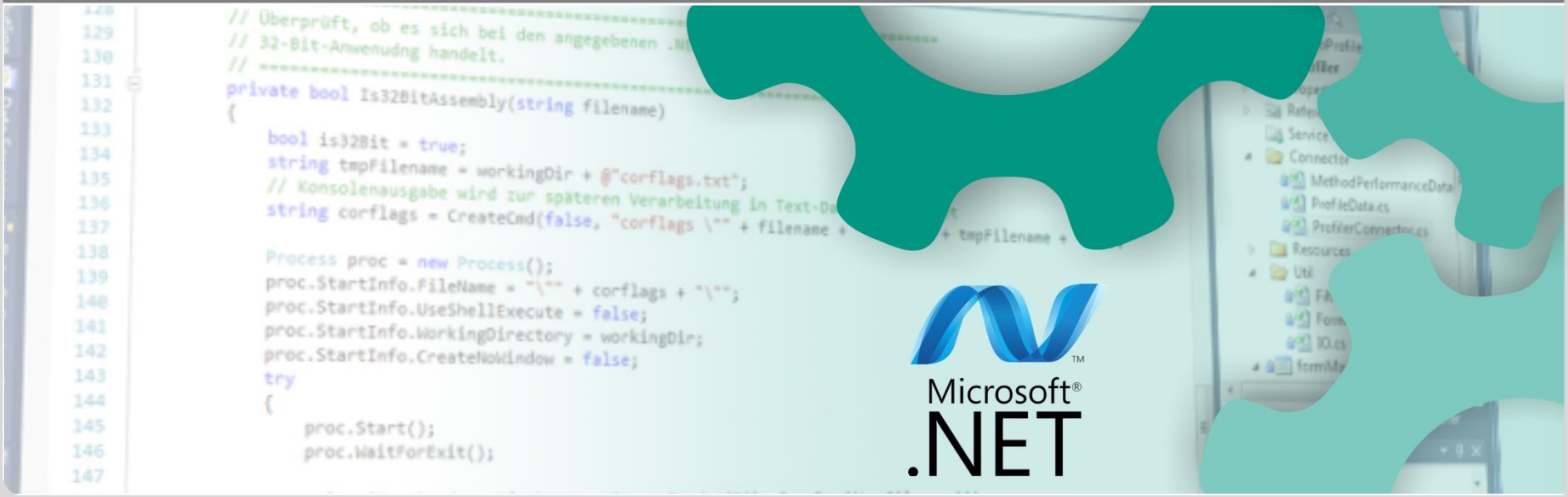
Karlsruhe Institute of Technology

Automatische Parallelisierung sequenzieller Programme mittels Architekturbeschreibungen

Diplomarbeit, Simon Wagner, 21. Juni 2013
Betreuer: Korbinian Molitorisz

IPD Tichy – Lehrstuhl für Programmiersysteme



Motivation

```
public void ProcessImages()
{

  foreach (Bitmap bmp in _inputStream)
  {
              Bitmap tmp_bmp = bmp;
              tmp_bmp = doCrop(tmp_bmp);
              tmp_bmp = doHistogramEqualization(tmp_bmp);
              tmp_bmp = doOilPainting(tmp_bmp);
              tmp_bmp = doResize(tmp_bmp);
              tmp_bmp = doSharpen(tmp_bmp);
              tmp_bmp = doSepia(tmp_bmp);
              tmp_bmp = ConvertColorsTo32bppArgb(tmp_bmp);
              ConsumeBmp(tmp_bmp);
  }

}
```

Motivation

```
public void ProcessImages()
{
  #region TADL: S1+ => S2+ => S3+ => S4+ => S5+ => S6+ => S7+ => S8+ => S9
  foreach (Bitmap bmp in _inputStream)
  {
    #region S1 Bitmap tmp_bmp = bmp;                                    #endregion
    #region S2 tmp_bmp = doCrop(tmp_bmp);                               #endregion
    #region S3 tmp_bmp = doHistogramEqualization(tmp_bmp);              #endregion
    #region S4 tmp_bmp = doOilPainting(tmp_bmp);                        #endregion
    #region S5 tmp_bmp = doResize(tmp_bmp);                             #endregion
    #region S6 tmp_bmp = doSharpen(tmp_bmp);                            #endregion
    #region S7 tmp_bmp = doSepia(tmp_bmp);                              #endregion
    #region S8 tmp_bmp = ConvertColorsTo32bppArgb(tmp_bmp);             #endregion
    #region S9 ConsumeBmp(tmp_bmp);                                     #endregion
  }
  #endregion
}
```

Ziel der Diplomarbeit

- Automatische Parallelisierung sequenzieller Programme
- Bereitstellung von Tuningparametern zur Optimierung auf verschiedenen Plattformen

Gliederung des Vortrags

- Motivation
- Ziel der Diplomarbeit
- Verwandte Arbeiten
- Anforderungen
- Automatische Transformation
- Evaluation
- Zusammenfassung
- Ausblick

Verwandte Arbeiten

	Quellcode-Analyse	Erzeugung von Parallelisierungs-vorschlägen	Architektur-beschreibungs-sprache	Erzeugung von parallelem Programmen	Transformation sequenzieller nach paralleler Quellcode	Autotuning-fähig
TADL 1 [S10]			x	x		x
XJava [O13]	x		x	x	x	x
Profiling [RV+07, RV+08, RV+10]	x	x				
Profiling (Pipeline) [TF10]	x	x				
Heuristiken [RV+09]	x	x				
Parallelisierende Compiler	x			x		
DA J. Huck [H13]	x	x	x			x *)
Diese Arbeit	x		x	x	x	x

*) Die Parallelisierungsvorschläge werden in TADL formuliert und enthalten dabei implizit Tuning-Parameter.

