Konzepte objektorientierter Programmierung

Prof. Dr. Peter Müller

Werner Dietl

Software Component Technology

Exercises 7: Aliasing

Wintersemester 05/06

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Homework – Exercise 1

```
class Motor {
   boolean isOK() { return true; }
   void start() { /*...*/ }
   ...
}
class Wheel {
   void deflate() { /*...*/ }
   ...
}
class MotorTrouble extends Exception {
   public Motor motor;
   public MotorTrouble( Motor m ) {
     motor = m;
   }
}
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exercise 1 – The Invariant

```
public class Car {


  /*@ invariant engine.isRunning() ==>
    @ (\forall int i;
    @      i >= 0 && i < wheels.length;
    @          wheels[i] != null &&
    @          wheels[i].isOK())
    @*/
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Problem 1 – Public Fields

```
public class Car {
  public Motor engine;
  public Wheel[] wheels;
```

```
Car c = new Car(...);
c.start();
// remove a wheel -> breaks invariant
c.wheels[0] = null;
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Solution for Problem 1

- Use proper Information Hiding
- Make all fields private or protected

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Problem 2 – Capturing

```
public class Car {
  public Car( Motor m, Wheel[] w ) {
     engine = m;
     wheels = w;
  }
```

```
Motor m = new Motor();
Car c = new Car(m, w);
c.start();
// stop the motor, car still thinks it runs
m.stop();
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Solution for Problem 2 – Capturing

- Never directly store parameters as internal state
- Clone the given objects
- For arrays you need a deep copy, otherwise the array elements might still be aliased

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Problem 3 – Leaking through Return Value

```
public Motor getMotor() {
     return engine; }

public Wheel[] getWheels() {
     return wheels; }
```

```
Car c = new Car(m, w);
c.start();
// change wheel while driving
c.getWheels()[0] = new Wheel();
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Solution for Problem 3 – Leaking

- Never return a reference to the internal state to the outside
- Always clone the objects
- Again for arrays you need a deep copy

Problem 4 – Leaking through Exception

```
public void start() throws MotorTrouble {
  if( engine == null || !engine.isOK() ) {
    throw new MotorTrouble( engine );
  } else { engine.start(); }
}
```

```
try { ... }
catch( MotorTrouble mt ) {
  mt.motor.reset();
}
```

Solution for Problem 4 – Exceptions

- Exceptions can give access to internal state
- Examine what information you really want to share
- Maybe only pass a String with this information

Invariants for Java (Simple Solution)

- Assumption: The invariants of object X may only refer to **private attributes** of X

- For each invariant, we have to show
  - That all exported methods **and constructors of class T** preserve the invariants **of all objects of T and T's subclasses**
  - That all constructors **in addition** establish the invariants of the new object

Invariants can refer to all attributes

```
public class Super {

  private int f;

}
```

```
public class Sub extends Super {

  /*@ invariant f > 0; @*/

}
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Homework – Exercise 2 a

```
public class A {
  /* default */ int f;

  /*@ invariant f > 0; @*/
}
```

```
public class B {
  private A myA;

  void violate() { myA.f = -10; }
}
```

Invariants for Java (Extended Solution)

- Assumption: The invariants of object X may refer to **private and default attributes** of X

- For each invariant, we have to show
  - That all exported methods **and constructors of class T** preserve the invariants **of all objects of all classes in T's package and all subclasses of classes in T's package**
  - That all constructors **in addition** establish the invariants of the new object

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Invariants of T objects

```
public class T {
  /* default */ int f;

  /*@ invariant f > 0; @*/

  void violate() { f = -10; }
}
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Invariants of objects in T's package

```
public class A {
  /* default */ int f;

  /*@ invariant f > 0; @*/
}
```

```
public class T {
  private A myA;

  void violate() { myA.f = -10; }
}
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Invariants of objects of T's subclasses

```
public class T {
  /* default */ int f;

  void violate() { f = -10; }
}
```

```
public class SubT extends T {

  /*@ invariant f > 0; @*/

}
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Subclasses of all classes in T's package

