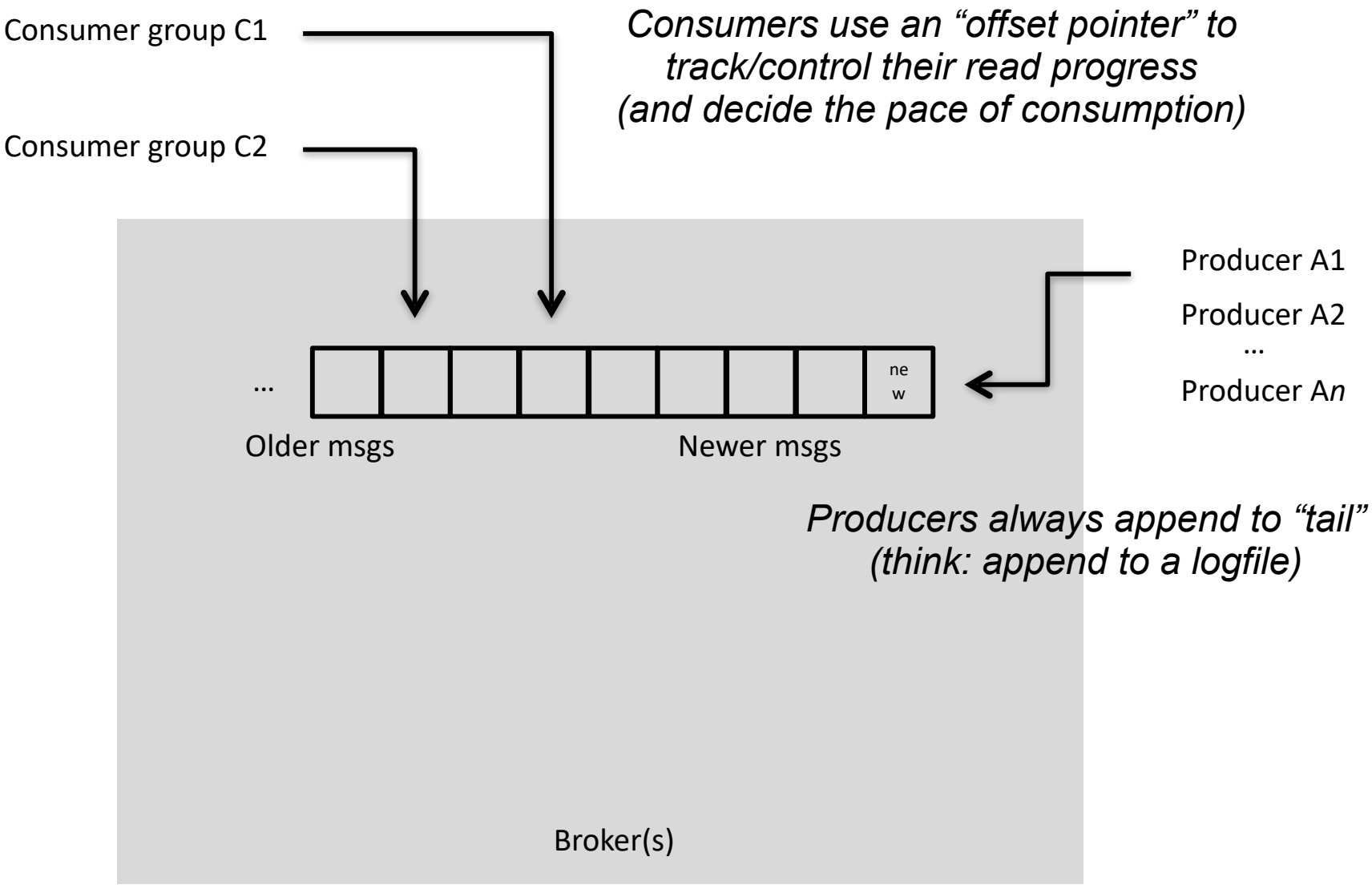
Topics

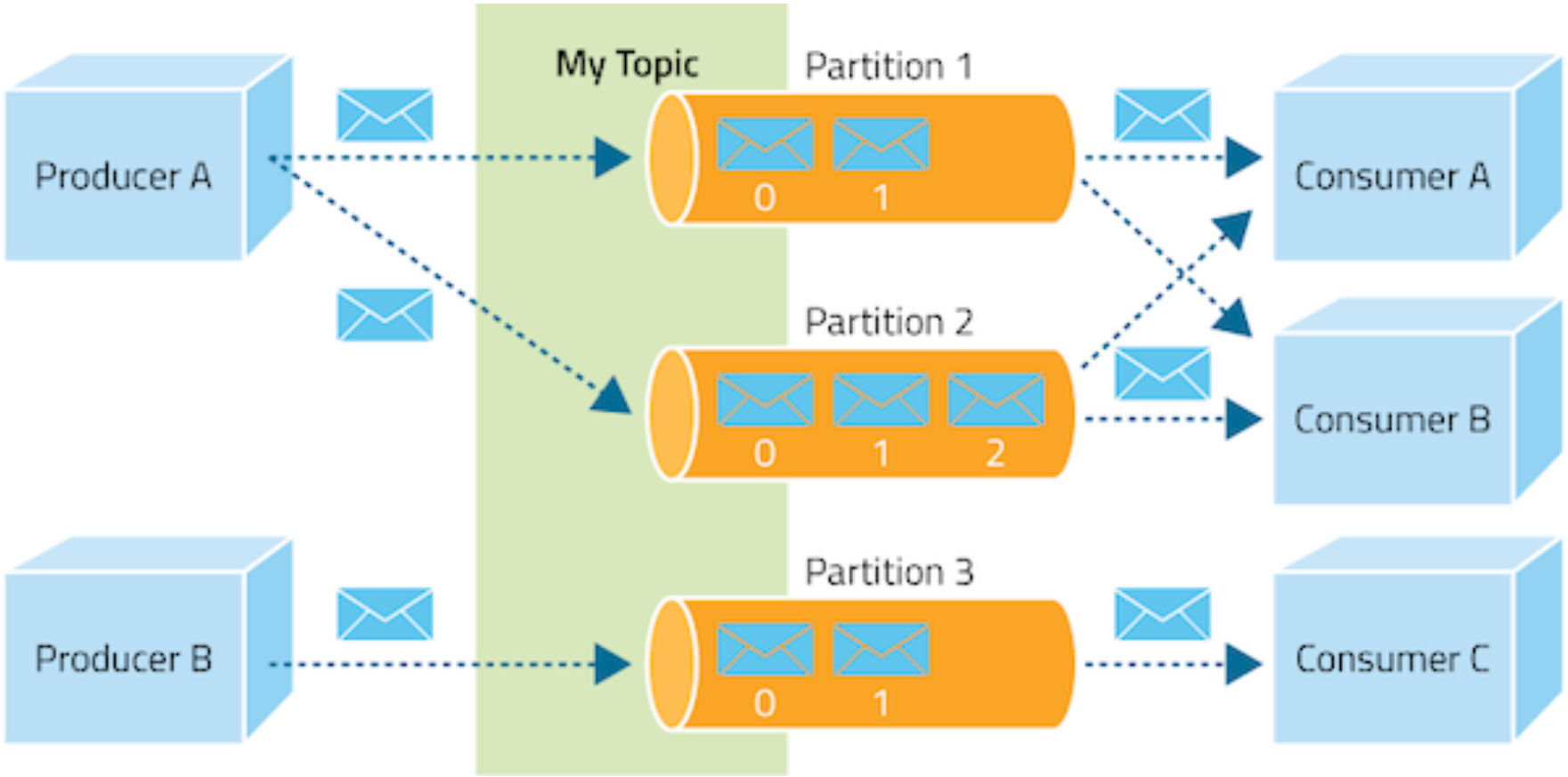
- **Topic:** feed name to which messages are published
  - Example: “access.events”