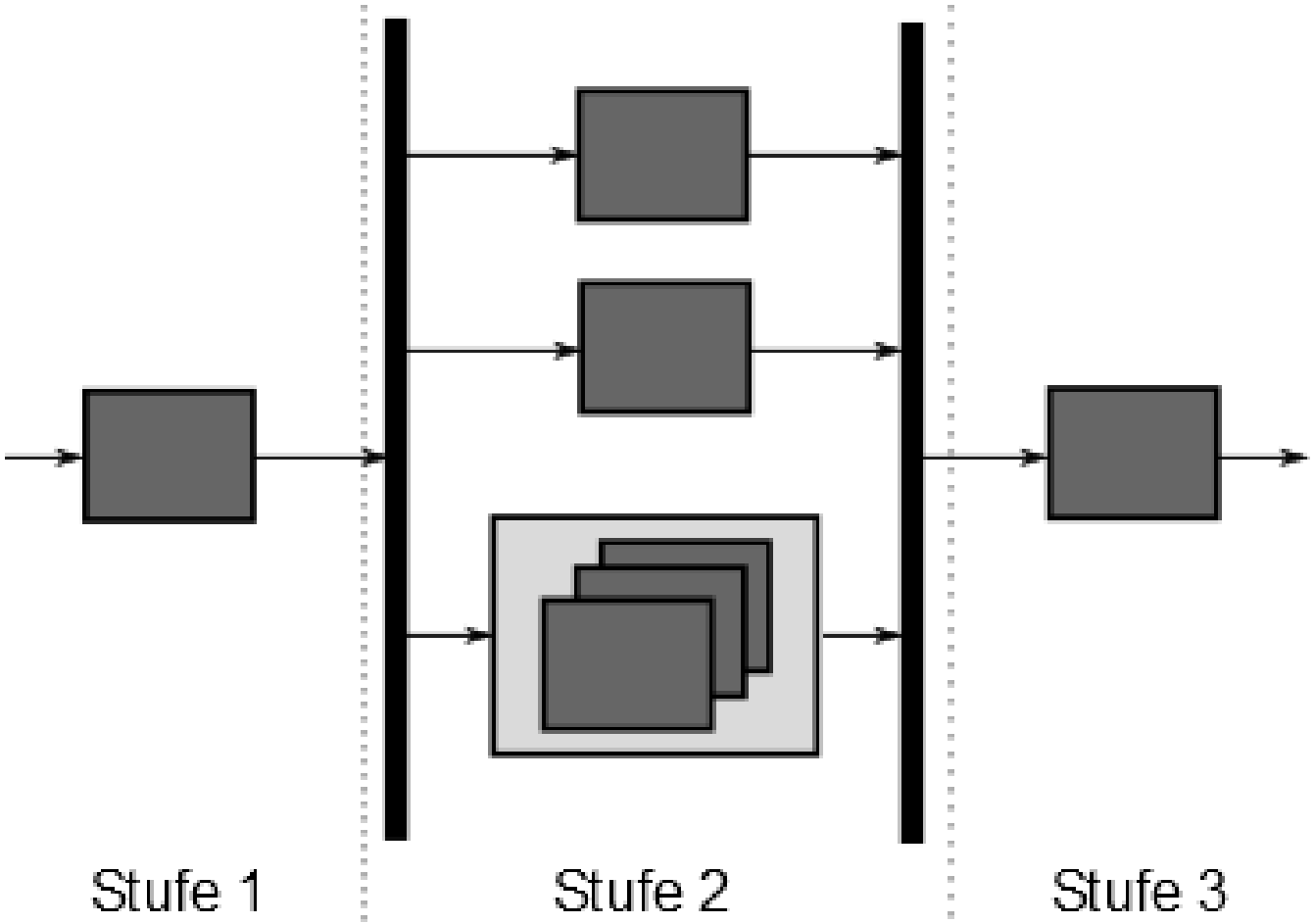
Anforderungen

- **A_1: Laufzeitbibliothek**
  - Übernahme der Steuerung der parallelen Ausführung und Zurverfügungstellung von parallelen Architekturmustern
- **A_2: Architekturbeschreibungssprache zur Angabe von parallelen Architekturen**
  - Auf beliebige Quelltextbereiche bezugnehmende Beschreibungssprache zur Angabe gewünschter paralleler Architekturen
- **A_3: Zutage Fördern von Tuningparametern zur Laufzeitoptimierung**
  - Anpassbarkeit der parallelen Programme ohne neu kompilieren zu müssen

