Das D Framework

Ein Werkzeug für verteiltes Programmieren

Inhalt

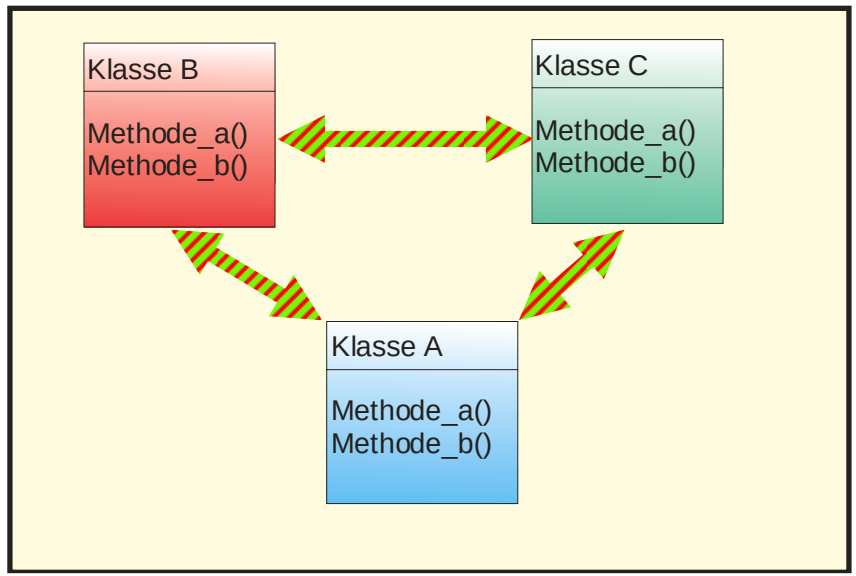
1. Motivation

2. Das D Framework
   1. Thread Synchronisierung
   2. Remote Interactions
   3. Fallstudie

3. Ausblick

Motivation

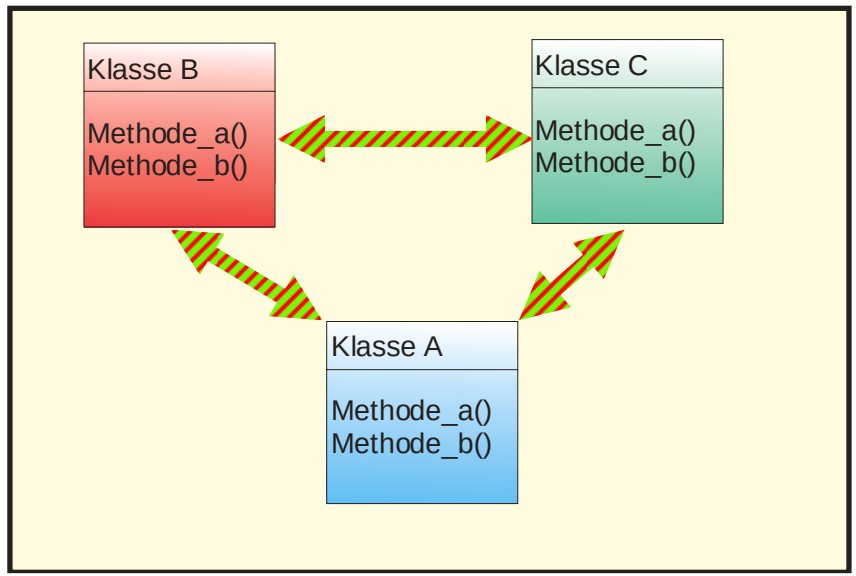
- Single Thread Anwendung



Motivation

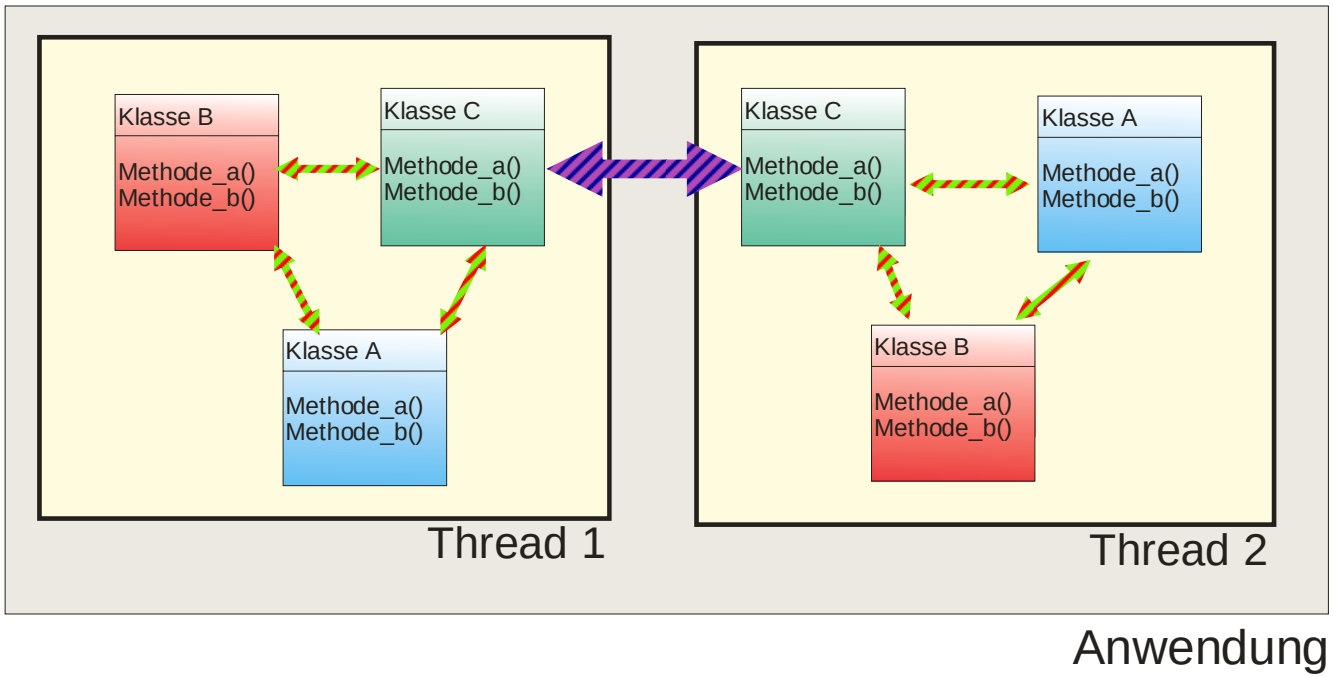
- Single Thread Anwendung

  - keine Nebenläufigkeit