Kafka prunes “head” based on age or max size or “key”

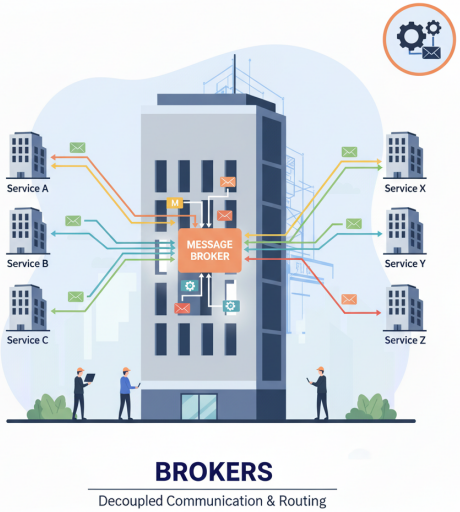


Topics


Consumer group C1
Consumer group C2
Consumers use an "offset pointer" to track/control their read progress (and decide the pace of consumption)
Producer A1
Producer A2
…
Producer An
…
new
Older msgs
Newer msgs
Producers always append to "tail" (think: append to a logfile)
Broker(s)


My Topic
Partition 1
Partition 2
Partition 3
Producer A
Producer B
Consumer A
Consumer B
Consumer C
0
1
0
1
2
0
1

Hur ska vi skriva och organisera koden



Code conventions

```
def loadSchemas():
  schemaTuples, badSchemaFiles = loadAllJsonObjects("schemas")
  schemas = {}
  for path, fileName, o in schemaTuples:
    schemas[fileName[:-5]] = o
  return schemas, badSchemaFiles
```

Syntax and style

```
def load_schemas():
    schema_tuples, bad_schema_files = load_all_json_objects("schemas")
    schemas = {}
    for path, file_name, o in schema_tuples:
        schemas[file_name[:-5]] = o
    return schemas, bad_schema_files
```

```
AbstractOption is a raw type. References to generic type AbstractOption<T> should be parameterized
AbstractOption is a raw type. References to generic type AbstractOption<T> should be parameterized
ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized
ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized
ArrayObjectQueue is a raw type. References to generic type ArrayObjectQueue<E> should be parameterized
```

Compiler warnings

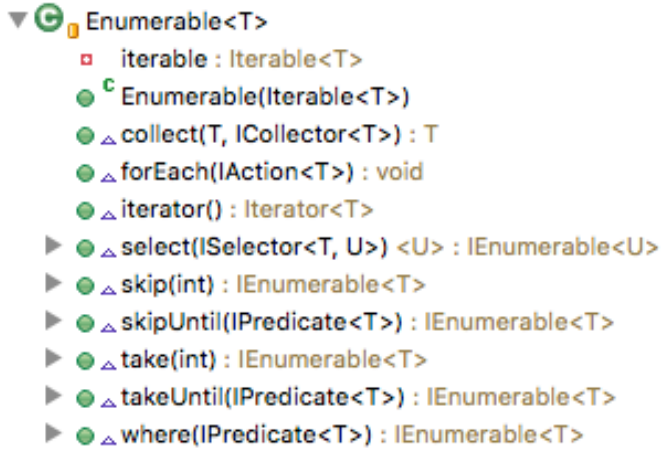
```
int[] a = new int[] {1,2,3};
int sum=0;
for (int i : a) {
    sum+=a[i];
}

Arrays.stream(a).sum();
```

Language constructs

Design principles

```
/** Constructor of the Cell class.
 *  It is the only way to set the name of the cell.
 *  @param name the name of the cell
 */
public Cell(String name) {
    this.name = name;
    formula = "";
    observers=new LinkedList<CellObserver>();
    anal = Analyzer.getInstance();
    state = EquationState.getInstance();
    astN = new ASTValue(0.0);
```