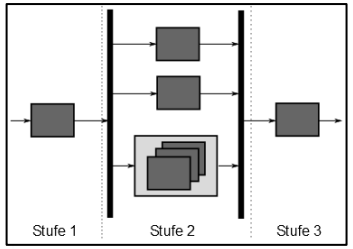
A_1 Laufzeitbibliothek

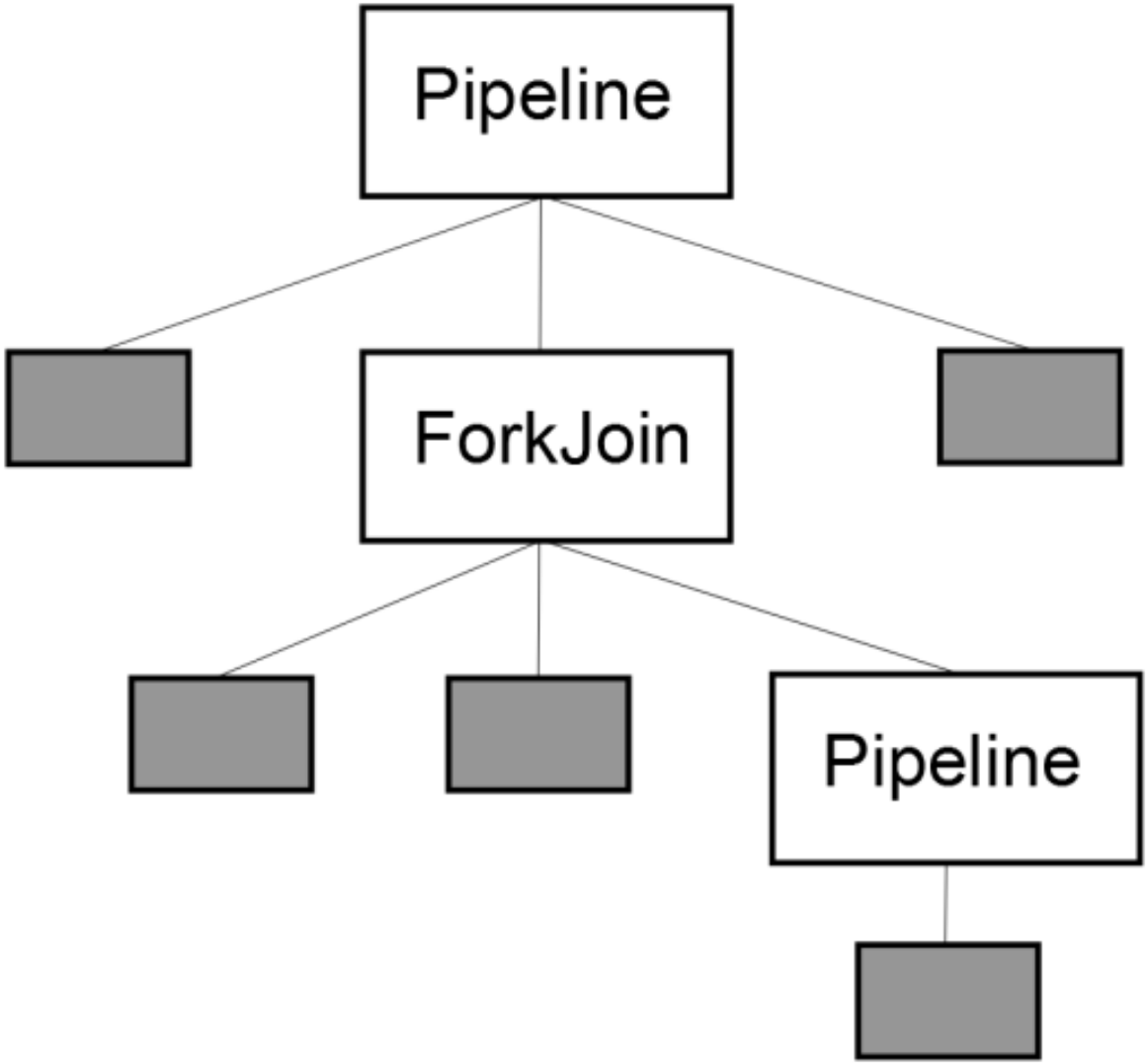
- Erzeugung und Verwaltung von Fäden
- Synchronisation wird vor dem Entwickler verborgen
- Implementierte Architekturen
  - Fork/Join
  - Pipeline
  - Sequenz
- Verschachtelung möglich