  - keine Synchronisation nötig
  - einfach zu entwickeln



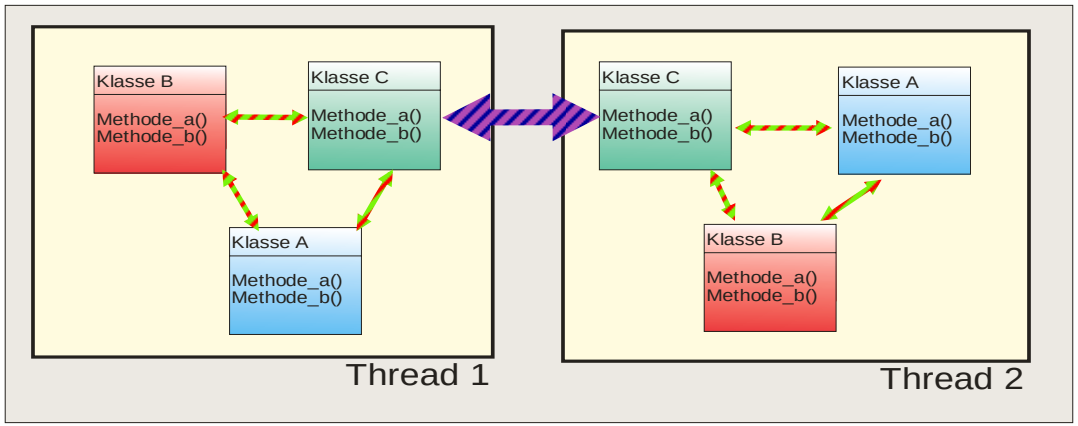
Motivation

- Multi Thread Anwendung



Motivation

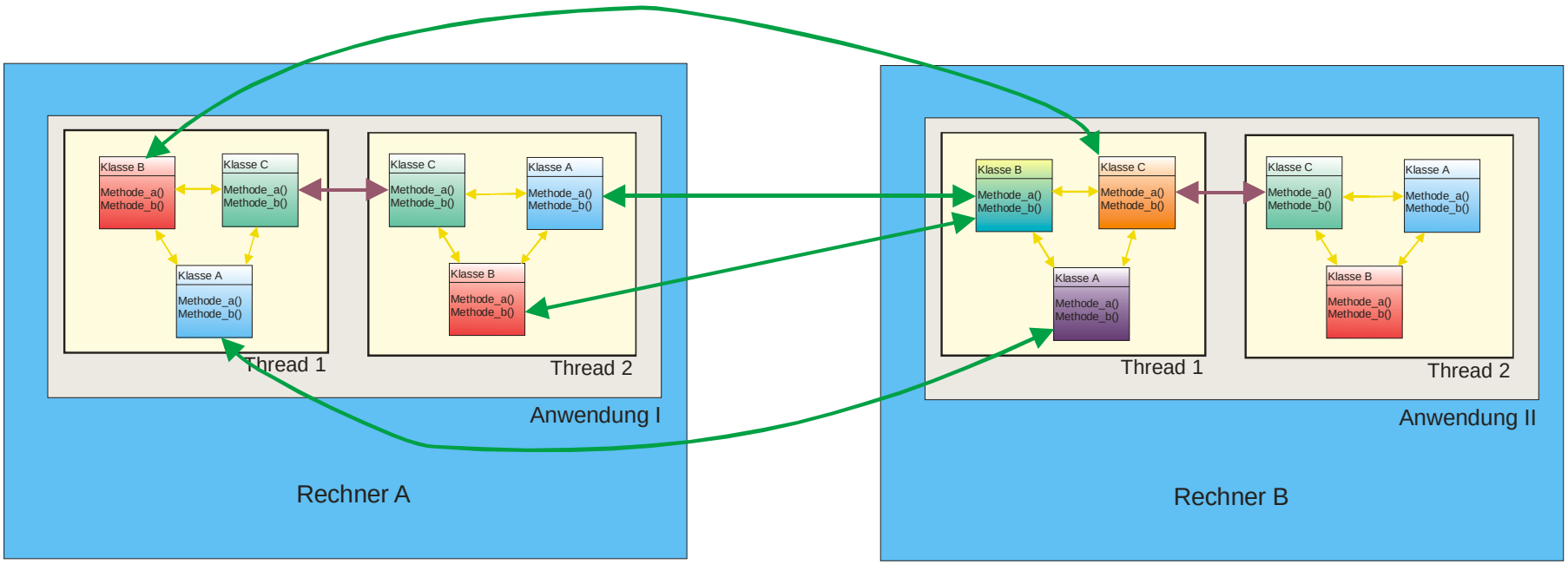
- Multi Thread Anwendung



- Nebenläufigkeiten
- Synchronisation mit Hilfe des Betriebsystems möglich (Semaphoren, …)
- querschneidende Belange, da Synchronisierung an mehreren Stellen notwendig sein kann (z.B. gemeinsam Datenstrukturen)