```
public class A {
  /* default */ int f; }

public class SubA extends A
  /*@ invariant f > 0; @*/ }
```

```
public class T {
  private A myA;

  void violate() { myA.f = -10; }
}
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

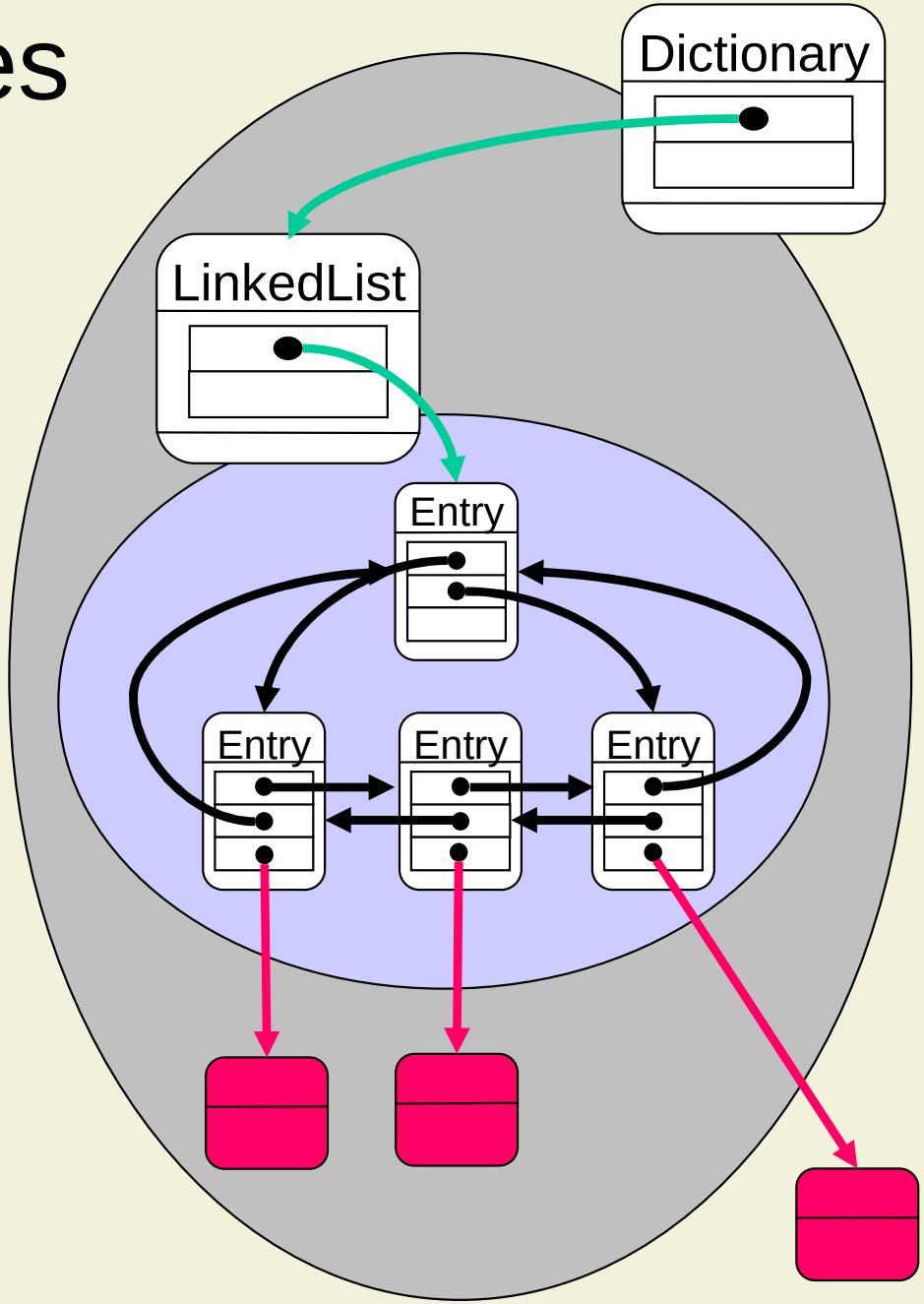
Homework – Exercise 3

- See suggested solution for the exam

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Meaning of Alias Modes

- Alias modes describe the role of an object relative to an interface object
- Informally: References
  - With **default mode** stay in the same context
  - With **rep-mode** go from an interface object into its context
  - With **arg-mode** may go to any context