A_1 Laufzeitbibliothek

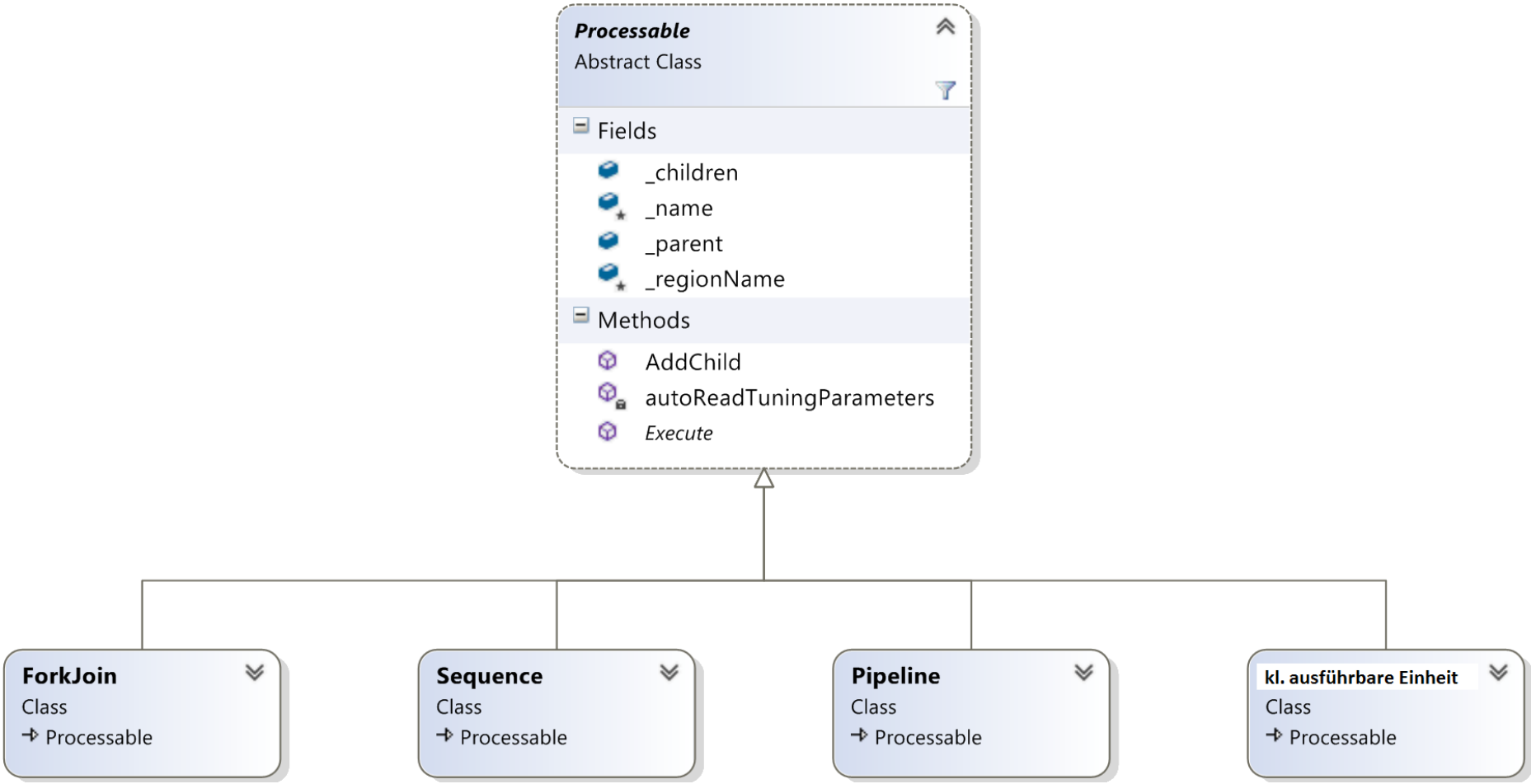


A_1 Laufzeitbibliothek





A_1 Laufzeitbibliothek



A_1 Laufzeitbibliothek – kleinste ausführbare Einheit

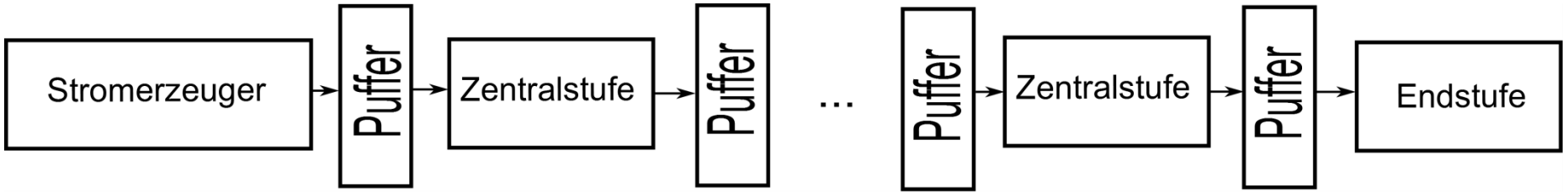
```
#region S2 tmp_bmp = doCrop(tmp_bmp);                    #endregion
```

```
class uniqueName__4S2 : TADL.Runtime.Processable
{
    public override Dictionary<String, Object> Execute(Dictionary<String, Object> _pipeInVars)
    {
        ImageProcessing.ImageProcessor thisPtr = (ImageProcessing.ImageProcessor)_pipeInVars["this"];
        Bitmap tmp_bmp = (Bitmap)_pipeInVars["tmp_bmp"];
        tmp_bmp = thisPtr.doCrop(tmp_bmp);
        _pipeInVars["tmp_bmp"] = tmp_bmp;
        return _pipeInVars;
    }
}
```