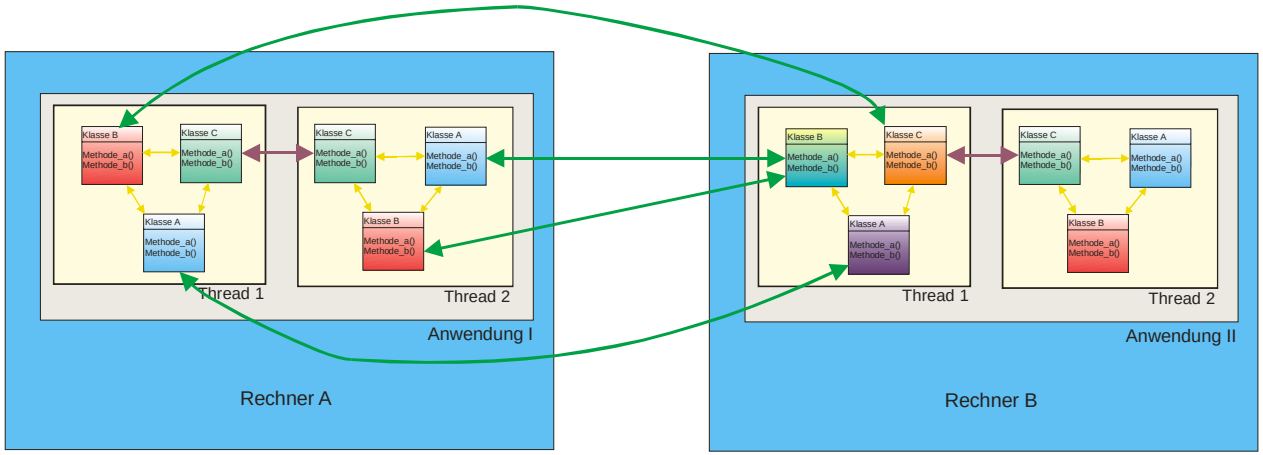
Motivation

- Verteilte Anwendung mit mehreren Threads



Motivation

- Verteilte Anwendung mit mehreren Threads



- Verteilung auf mehrere Rechner und Threads
- Schwer überschaubar – sehr komplex
- Synchronisierung auf Thread- und Rechnerebene
- Prozedurfernaufrufe (RPC) benötigen spezielle Behandlung
- Spezialwissen über verteilte Programmierung nötig

Motivation

Code einer nichtverteilten Anwendung:

```
public class Shape {
        protected double x_= 0.0, y_= 0.0;
        protected double width_=0.0, height_=0.0;
        double get_x() { return x_(); }
        void     set_x(int x) { x_ = x; }
        double get_y() { return y_(); }
        void     set_y(int y) { y_ = y; }
        double get_width(){ return width_(); }
        void      set_width(int w) { width_ = w; }
        double get_height(){ return height_(); }
        void      set_height(int h) { height_ = h;}
        void      adjustLocation() {
                x_ = longCalculation1();
                y_ = longCalculation2();
         }
        void adjustDimensions() {
                width_ = longCalculation3();
                height_ = longCalculation4();
        }
}
```

Motivation

Gleicher Code für verteilten Einsatz:

```
public class Shape {
        protected double x_= 0.0, y_= 0.0;
        protected double width_=0.0, height_=0.0;
        double get_x() { return x_(); }
        void     set_x(int x) { x_ = x; }
        double get_y() { return y_(); }
        void     set_y(int y) { y_ = y; }
        double get_width(){ return width_(); }
        void     set_width(int w) { width_ = w; }
        double get_height(){ return height_(); }
        void     set_height(int h) { height_ = h;}
        void     adjustLocation() {
                x_ = longCalculation1();
                y_ = longCalculation2();
         }
        void adjustDimensions() {
                width_ = longCalculation3();
                height_ = longCalculation4();
        }
}
```

```
public class Shape implements ShapeI {
  protected AdjustableLocation loc;
  protected AdjustableDimension dim;
  public Shape() {
    loc = new AdjustableLocation(0, 0);
    dim = new AdjustableDimension(0, 0);
  }
  double get_x() throws RemoteException { return loc.x(); }
  void   set_x(int x) throws RemoteException { loc.set_x(); }
  double get_y() throws RemoteException { return loc.y(); }
  void   set_y(int y) throws RemoteException { loc.set_y(); }
  double get_width() throws RemoteException { return dim.width(); }
  void   set_width(int w) throws RemoteException { dim.set_w(); }
  double get_height() throws RemoteException { return dim.height(); }
  void   set_height(int h) throws RemoteException { dim.set_h(); }
  void   adjustLocation() throws RemoteException {
    loc.adjust();
  }
  void   adjustDimensions() throws RemoteException {
    dim.adjust();
  }
}

class AdjustableLocation {
  protected double x_, y_;
  public AdjustableLocation(double x, double y) {
    x_ = x; y_ = y;
  }
  synchronized double get_x() { return x_; }
  synchronized void   set_x(int x) {x_ = x;}
  synchronized double get_y() { return y_; }
  synchronized void   set_y(int y) {y_ = y;}
  synchronized void adjust() {
    x_ = longCalculation1();
    y_ = longCalculation2();
  }
}

class AdjustableDimension {
  protected double width_=0.0, height_=0.0;
  public AdjustableDimension(double h, double w) {
    height_ = h; width_ = w;
  }
  synchronized double get_width() { return width_; }
  synchronized void   set_w(int w) {width_ = w;}
  synchronized double get_height() { return height_; }
  synchronized void   set_h(int h) {height_ = h;}
  synchronized void adjust() {
    width_ = longCalculation3();
    height_ = longCalculation4();
  }
}
interface ShapeI extends Remote {
  double get_x() throws RemoteException ;
  void   set_x(int x) throws RemoteException ;
  double get_y() throws RemoteException ;
  void   set_y(int y) throws RemoteException ;
  double get_width() throws RemoteException ;
  void   set_width(int w) throws RemoteException ;
  double get_height() throws RemoteException ;
  void   set_height(int h) throws RemoteException ;
  void   adjustLocation() throws RemoteException ;
  void   adjustDimensions() throws RemoteException ;
}
```