Coupling



Cohesion

Software architecture

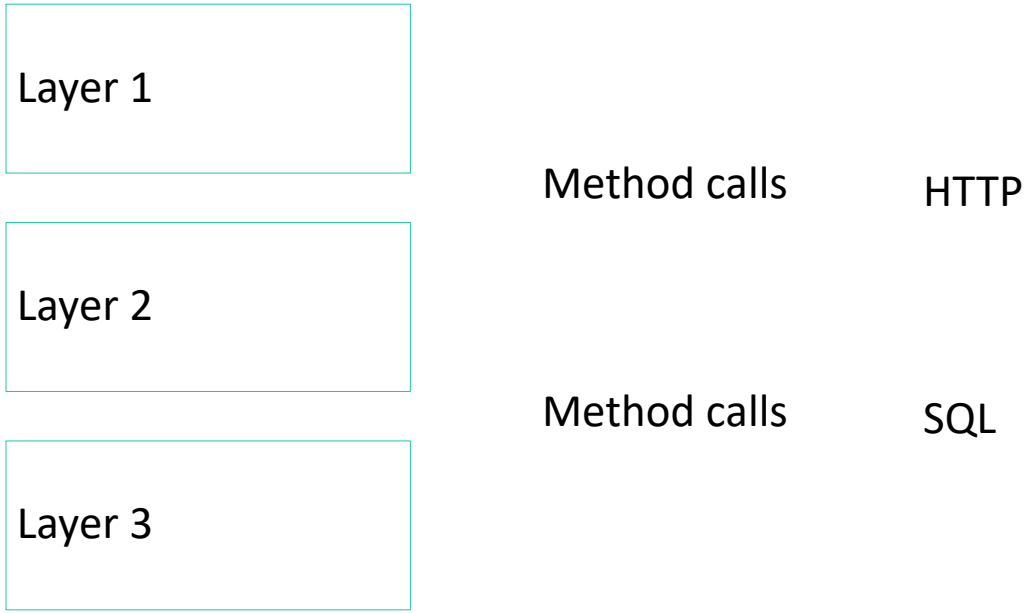
- ”*The set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both*” - Documenting Software Architectures: Views and Beyond (2nd Edition), Clements et al, Addison-Wesley, 2010
- ”*the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution*” - ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems
- ”*That peculiar consistency and homogeneity of Design that would give rise to ease of use and reuse, which is wished for by the Developer who calls himself "The Software Architect.*”” - John Carter (Software Technologist, Tait Electronics, Christchurch, New Zealand)

