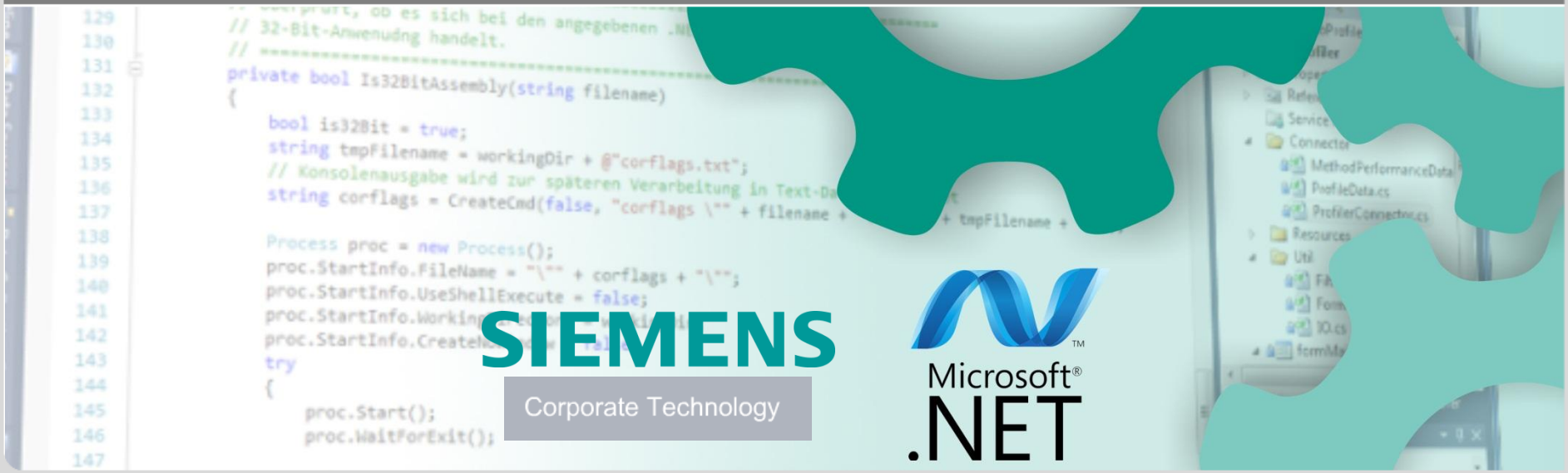
Karlsruhe Institute of Technology

Automatic Parallelization using *AutoFutures*

Dipl.-Inform. Korbinian Molitorisz

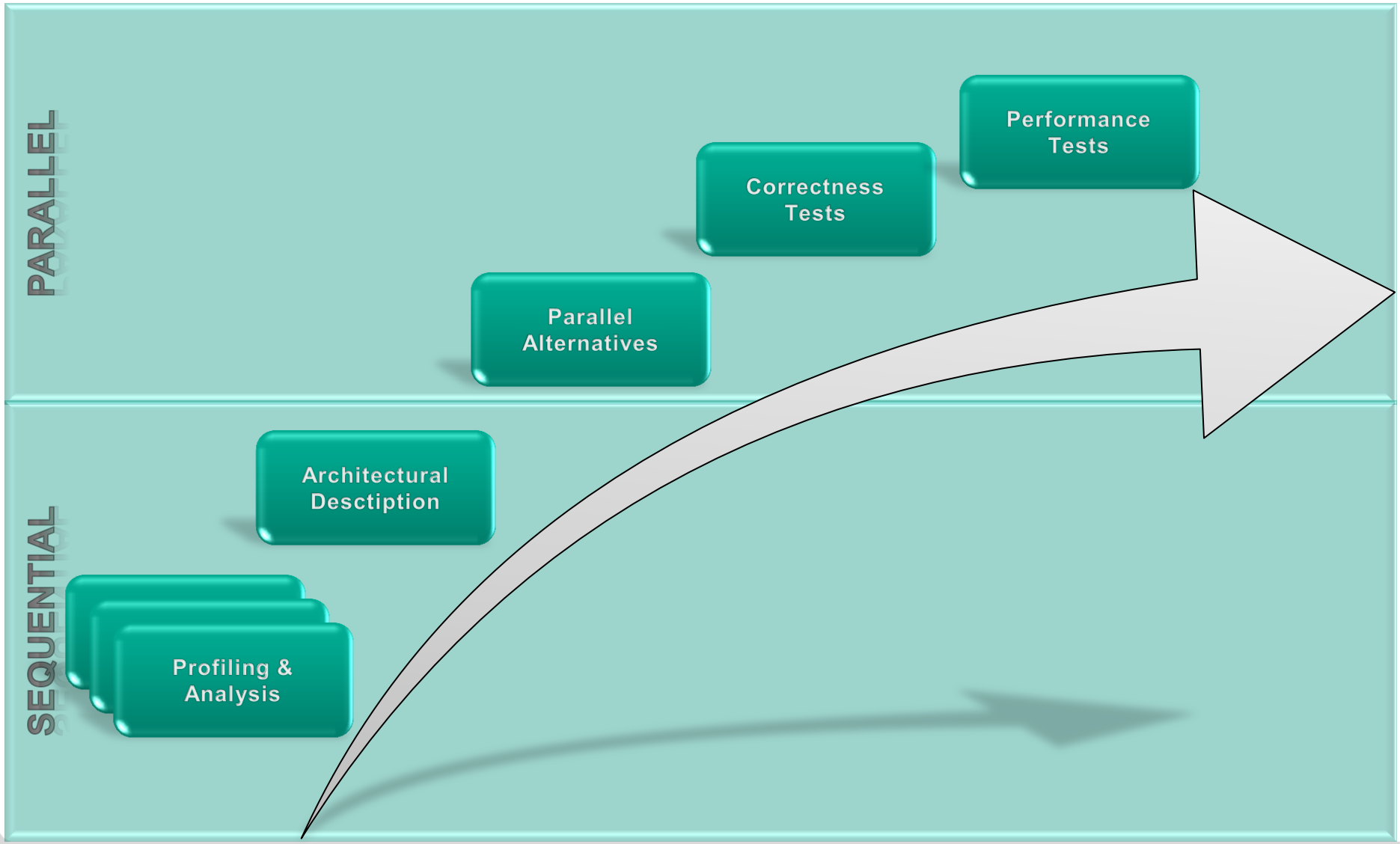
IPD Tichy, Shared Research Group: *Pattern-driven Parallelization*