Das D Framework

D Framework

- Besteht aus 3 Teilen:
  - „Coordinators“
  - „Portals“
  - Komponentensprache

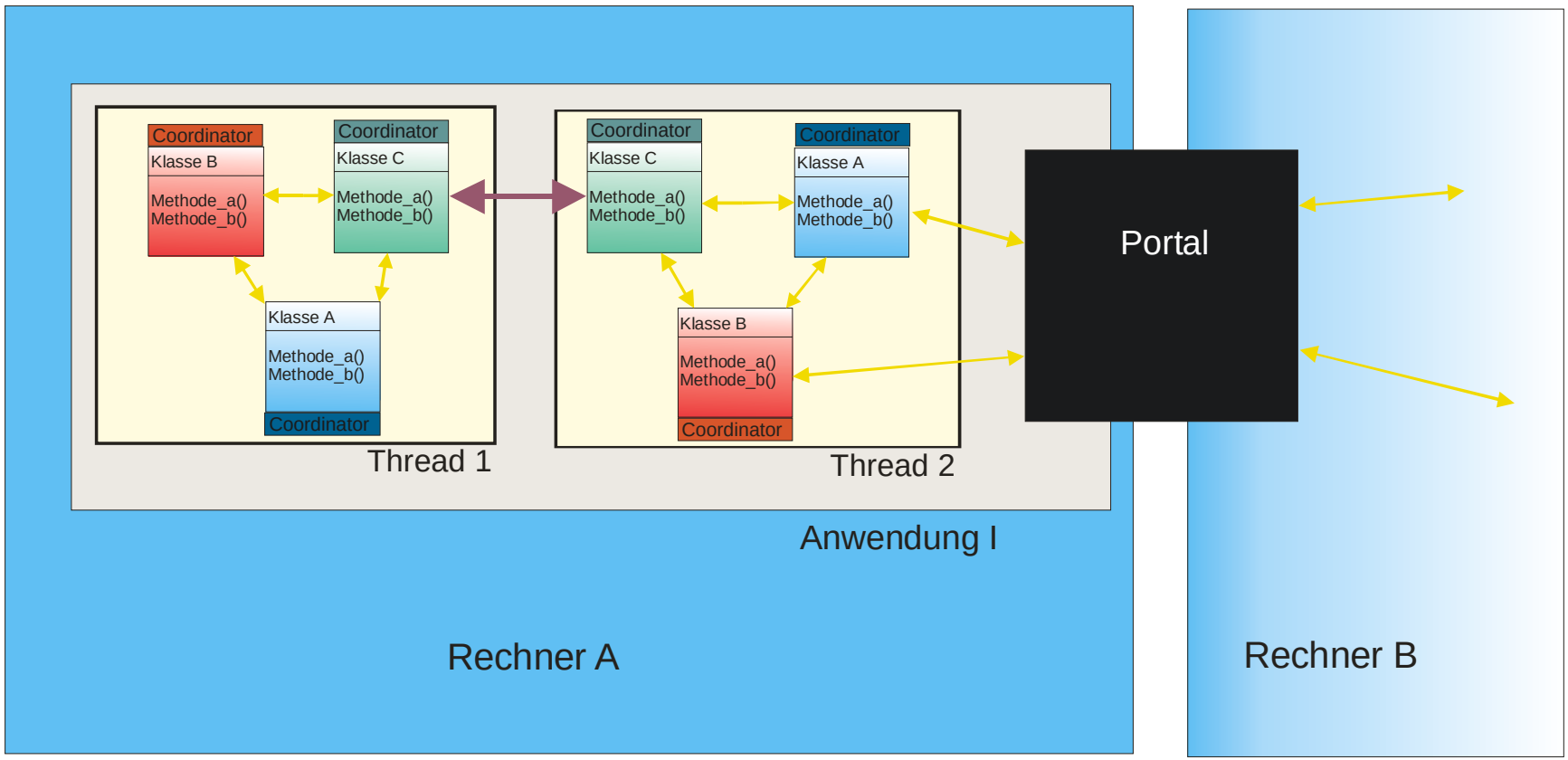
D Framework

- Besteht aus 3 Teilen:
  - „Coordinators“
    - Sprache: COOL
    - Koordinierung der Klassen auf Thread Ebene
  - „Portals“
    - Sprache: RIDL
    - Unterstützt Prozedurfernaufruf über Rechnergrenzen hinweg
  - Komponentensprache