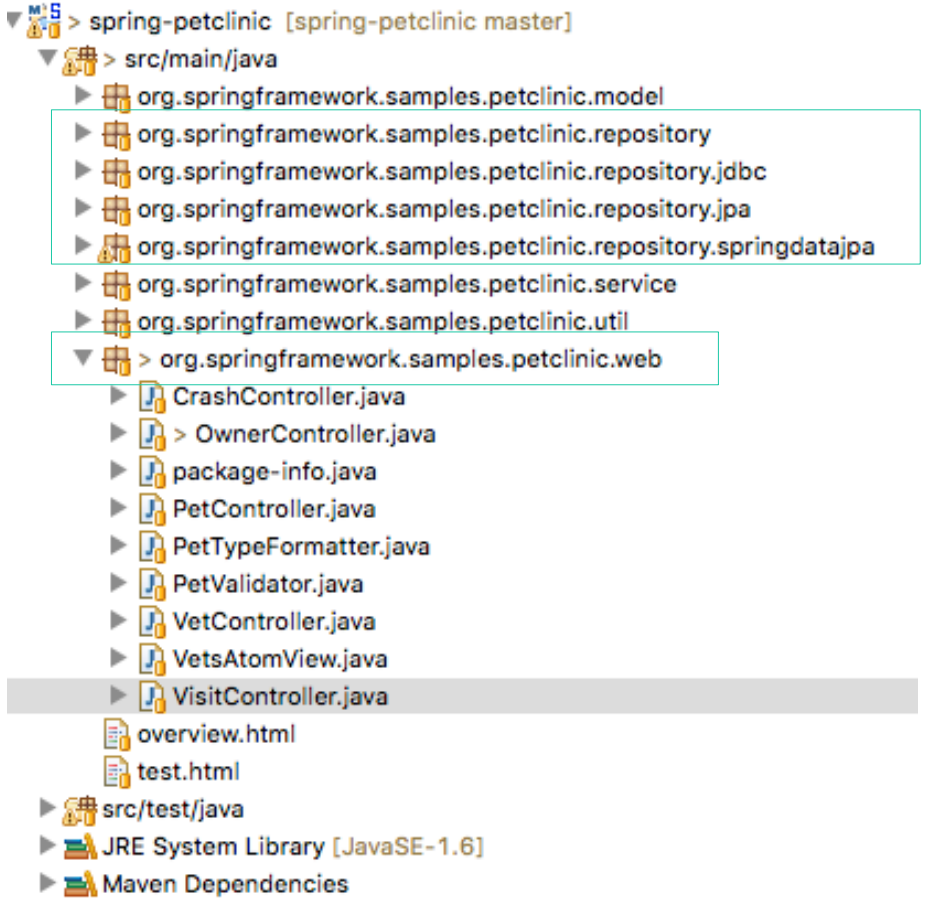
https://www.sei.cmu.edu/architecture/start/glossary/definition-form.cfm

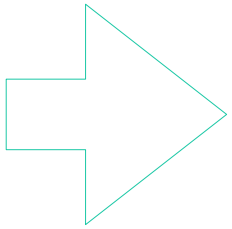
Software architectures

- Flexibility - estimated cost of making changes to individual parts
- Testability - estimated cost of devising tests for parts or the whole
- Scalability/Performance - estimated cost of changing the system to accommodate higher loads
- Cost of development - estimated cost of developing the common core
- Cost of deployment - estimated cost of running the system

Layered architecture




> spring-petclinic [spring-petclinic master]
> src/main/java
org.springframework.samples.petclinic.model
org.springframework.samples.petclinic.repository
org.springframework.samples.petclinic.repository.jdbc
org.springframework.samples.petclinic.repository.jpa
org.springframework.samples.petclinic.repository.springdatajpa
org.springframework.samples.petclinic.service
org.springframework.samples.petclinic.util
> org.springframework.samples.petclinic.web
CrashController.java
> OwnerController.java
package-info.java
PetController.java
PetTypeFormatter.java
PetValidator.java
VetController.java
VetsAtomView.java
VisitController.java
overview.html
test.html
src/test/java
JRE System Library [JavaSE-1.6]
Maven Dependencies