Agenda

- What is this „Shared Research Group: *Pattern-driven Parallelization*“?
  - Cooperation shared equally between the KIT and Siemens CT
  - Founded by Urs Gleim (Siemens CT), Prof. Walter F. Tichy and Korbinian Molitorisz (KIT) in late 2011
- What are its research topics?
  - Pattern-driven parallelization process
  - Detecting patterns at different „layers“
  - Formalizing them to make them detectable by tools
  - Replacing by parallel patterns
  - Testing their performance and correctness
- What are *AutoFutures*?

Pattern-driven parallelization process



AutoFutures - Concept

- Computation concept *Future*
  - Placeholder for the result of an asynchronous computation
  - Access to the result either returns it oder blocks until available
- Concept *AutoFuture*
  - Automatically detect potential separation points
  - Split control flow via Futures
  - Join control flow heuristically at synchronisation points
- Informal pattern: Function
  - Criterion 1: Method with return value
  - Criterion 2: Method without global access
  - Criterion 3: Return value used „some time later“
- Criterion 3 crucial, see later

AutoFutures - Implementation

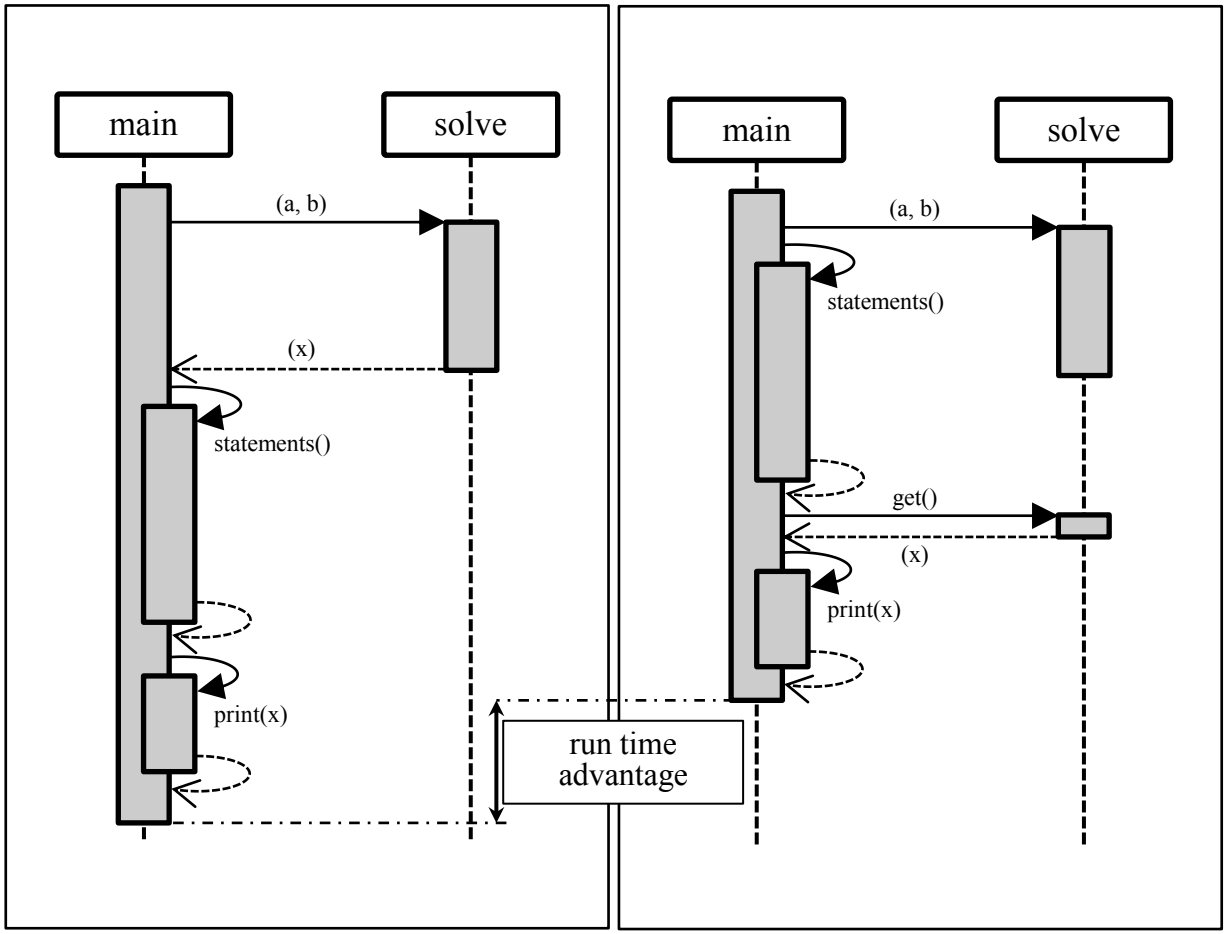
```
main() {
   x = solve(A, b);
   statements();
   print(x);
}
```

sequential

```
main() {
   f = async(solve(A, b));
   statements();
   x = get(f);
   print(x);
}
```

parallel

AutoFutures - Implementation



Synchronisation points – pessimistic heuristic

- Context-insensitive side effect analysis

```
class Foo {
  int i;

  inc() {
    i++;
  }

  main() {
    Foo f1 = new Foo();
    Foo f2 = new Foo();
    f1.inc();
    f2.inc();
  }
}
```

R: {i}, W: {i}

R: {i}, W: {i}

Synchronisation points – optimistic heuristic

- Side effect analysis & heuristics

```
class Foo {
  int i;