D Framework

- Besteht aus 3 Teilen:
  - „Coordinators“
    - Sprache: COOL
    - Koordinierung der Klassen auf Thread Ebene
  - „Portals“
    - Sprache: RIDL
    - Unterstützt Prozedurfernaufruf über Rechnergrenzen hinweg
  - Komponentensprache
    - Beliebige objektorientierte Sprache (Java, C++, …)
    - Implementierung des eigentlichen Problems

D Framework


Coordinator
Klasse B
Methode_a()
Methode_b()
Coordinator
Klasse C
Methode_a()
Methode_b()
Klasse A
Methode_a()
Methode_b()
Coordinator
Thread 1
Coordinator
Klasse C
Methode_a()
Methode_b()
Coordinator
Klasse A
Methode_a()
Methode_b()
Klasse B
Methode_a()
Methode_b()
Coordinator
Thread 2
Anwendung I
Portal
Rechner A
Rechner B

Die Portale in der

Coordination **L**anguage

(COOL)

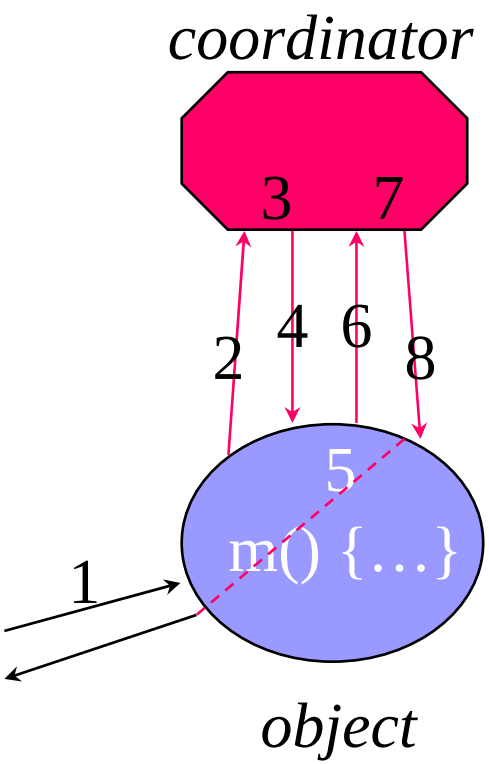
D Framework - COOL

- unterstützt die Verteilung des Problems auf Threads
- bietet Methoden zur Sperrung exklusiver Bereiche an
- überwacht die Ausführung der Methoden und synchronisiert sie mit anderen Threads

- Der Aufruf von Methoden folgt einem fest vorgegebenen Protokoll.

D Framework - COOL

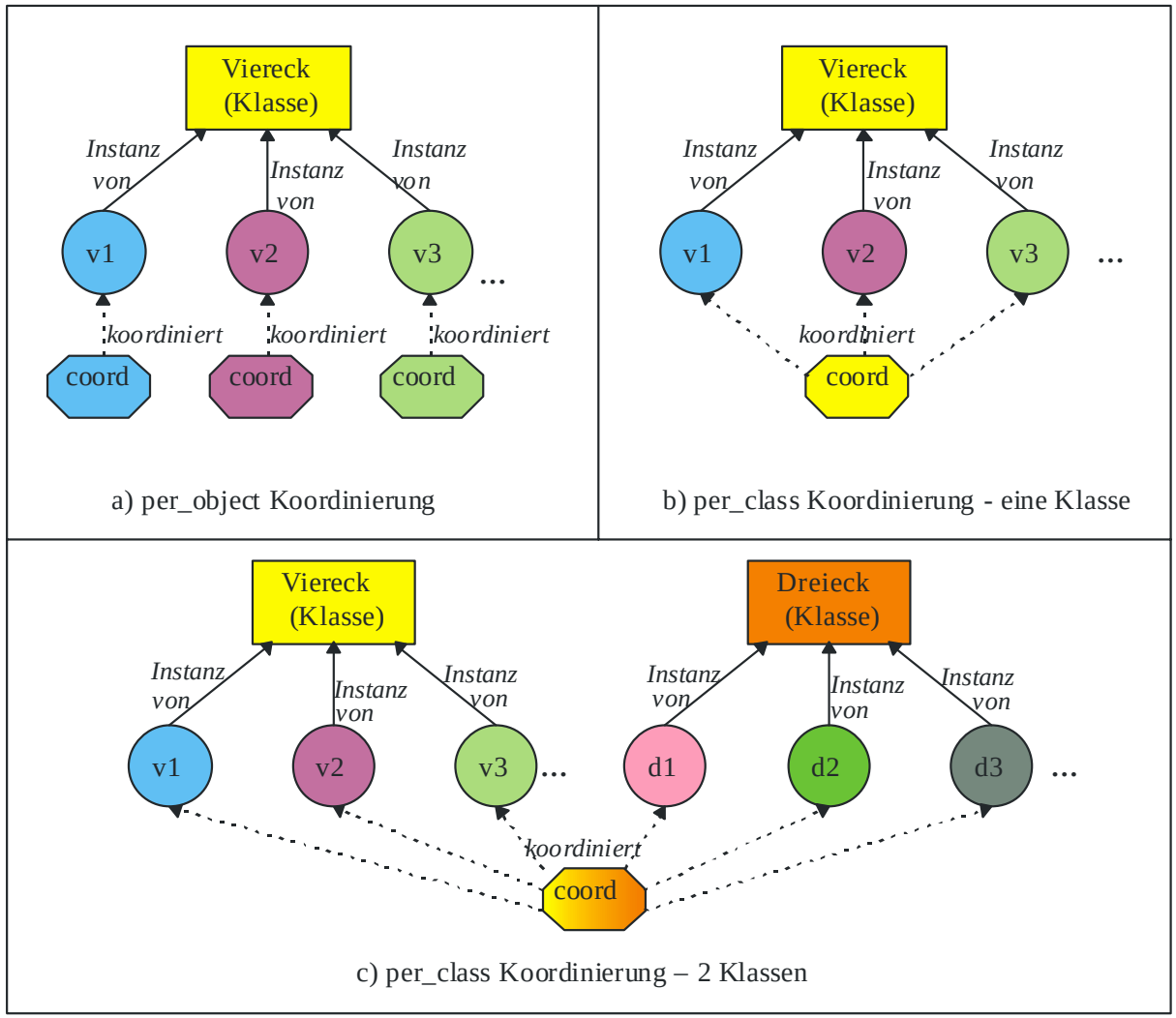
- Protokoll zwischen Objekten und Coordinators:



1. Im Thread T wird die Methode *m* des Objekts angefordert: *obj.m()*
2. Die Anfrage wird an den *coordinator* weitergeleitet.
3. Der *coordinator* überprüft ob bereits Sperrungen vorliegen und ob die definierten Vorraussetzungen alle erfüllt sind. Wenn nicht, dann wird der Aufruf verzögert. Wenn doch, dann werden die *on_entry statements* für Methode m ausgeführt .
4. Die Anfrage wird an das Objekt weitergereicht.
5. Thread *T* führt die Methode *m()* des Objektes aus.
6. Die Beendigung der Methode wird dem *coordinator* gemeldet.
7. Der *coordinator* führt seine *on_exit* Statements aus.
8. Der Aufruf ist beendet.

D Framework - COOL

Mögliche Verknüpfungen zwischen Klassen und *coordinator*:



D Framework - COOL

- Sperrungen kritischer Methoden:
  - Self-Exclusive Methods (selfex)
    - Selfex Methoden, können höchstenfalls von einem Thread gleichzeitig ausgeführt werden (z.B. internen Counter erhöhen)
  - Mutual Exclusion (mutex) Regeln
    - Legt eine Liste (!) von Methoden fest, die nicht parallel von mehreren unterschiedlichen Threads ausgeführt werden dürfen (z.B. Dateizugriffsmethoden)
    - Beinhaltet mindestens 2 Methoden

D Framework – COOL

Beispiel:

- „set“ Methoden sind von einander un-abhängig, da sie unterschiedliche Variablen verändern - *selfex*
- „set“ und „area“ bzw. „fill“ benutzen die gleichen Variablen - *mutex*

```
public class Rectangle {
  int width, height;
  Bitmap pixels;
  public Rectangle(int w, int h) {
    width = w; height = h;
    pixels = new Bitmap(w, h);
  }
  public void set_width(int newvalue) {
    width = newvalue;
    pixels.set_width(newvalue);
  }
  public void set_height(int newvalue) {
    height = newvalue;
    pixels.set_height(newvalue);
  }
  public int area() {
    return width*height;
  }
  public void fill(Color c) {
    pixels.fill( c );
  }
}
```

```
coordinator Rectangle {
  selfex set_width, set_height, fill; // area is not selfex
  mutex {set_width, area};
  mutex {set_height, area};
  mutex {set_width, fill};
  mutex {set_height, fill};
}
```

D Framework – COOL

„on_entry“ , „on_exit“, „requires“ Statements:

- on_entry & on_exit

- requires:
  - Prüft *coordinator*-Statusvariablen (vom Typ *Boolean*)
  - Damit lassen sich kritische Bereiche realisieren:
    - Variable == false ➔ kritischer Bereich frei – Variable true setzen
    - Variable == true ➔ kritischer Bereich besetzt - warten
    - Beim Verlassen des kritischen Bereichs: Variable = false

D Framework - COOL

Beispiel – Datenpuffer fester Größe:

```
coordinator BoundedBuffer {
  selfex put, take;
  mutex {put, take};
  condition empty = true, full = false;

  put: requires !full;
       on_exit {
         if (empty) empty = false;

       }
  take: requires !empty;
       on_exit {
         if (full) full = false;
         if (usedSlots == 0) empty = true;
       }
}
```