A_1 Laufzeitbibliothek – Pipelinearchitektur

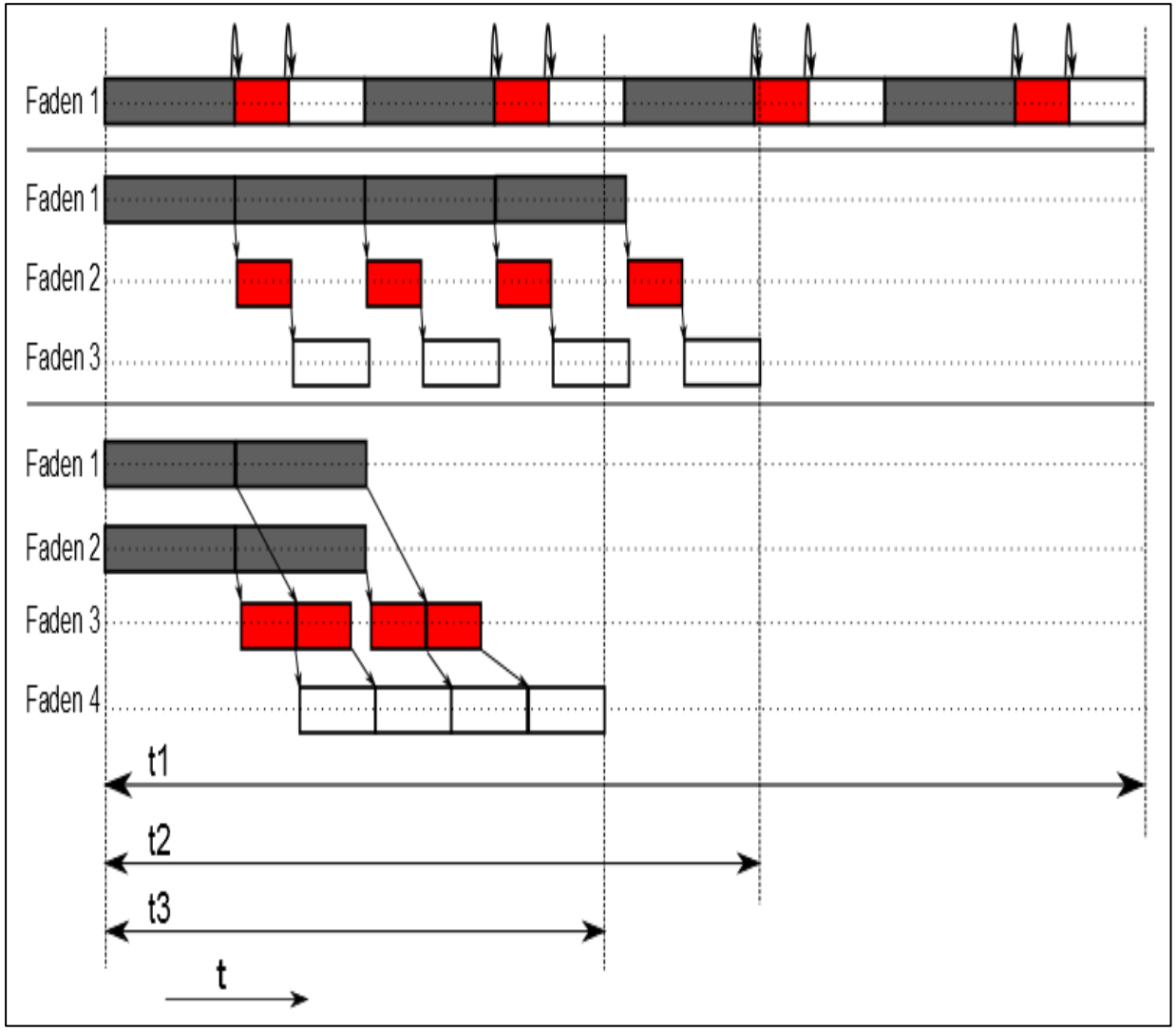
- Stromerzeugende Stufe
- Zentralstufe
  - Ein- und Ausgabepuffer
- Endstufe
  - Nur Eingabepuffer

- Tuningparameter
  - Replikation
  - Fusion
  - Schrittgröße
  - Puffergröße
  - Reihenfolgeerhaltung



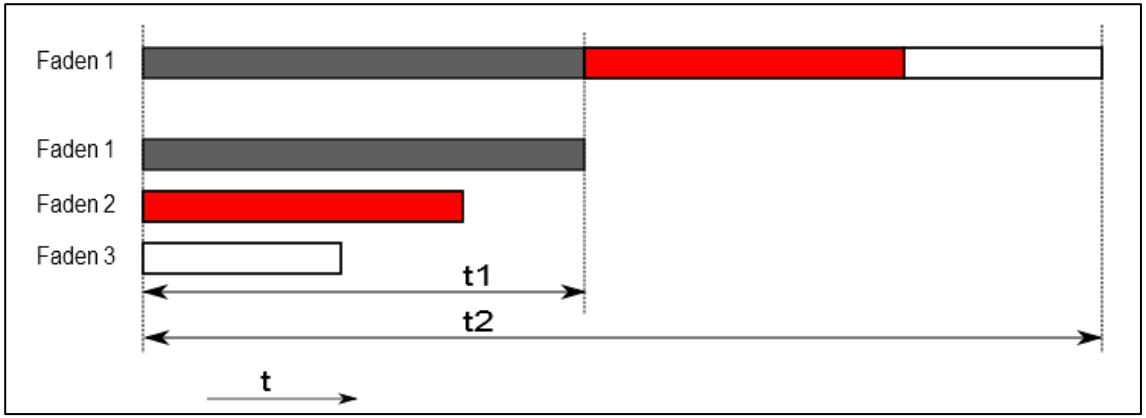
A_1 Laufzeitbibliothek – Pipelinearchitektur
Auswirkung der Stufenreplikation