  inc() {
    i++;
  }

  main() {
    Foo f1 = new Foo();
    Foo f2 = new Foo();
    f1.inc();
    f2.inc();
  }
}
```

R: {i}, W: {i}

R: {i}, W: {i}

Context-sensitivity:

- Same method
- No arguments
- different objects

Pilot study results

	MergeSort	Matrix	PMD	ANTLR	ImageJ
Source lines	34	81	44782	36733	93899
Methods	3	5	3508	1998	4505
AutoFutures	1	21	189	1043	3351
Input data	array	matrix	rules	files	images
Speedup min	2.16	2.61	0.91	0.76	1.74
Input size	1.600.000	400x400	13	18	512x512
Speedup max	2.70	3.34	--	--	2.04
Input size	8.000.000	600x600	--	--	1448x1448

- Speedup [0,76;3,34] on an Intel Quadcore workstation
- Lessons learned:
  - Pattern Function hardly found: Criterion 3 fails → prepone?
  - Criterion 1 extended to capture functions and procedures
  - Static heuristic for synchronisation points not precise enough → dynamic analysis?
  - Look for more use cases for *AutoFuture*

Current & Future work

- Results from automated use case analysis (~180.000 LOC):
  - Informal Pattern 2: Speculative operation
    - Criterion 3 revised: Prepone calculation
    - Speculative operation at an earlier point in program execution via *AutoFutures* possible (~ every 45.000 LOC)
  - Informal Pattern 3: Function extraction
    - Criterion 1: Statement block with internal data dependencies
    - Criterion 2: No write accesses outside
    - Parallel execution of statement blocks as *AutoFutures* (~ every 2.800 LOC)
  - Informal Pattern 4: Iterator parallelization
    - Criterion 1: Loop without iteration-bound dependencies
    - Typical case: forech-loops
    - Parallelization via scheduling iterations as *AutoFutures* (~ every 760 LOC)
- Future work:
  - Implementation of found transformations
  - Add dynamic analysis information

Thanks for your attention!

Any questions?