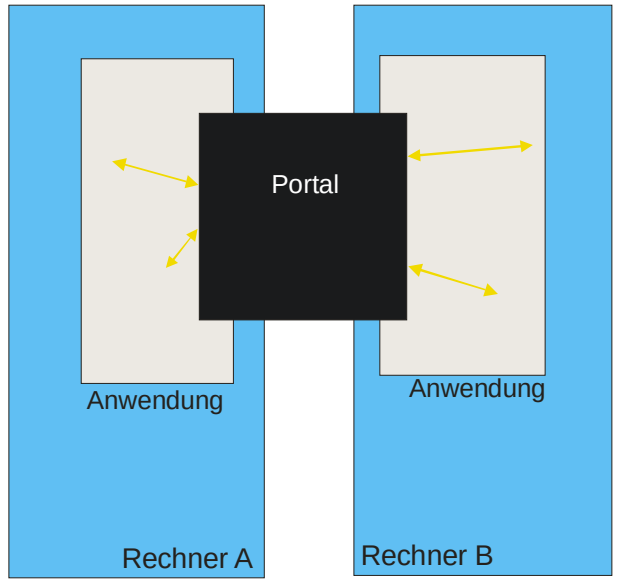
Das Remote Interface in

Remote **I**nterface **D**escription **L**anguage

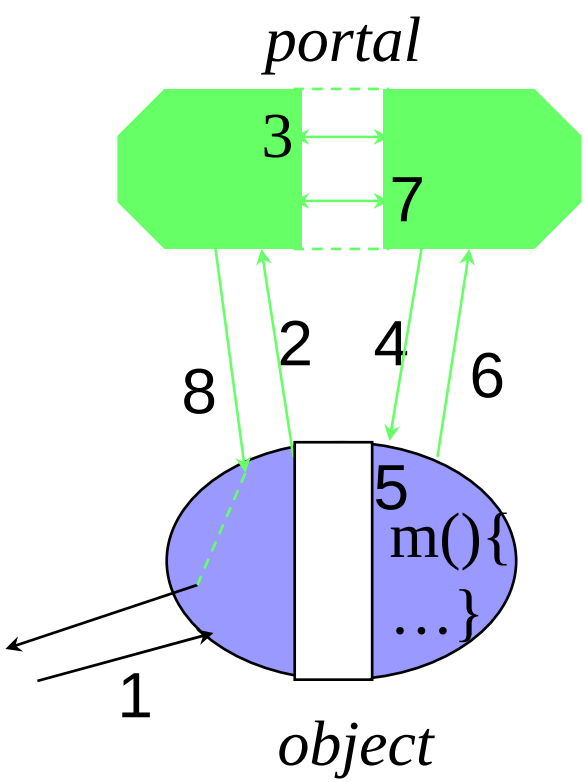
(RIDL)

D Framework - RIDL

- Mit RIDL entwirft man Portale, mit denen die Klassen der Programme von mehreren Rechnern verbunden werden können
- In der Portaldefinition wird festgelegt, welche Klassen von der Remoteseite aus erreichbar sind
- Für die Kommunikation muss vorher definiert werden, welchen Methoden mit welchem Datentyp kommunizieren darf und was für Datentypen sie zurückgeben.
- Für die Kommunikation gibt es auch wieder ein festgelegtes Protokoll



D Framework - RIDL



Protokoll Objekt/Portal:

1: Remote Methodenaufruf

2: Aufruf wird dem Portal gemeldet

3: Parameter werden von der Remote-Seite übertragen und ausgewertet

4: Anfrage an das Objekt

5: Ausführung der Methode

6: Rückgabewert an das Portal

7: Rückgabewert wird an die Remote-Seite gemäß den Protokollen übertragen

8: Methodenaufruf beendet, Rückgabewert gesendet

D Framework - RIDL

Remote operations :

- Die *remote operations* definieren, welche Methode über das Portal ansprechbar ist
- Die *return types* geben den erwarteten Typ des Rückgabewertes der Methode wieder
- Die Übergabeparameter müssen ebenfalls spezifiziert werden
- Es kann festgelegt werden, wie Daten übergeben werden (z.B. für *call by reference* / *call by value*)

D Framework – RIDL

Beispiel:

```
portal BoundedBuffer {
  void put(Book o);
  Book take();
 default:
    Book : copy { Book only all.String, all.int; };
}
```

```
public class BoundedBuffer {
  protected Object[] array;               // the elements
  protected int putPtr = 0, takePtr = 0;  // circular indices
  protected int capacity;
  protected int usedSlots = 0; // counter

  public BoundedBuffer (int size) {
    array = new Object[size]; capacity = size;
  }
  public void put(Object o) {
    array[putPtr] = o;
    putPtr = (putPtr + 1) % array.length;
    ++usedSlots;
  }
  public Object take() {
    Object old;
    old = array[takePtr];
    takePtr = (takePtr + 1) % array.length;
    --usedSlots;
    return old;
  }
}
```

Fallstudie mit DJ

D Framework - Fallstudie

- Vergleich von 10 Programmen:
  - mit D Framework und Java als Komponentensprache (DJ) geschrieben
  - in normalem Java geschrieben
- Mehrere typische Probleme, die auf verteilten Systemen implementiert werden können
- Beispiele: siehe Paper

D Framework – Fallstudie - Puffer

```
public class BoundedBuffer {
  private Object array[];
  private int putPtr = 0, takePtr = 0;
  private int usedSlots=0;

public void put(Object o) {
    array[putPtr] = o;
    putPtr = (putPtr + 1) % array.length;
    usedSlots++;
  }

  public Object take() {
    Object old = array[takePtr];
    array[takePtr] = null;
    takePtr = (takePtr + 1) % array.length;
    usedSlots--;
    return old;
  }
}
```