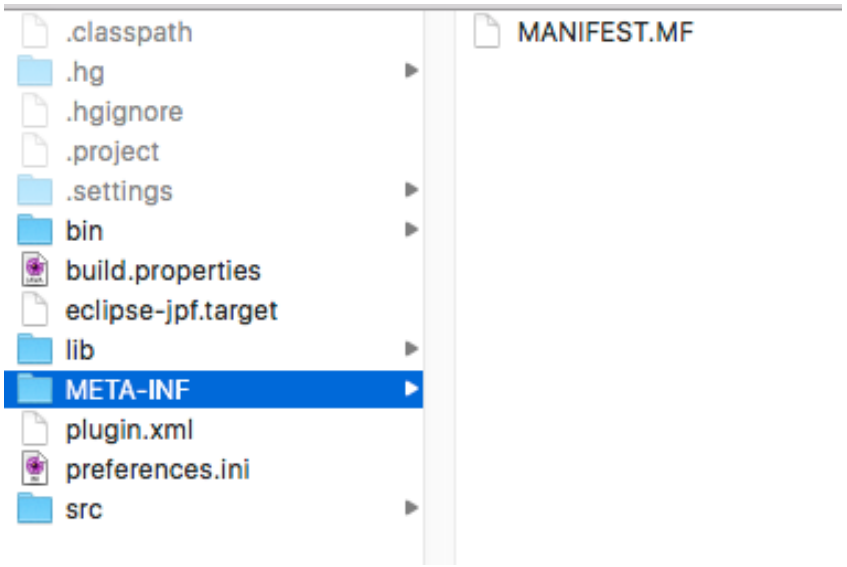
```java
public class VisitController {

    private final ClinicService clinicService;


    public VisitController(ClinicService clinicService) {

    public void setAllowedFields(WebDataBinder dataBinder) {

     *  Called before each and every @RequestMapping annotated method.
    @ModelAttribute("visit")
    public Visit loadPetWithVisit(@PathVariable("petId") int petId) {
        Pet pet = this.clinicService.findPetById(petId);
        Visit visit = new Visit();
        pet.addVisit(visit);
        return visit;
    }

    // Spring MVC calls method loadPetWithVisit(...) before initNewVisitForm is called
    @RequestMapping(value = "/owners/*/pets/{petId}/visits/new", method = RequestMethod.GET)
    public String initNewVisitForm(@PathVariable("petId") int petId, Map<String, Object> model) {
        return "pets/createOrUpdateVisitForm";
    }

    // Spring MVC calls method loadPetWithVisit(...) before processNewVisitForm is called
    @RequestMapping(value = "/owners/{ownerId}/pets/{petId}/visits/new", method = RequestMethod.POST)
    public String processNewVisitForm(@Valid Visit visit, BindingResult result) {
        if (result.hasErrors()) {
            return "pets/createOrUpdateVisitForm";
        } else {
            this.clinicService.saveVisit(visit);
            return "redirect:/owners/{ownerId}";
        }
    }
```

Analysis

- Testability?
- Scalability?
- Flexibility?
- Ease of development?
- Ease of deployment?

Microkernel architecture

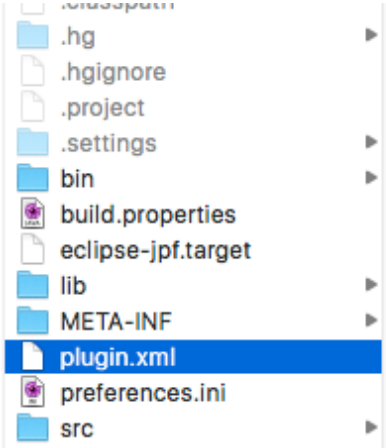
- Eclipse - Equinox - OSGi
- (Atom - Node)



Manifest


```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Eclipse-jpf
Bundle-SymbolicName: eclipse-jpf;singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-ClassPath: lib/antlr-runtime-3.4.jar,
 lib/ST-4.0.4.jar,
 lib/jpf-template.jar,
 lib/RunJPF.jar,
 .
Bundle-Activator: eclipse_jpf.Activator
Export-Package: eclipse_jpf,
 gov.nasa.runjpf,
 gov.nasa.runjpf.options,
 gov.nasa.runjpf.util,
 gov.nasa.runjpf.wizard
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.ui,
 org.eclipse.core.runtime
Bundle-RequiredExecutionEnvironment: JavaSE-1.8
Bundle-ActivationPolicy: lazy
Import-Package: org.eclipse.core.filesystem,
 org.eclipse.core.resources,
 org.eclipse.core.runtime;version="3.4.0",
 org.eclipse.core.runtime.jobs,
 org.eclipse.core.runtime.preferences;version="3.3.0",
 org.eclipse.jdt.core,
 org.eclipse.jface.action,
 org.eclipse.jface.dialogs,
 org.eclipse.jface.preference,
 org.eclipse.jface.text,
 org.eclipse.jface.viewers,
 org.eclipse.jface.wizard,
 org.eclipse.osgi.util;version="1.1.0",
 org.eclipse.swt,
 org.eclipse.swt.events,
 org.eclipse.swt.graphics,
 org.eclipse.swt.layout,
 org.eclipse.swt.widgets,
 org.eclipse.ui,
 org.eclipse.ui.console,
 org.eclipse.ui.dialogs,
 org.eclipse.ui.ide,
```