- Sequenzielle Ausführung
- Ein Faden pro Stufe
- Replikation der ersten Stufe



A_1 Laufzeitbibliothek – Fork/Join-Architektur

- Verkürzung der Laufzeit durch Anwendung der Fork/Join-Architektur



- Tuningparameter:
  - Aufgabenfusion

A_2 Architekturbeschreibungssprache TADL

- TADL = „Tunable Architecture Description Language“
- TADL-Ausdrücke
  - Operatoren
  - Bezeichner

Operator	Semantik	Beispiel
=>	Pipeline	A => B => C
||	Fork/Join	A || B || C
;	Sequenz	A ; B ; C
+	Replikation	A+

- Weitere Beispiele
  - S1+ => S2+ => S3+ => S4+ => S5+ => S6+ => S7+ => S8+ => S9
  - A+ => (B || C)+ => D
  - DefindeProblem ; (ComputeX || SearchWord) ; DisplayResult

TADL-Annotation – Beispiel ImageProcessing

```
public void ProcessImages()
{
  #region TADL: S1+ => S2+ => S3+ => S4+ => S5+ => S6+ => S7+ => S8+ => S9
  foreach (Bitmap bmp in _inputStream)
  {
    #region S1 Bitmap tmp_bmp = bmp;                                    #endregion
    #region S2 tmp_bmp = doCrop(tmp_bmp);                               #endregion
    #region S3 tmp_bmp = doHistogramEqualization(tmp_bmp);              #endregion
    #region S4 tmp_bmp = doOilPainting(tmp_bmp);                        #endregion
    #region S5 tmp_bmp = doResize(tmp_bmp);                             #endregion
    #region S6 tmp_bmp = doSharpen(tmp_bmp);                            #endregion
    #region S7 tmp_bmp = doSepia(tmp_bmp);                              #endregion
    #region S8 tmp_bmp = ConvertColorsTo32bppArgb(tmp_bmp);             #endregion
    #region S9 ConsumeBmp(tmp_bmp);                                     #endregion
  }
  #endregion
}
```

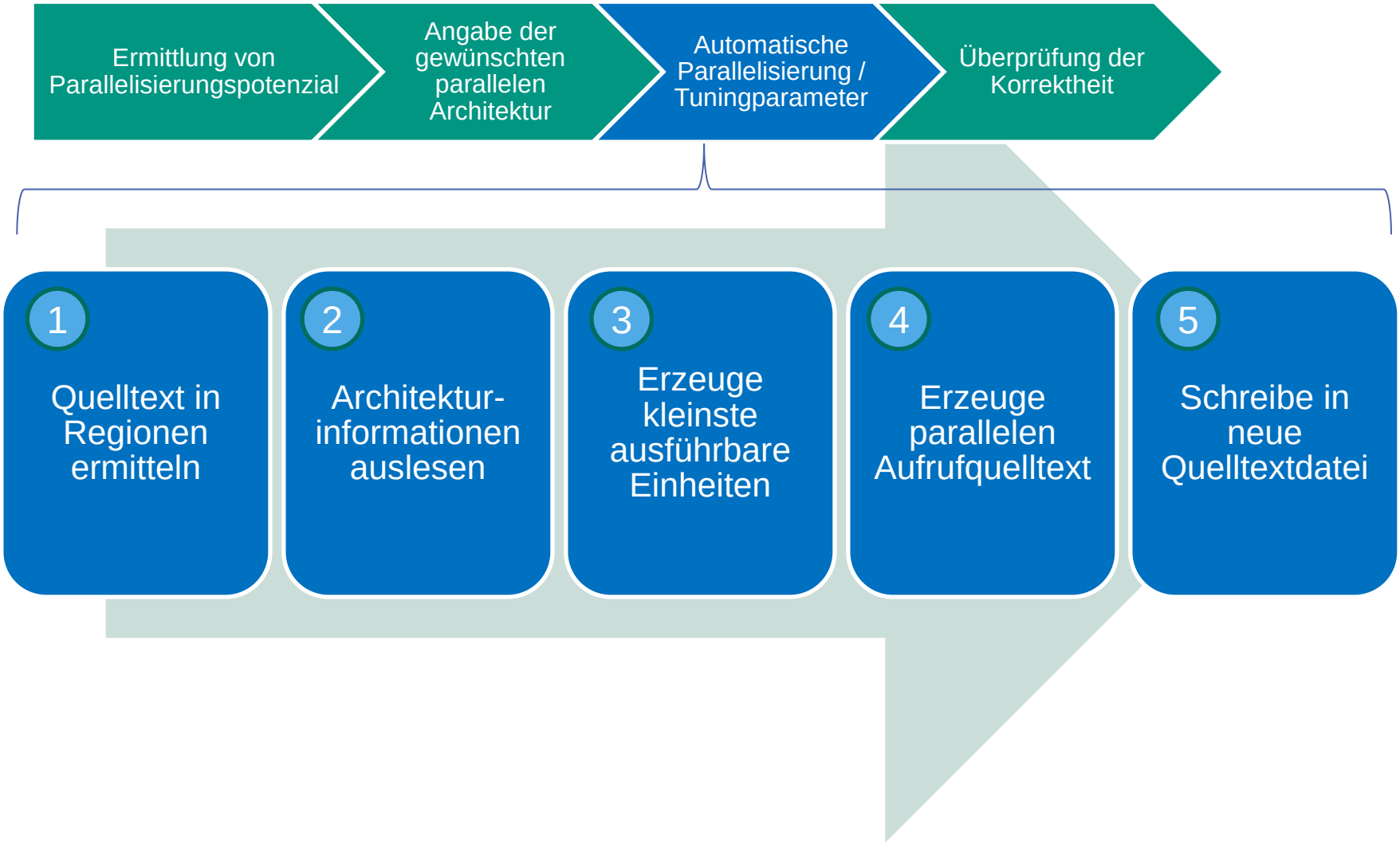
A_3 Tuningparameter

- Replikationsgrad
- Fusion
  - Pipelinestufenfusion
  - Aufgabenfusion bei Fork/Join
- Reihenfolgeerhaltung
- Puffergröße
- Schrittgröße
- Sequenzielle Ausführung
- Verteilungsstrategie (*)
- Alternative (*)
- Parallele Rekursionstiefe (*)