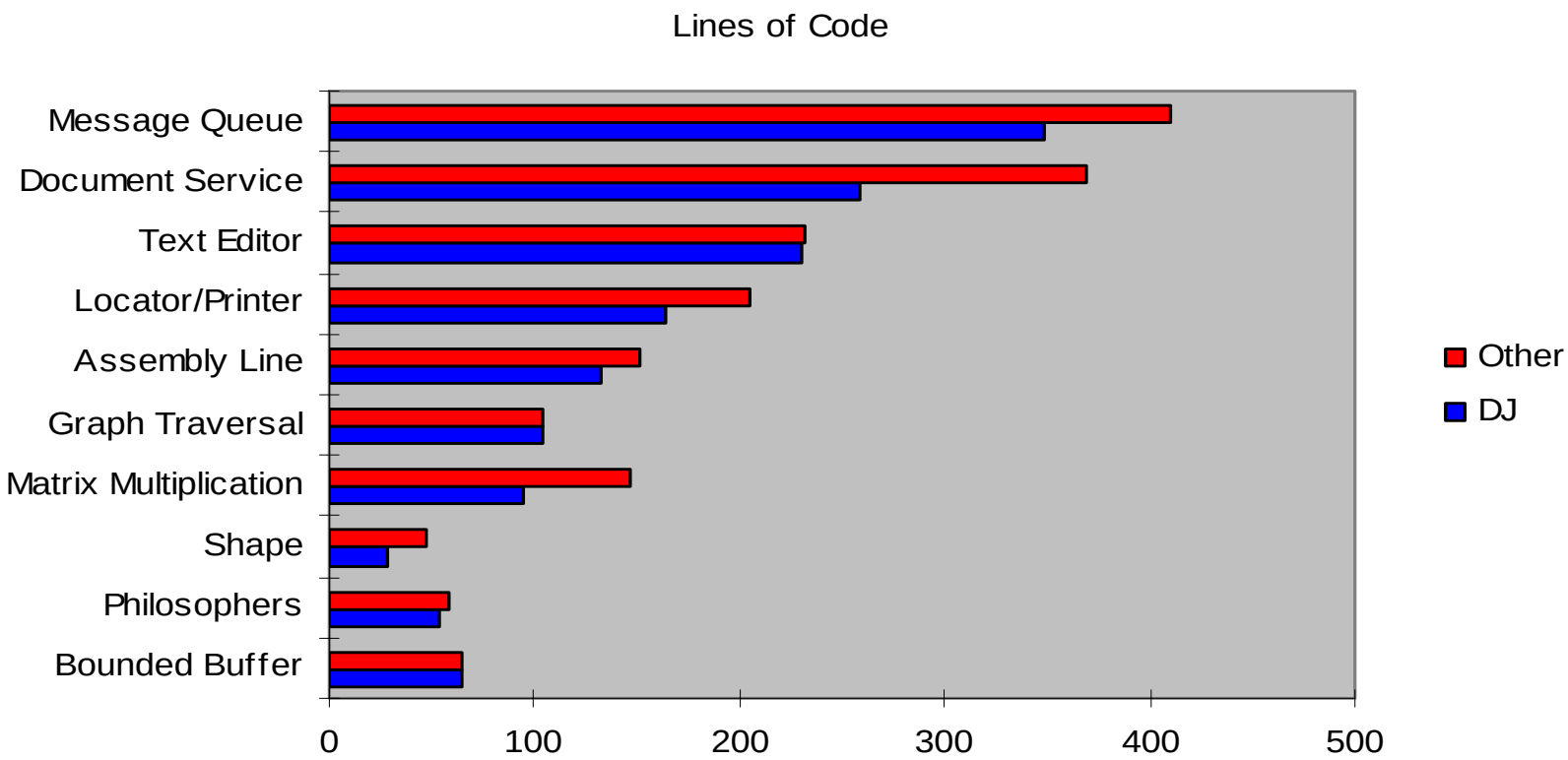
```
coordinator BoundedBuffer {
 selfex put, take;
 mutex {put, take};
 cond full = false, empty = true;
 put: requires !full;
   on_exit {
     empty = false;
     if (usedSlots == array.length)
          full = true;
   }
 take:  requires !empty;
     on_exit {
       full = false;
       if (usedSlots == 0) empty = true;
     }
}
```

```
public class BoundedBuffer {
  private Object[] array;
  private int putPtr = 0, takePtr = 0;
  private int usedSlots = 0;

public synchronized void put(Object o) {
    while (usedSlots == array.length) {
      try {
        wait();
      }
      catch (InterruptedException e) {};
    }
    array[putPtr] = o;
    putPtr = (putPtr + 1) % array.length;
    }
    if (usedSlots++ == 0)
      notifyAll();
  }

  public synchronized Object take() {
    while (usedSlots == 0) {
      try {
        wait();
      }
      catch (InterruptedException e) {};
    }
    Object old = array[takePtr];
    array[takePtr] = null;
    takePtr = (takePtr+1) % array.length;
    }
    if (usedSlots-- == array.length)
      notifyAll();
    return old;
  }
}
```

D Framework - Fallstudie



D Framework - Fallstudie

COOL

1000 method invocations per thread	
DJ	Java
Single thread calling a selfex method: **28ms**	*Single thread calling a synchronized method*: **7ms**
Two threads calling the same selfex method: **90ms**	*Two threads calling the same synchronized method:* **30ms**
Two threads calling 2 methods with requires, on_exit: **13s**	*Two threads, calling 2 methods with wait, notification:* **12s**

D Framework - Fallstudie

RIDL

1000 method invocations	
DJ	Java
no parameters: **10s**	*no parameters:* **10s**
one gref parameter: **24s**	*one parameter of type Remote:* **24s**
one copy parameter (object with 4 Integer fields): **26s**	*one parameter Serializable (object with 4 Integer fields):* **30s**
copying directive that selects 3 out of 4 Integer fields of a parameter: **35s**	*parameter with 3 Integer fields that is partially copied from an object of another class:* **28s**

Ausblick & Zusammenfassung

+ Trennung der Verteilung vom eigentlichen Problem
+ Verteilung in eigenständigen und unabhängigen Komponenten
+ Verwendung beliebiger Sprachen möglich

- Wenig genutzt
- Schlechte Performance

Literatur & Bildnachweis

Christina Lopes, “D: A Language Framework for Distributed Programming”
http://www2.parc.com/csl/groups/sda/publications/papers/Lopes-Thesis/

Christina Lopes, [PowerPoint Präsentation]
http://www.ccs.neu.edu/research/demeter/course/f99/lectures/lec6-RIDL.ppt