Extension points




```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension
         point="org.eclipse.core.contenttype.contentTypes">
      <content-type
            base-type="org.eclipse.core.runtime.properties"
            file-extensions="jpf"
            id="com.javapathfinder.vjp.configContentType"
            name="JPF Config File"
            priority="normal">
      </content-type>
   </extension>
   <extension
         point="org.eclipse.ui.popupMenus">
      <objectContribution
            adaptable="false"
            id="runjpf.jpfConfigLaunch"
            nameFilter="*.jpf"
            objectClass="org.eclipse.core.resources.IFile">
         <action
               class="gov.nasa.runjpf.VerifyActionDelegate"
               enablesFor="1"
               id="RunJPF.verifyaction"
               label="Verify..."
               menubarPath="additions"
               tooltip="Execute this property file as a JPF configuration">
         </action>
      </objectContribution>
   </extension>
   <extension
         point="org.eclipse.ui.preferencePages">
      <page
            class="gov.nasa.runjpf.options.Preferences"
            id="eclipse-jpf.preferences"
            name="JPF Preferences">
      </page>
   </extension>
        <extension
              point="org.eclipse.core.runtime.preferences">
           <initializer class="gov.nasa.runjpf.options.DefaultPreferences" />
        </extension>
<extension
      point="org.eclipse.ui.newWizards">
   <category
         id="eclipse-jpf.category.wizards"
```

LINKÖPINGS UNIVERSITET

Lifecycle


```java
public class EclipseJPF extends AbstractUIPlugin {

    /**
     *  The plug-in ID
     */
    public static final String PLUGIN_ID = "RunJPF";

    // The shared instance
    private static EclipseJPF plugin;

    public EclipseJPF() {
    }

    /*
     * (non-Javadoc)
     * @see org.eclipse.ui.plugin.AbstractUIPlugin#start(org.osgi.framework.BundleContext)
     */
    public void start(BundleContext context) throws Exception {
        super.start(context);
        plugin = this;
    }

    /*
     * (non-Javadoc)
     * @see org.eclipse.ui.plugin.AbstractUIPlugin#stop(org.osgi.framework.BundleContext)
     */
    public void stop(BundleContext context) throws Exception {
        plugin = null;
        super.stop(context);
    }
```