Tuningparameter – Format der Tuningdatei

```
<Replication>
  <ID>uniqueName__12S6</ID>
  <Value>3</Value>
  <TADL-Expression>S1+ => S2+ => S3+ => S4+ => S5+ =>
     S6+ => S7+ => S8+ => S9</TADL-Expression>
  <Project>ImageProcessing</Project>
  <Document>ImageProcessing.cs</Document>
  <Position>2766</Position>
</Replication>
```

Automatische Parallelisierung



Automatische Parallelisierung – 3 Erzeugung der kleinsten ausführbaren Einheiten

1. Datenflussanalyse
2. Erzeuge Ein- und Ausgabe-Quelltext
3. Füge Originalquelltext ein
4. Füge vor Klassenvariablen und Methoden „thisPtr.“ hinzu

```
class uniqueName__4S2 : TADL.Runtime.Processable
{
    public override Dictionary<String, Object> Execute(Dictionary<String, Object> _pipeInVars)
    {
        ImageProcessing.ImageProcessor thisPtr = (ImageProcessing.ImageProcessor)_pipeInVars["this"];
        Bitmap tmp_bmp = (Bitmap)_pipeInVars["tmp_bmp"];
        tmp_bmp = thisPtr.doCrop(tmp_bmp);
        _pipeInVars["tmp_bmp"] = tmp_bmp;
        return _pipeInVars;
    }
}
```

Automatische Parallelisierung – 4 paralleler Aufrufquelltext

```
if (!TADL.Runtime.TuningParameters.getSeqExecParam("sequencialExecution0")) {
    uniqueName__2S1 inst_uniqueName__2S1 = new uniqueName__2S1("uniqueName__2S1");


    StreamGenerator_inst_uniqueName__0Pipeline StreamGenerator_inst_uniqueName__0Pipeline =
        new StreamGenerator_inst_uniqueName__0Pipeline(this );
    TADL.Runtime.Pipeline inst_uniqueName__0Pipeline =
        new TADL.Runtime.Pipeline("uniqueName__0Pipeline");
    inst_uniqueName__0Pipeline.setLoopStage(StreamGenerator_inst_uniqueName__0Pipeline);
    inst_uniqueName__0Pipeline.AddChild(inst_uniqueName__2S1);


    inst_uniqueName__0Pipeline.Initialize();
    Dictionary<String, Object> result = inst_uniqueName__0Pipeline.Execute();
}
else {
```

Evaluation

Programm	Speedup manuell parallelisiert	Speedup Automatisch parallelisiert	% des manuellen Speedups	Anzahl TADL-Ausdrücke	Anzahl Tuning-paramter
Image-Processing	7,19	7,18	99,9%	1 Pipeline	13
MergeSort	4,52	4,60	101,8%	1 Fork/Join und Sequenz	1 (*)
RayTracer		3,13		2 Pipelines (verschachtelt)	5 und 7
Desktop-Search	(2,12)	1,67	(78,8%)	2 Pipelines (verschachtelt)	12 und 7

Zusammenfassung

- Gute Speedups automatisiert erreichbar ohne den Quelltext verstehen zu müssen
- Ergebnisse der Diplomarbeit auf drei Arten nutzbar:
  - Nutzung der Laufzeitbibliothek durch den Entwickler
  - Manuelle TADL-Annotation und automatische Parallelisierung
  - Automatische Erzeugung von in TADL formulierten Parallelisierungsvorschlägen und automatische Parallelisierung
- Optimierbar durch Autotuner
  - Getrennte Tuningparameterräume sind einzeln optimierbar

Ausblick

- Erweiterung der Datenflussanalyse um fehlerhafte Annotationen zu erkennen
  - Die Datenflussanalyse wird bisher nur für die direkt zur Ausführung benötigten Variablen durchgeführt
  - Was in aufgerufenen Methoden passiert, wird nicht untersucht
- Erweiterung/Verbesserung der bestehenden Laufzeitbibliothek
  - Austausch Dictionaries
- Implementierung der restlichen TADL-Operatoren
  - Reduktion (–), Rekursion (\), Alternative (?)
- Portierung auf andere Laufzeitbibliothek
  - Z.B. Intel Threading Building Blocks (TBB)
  - Dabei sollte das Autotuning weiter möglich bleiben

Automatische Parallelisierung sequenzieller Programme mittels Architekturbeschreibungen