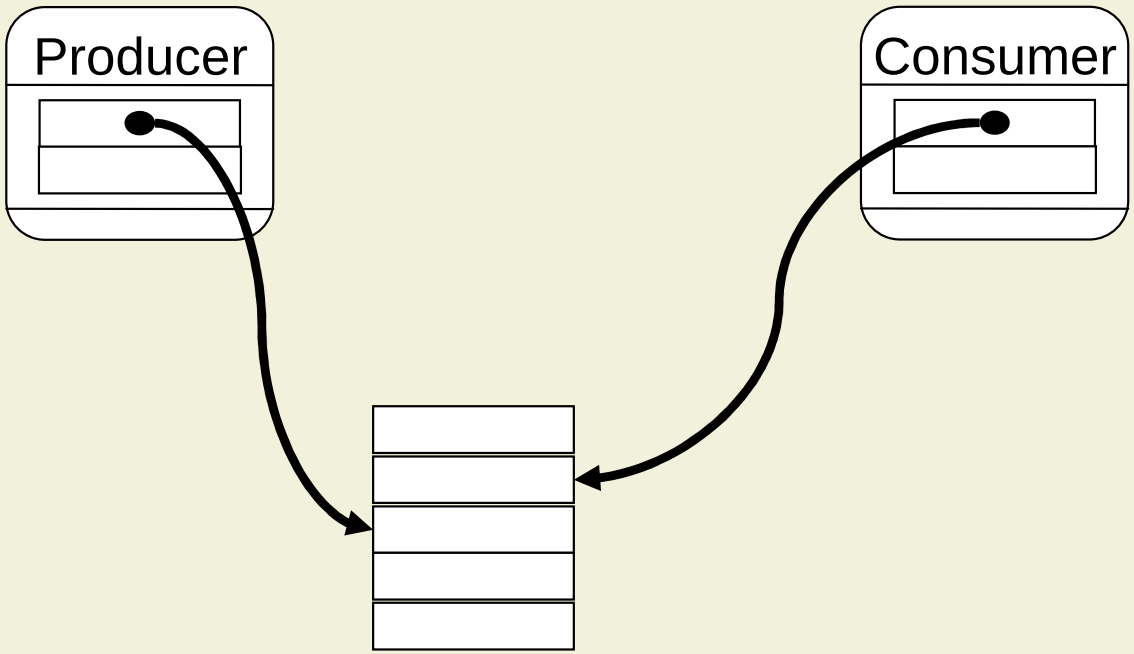


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

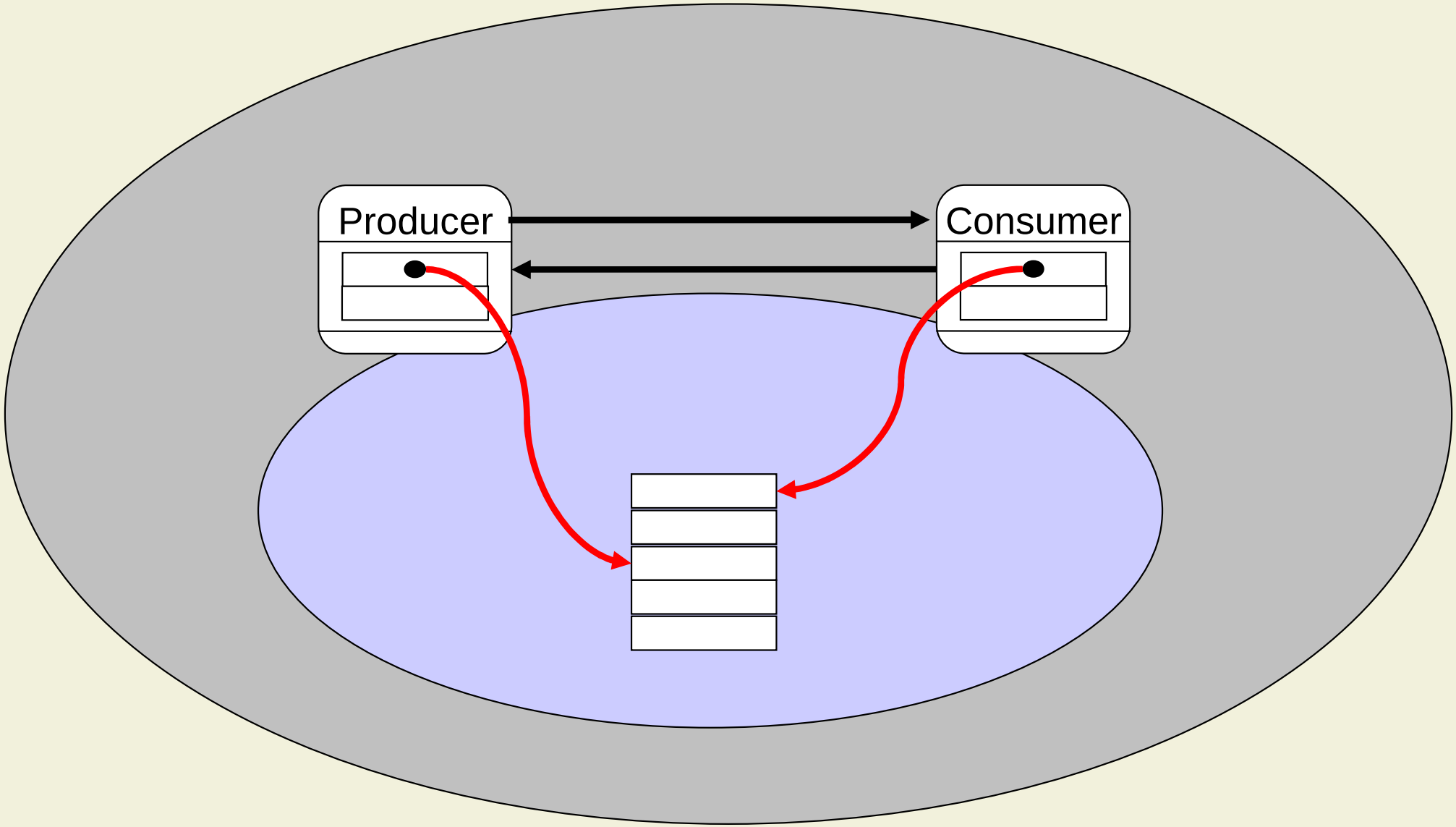
(Simplified) Programming Discipline

- **Rule 1: No Role Confusion**
  - Expression with one alias mode must not be assigned to variables with another mode
- **Rule 2: No Representation Exposure**
  - rep-mode must not occur in an object's interface
  - Methods must not take or return rep-objects
  - Fields with rep-mode may only be accessed on **this**
- **Rule 3: No Argument Dependence**
  - Implementations must not depend on the state of argument objects