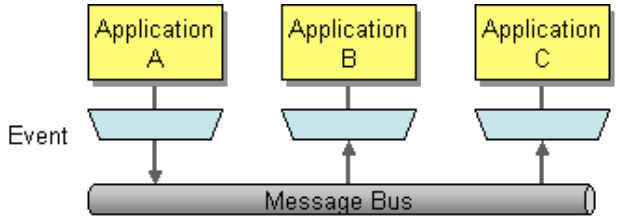
Plugin

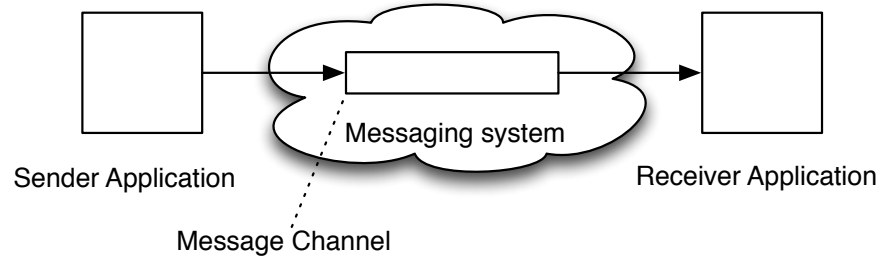
```
public class DefaultPreferences extends AbstractPreferenceInitializer {

  @Override
  public void initializeDefaultPreferences(){
    IPreferenceStore store = EclipseJPF.getDefault().getPreferenceStore();
    store.setDefault(EclipseJPFLauncher.SITE_PROPERTIES_PATH,
EclipseJPFLauncher.DEFAULT_SITE_PROPERTIES_PATH);
    store.setDefault(EclipseJPFLauncher.PORT, EclipseJPFLauncher.DEFAULT_PORT);
  }
}
```

Analysis

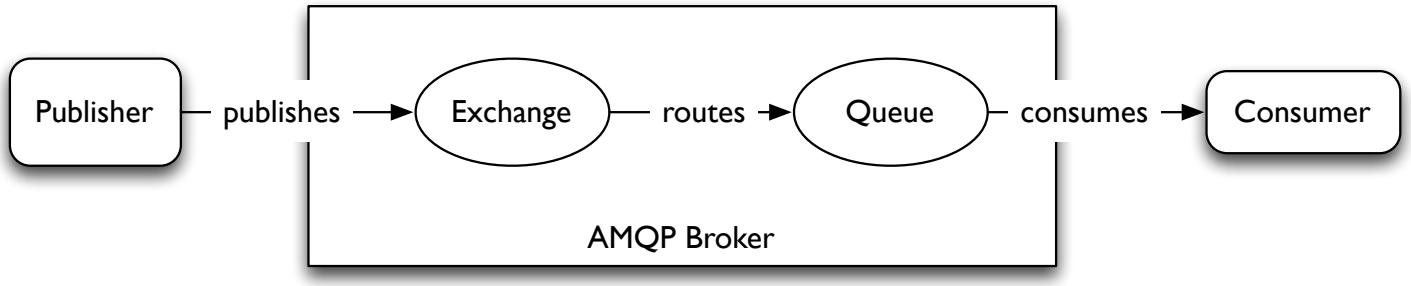
- Testability?
- Scalability?
- Flexibility?
- Ease of development?
- Ease of deployment?

Message bus architectures

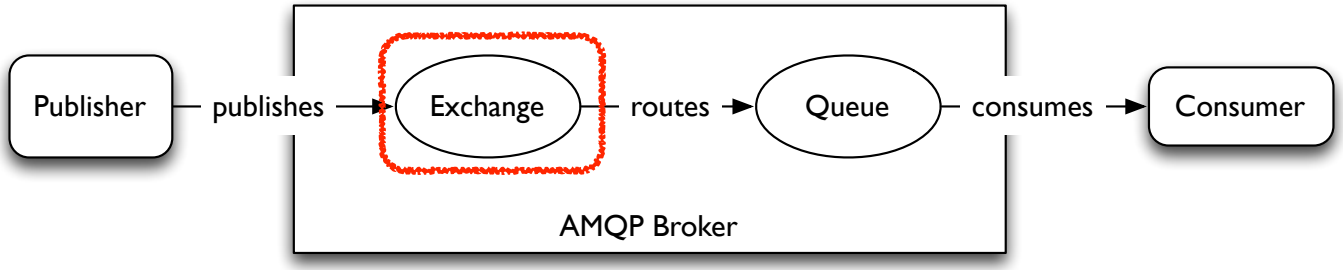




AMQP: Advanced Message Queueing Protocol



https://www.rabbitmq.com/tutorials/amqp-concepts.html


Publisher
publishes
Exchange
routes
Queue
consumes
Consumer
AMQP Broker

Analysis

- Testability?
- Scalability?
- Flexibility?
- Ease of development?
- Ease of deployment?

Demo

- Hela Stacken+kafka

Linköpings universitet

www.liu.se