Karlsruhe Institute of Technology

Literatur

[S10]	C. Schäfer, „Automatisierte Performanzoptimierung Paralleler Architekturen“ Institut für Programmstrukturen und Datenorganisation, Karlsruher Institut für Technologie, 2010, Dissertation
[O13]	F. Otto, „Objektorientierte Stromprogrammierung“ Dissertation, Institut für Programmstrukturen und Datenorganisation, Karlsruher Institut für Technologie (2013)
[RV+07]	S. Rul, H. Vandierendonck, K. D. Bosschere, „Function Level Parallelism Driven by Data Dependencies“ ACM SIGARCH, Computer Architecture News, 2007
[RV+08]	S. Rul, H. Vandierendonck, K. D. Bosschere, „Detecting the Existence of Coarse-Grain Parallelism in General-Purpose Programs“ Proceedings of the First Workshop on Programmability Issues for Multi-Core Computers, MULTIPROG-1, page 12, 1 2008.
[RV+09]	S. Rul, H. Vandierendonck, K. Bosschere, „Towards Automatic Program Partitioning“ CF '09 Proceedings of the 6th ACM conference on Computing frontiers, 2009
[RV+10]	S. Rul, H. Vandierendonck, K. D. Bosschere, „A profile-based tool for finding pipeline parallelism in sequential programs“ Parallel Computing (2010) Vol. 36, Nr. 9, 531-551
[TF10]	G. Tournavitis, B. Franke, „Semi-Automatic Extraction and Exploitation of Hierarchical Pipeline Parallelism Using Profling Information“ International Conference on Parallel Architectures and Compilation Techniques PACT. 2010, S. 377-388.
[H13]	J. Huck, „Generierung paralleler Architekturbeschreibungen durch kombinierte statische und dynamische Analysen“ Diplomarbeit, Institut für Programmstrukturen und Datenorganisation, Karlsruher Institut für Technologie (2013)

Karlsruhe Institute of Technology

Ende

TADL-Annotation – Beispiel RayTracer

```
#region TADL: S1S2+
//for (int y = 0; y < screenHeight; y++)
foreach (int y in Enumerable.Range(0, screenHeight - 1).ToList())
{
    #region S1S2
    #region TADL: S1+ => S2+
    //for (int x = 0; x < screenWidth; x++)
    foreach (int x in Enumerable.Range(0,screenWidth - 1).ToList())
    {
        #region S1
        Color color = TraceRay(new Ray() {
                            Start = scene.Camera.Pos,
                            Dir = GetPoint(x, y, scene.Camera) },
                            scene, 0);
        #endregion
        #region S2
        setPixel(x, y, color.ToDrawingColor());
        #endregion
    }
    #endregion
    #endregion
}
#endregion
```

Automatische Parallelisierung – 4 paralleler Aufrufquelltext

```
if (!TADL.Runtime.TuningParameters.getSeqExecParam("sequencialExecution0")) {
    uniqueName__2S1 inst_uniqueName__2S1 = new uniqueName__2S1("uniqueName__2S1");
    uniqueName__4S2 inst_uniqueName__4S2 = new uniqueName__4S2("uniqueName__4S2");
    uniqueName__6S3 inst_uniqueName__6S3 = new uniqueName__6S3("uniqueName__6S3");
    uniqueName__8S4 inst_uniqueName__8S4 = new uniqueName__8S4("uniqueName__8S4");
    uniqueName__10S5 inst_uniqueName__10S5 = new uniqueName__10S5("uniqueName__10S5");
    uniqueName__12S6 inst_uniqueName__12S6 = new uniqueName__12S6("uniqueName__12S6");
    uniqueName__14S7 inst_uniqueName__14S7 = new uniqueName__14S7("uniqueName__14S7");
    uniqueName__16S8 inst_uniqueName__16S8 = new uniqueName__16S8("uniqueName__16S8");
    uniqueName__17S9 inst_uniqueName__17S9 = new uniqueName__17S9("uniqueName__17S9");
    StreamGenerator_inst_uniqueName__0Pipeline StreamGenerator_inst_uniqueName__0Pipeline =
        new StreamGenerator_inst_uniqueName__0Pipeline(this );
    TADL.Runtime.Pipeline inst_uniqueName__0Pipeline = new TADL.Runtime.Pipeline("uniqueName__0Pipeline");
    Dictionary<String, Object>input_inst_uniqueName__0Pipeline = new Dictionary<String, Object>();
    inst_uniqueName__0Pipeline.setLoopStage(StreamGenerator_inst_uniqueName__0Pipeline);
    inst_uniqueName__0Pipeline.AddChild(inst_uniqueName__2S1);
    inst_uniqueName__0Pipeline.AddChild(inst_uniqueName__4S2);
    inst_uniqueName__0Pipeline.AddChild(inst_uniqueName__6S3);
    inst_uniqueName__0Pipeline.AddChild(inst_uniqueName__8S4);
    inst_uniqueName__0Pipeline.AddChild(inst_uniqueName__10S5);
    inst_uniqueName__0Pipeline.AddChild(inst_uniqueName__12S6);
    inst_uniqueName__0Pipeline.AddChild(inst_uniqueName__14S7);
    inst_uniqueName__0Pipeline.AddChild(inst_uniqueName__16S8);
    inst_uniqueName__0Pipeline.AddChild(inst_uniqueName__17S9);
    inst_uniqueName__0Pipeline.Initialize();
    Dictionary<String, Object> result = inst_uniqueName__0Pipeline.Execute(input_inst_uniqueName__0Pipeline);
}
else {
```