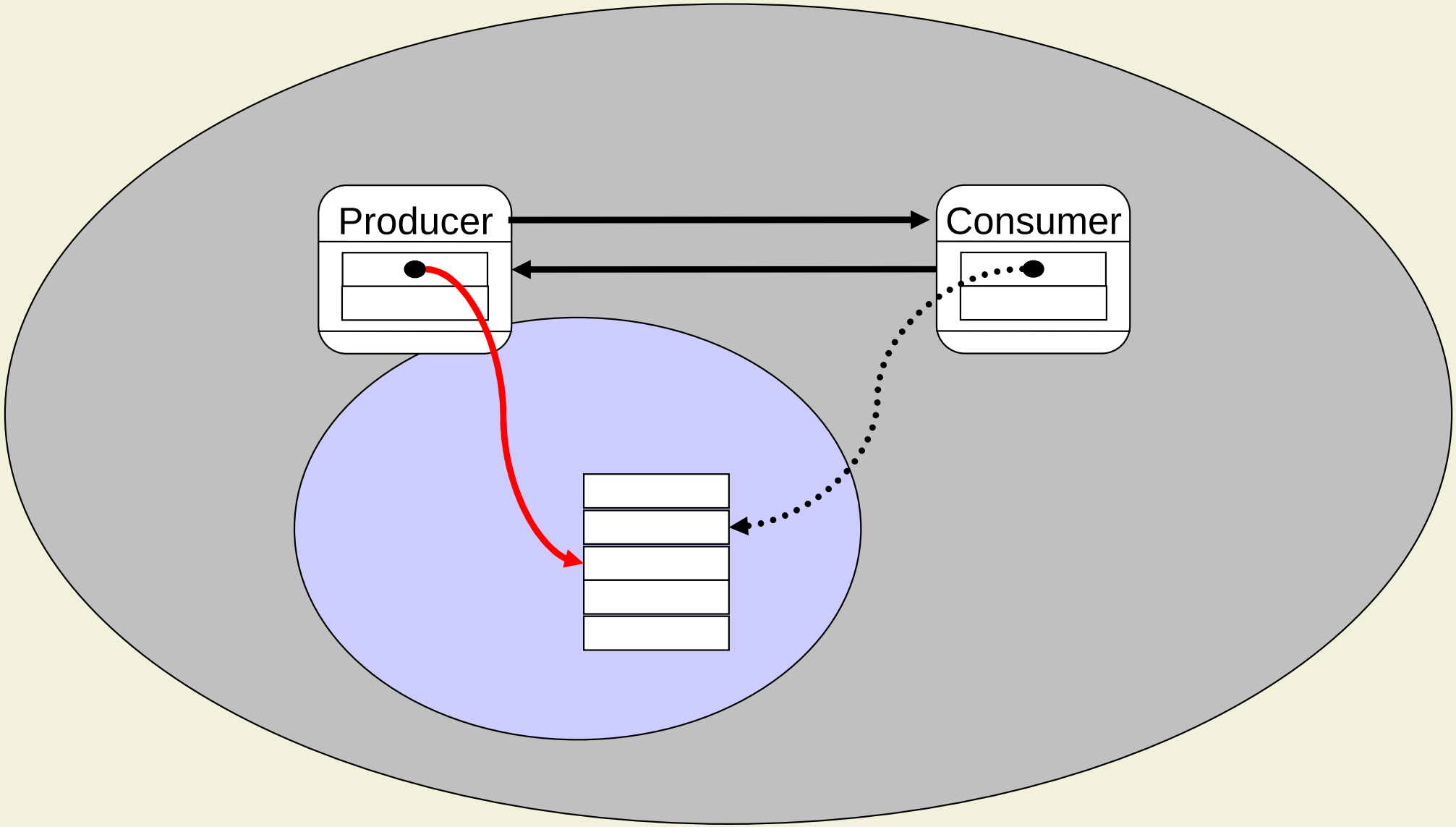
Producer-Consumer Example



Producer-Consumer Example



Producer-Consumer Example



Producer – Original

```
class Producer {
    int[] buf;
    int n;
    Consumer con;

    Producer() {
        buf = new int[10];
    }

    void produce(int x) {
        buf[n] = x;
        n = (n+1) % buf.length;
    }
}
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Producer – Annotated

```
class Producer {
    /*@ rep @*/ int[] buf;
    int n;
    /*@ peer @*/ Consumer con;

    Producer() {
        buf = new /*@ rep @*/ int[10];
    }

    void produce(int x) {
        buf[n] = x;
        n = (n+1) % buf.length;
    }
}
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Consumer – Original

```
class Consumer {
    int[] buf;
    int n;
    Producer pro;

    Consumer(Producer p) {
        buf = p.buf;          pro = p;
        n = buf.length-1;     p.con = this;
    }

    int consume() {
        n = (n+1) % buf.length;
        return buf[n];
    }
}
```

Consumer – Annotated

```
class Consumer {
    /*@ arg @*/ int[] buf;
    int n;
    /*@ peer @*/ Producer pro;

    Consumer(/*@ peer @*/ Producer p) {
        buf = p.buf;          pro = p;
        n = buf.length-1;     p.con = this;
    }

    int consume() {
        n = (n+1) % buf.length;
        return buf[n];
    }
}
```


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich