UNIVERSITÄT
LEIPZIG

11th Symposium on Software Performance

Graph-Based Performance Analysis at System- and Application-Level

Leipzig, November 13, 2020

Richard Müller and Tom Strempel

KIEKER

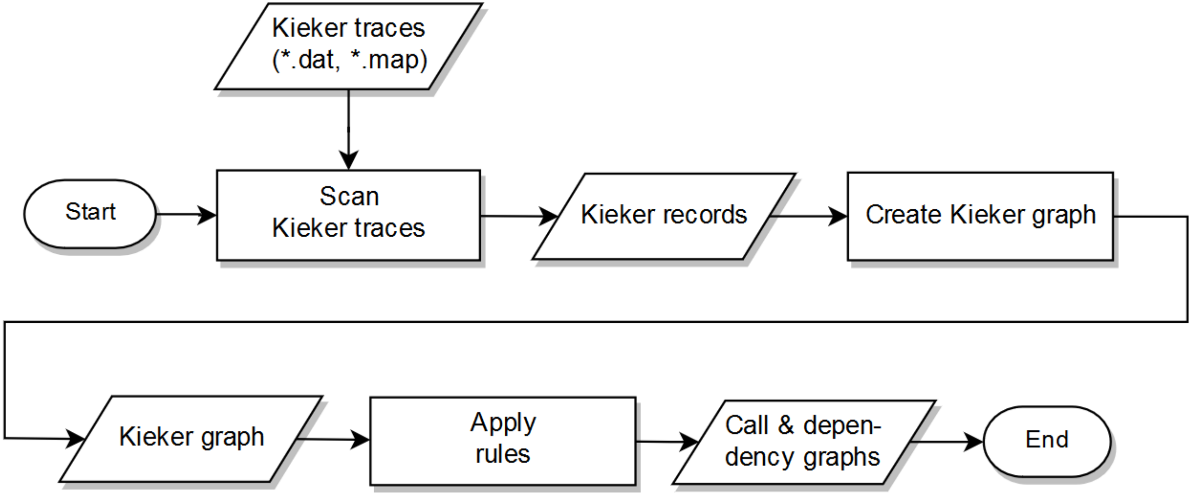
- The Kieker framework provides
  - monitoring,
  - analysis,
  - and visualization support

  for
  - application and system performance analysis as well as
  - reverse engineering

[Hasselbring and van Hoorn 2020, http://kieker-monitoring.net/]

KIEKER PLUGIN

- Transforms monitored log data into graphs
- Supports software engineers with performance analysis and architecture discovery



[Müller and Fischer 2019, https://github.com/softvis-research/jqa-kieker-plugin]

KIEKER PLUGIN ISSUES

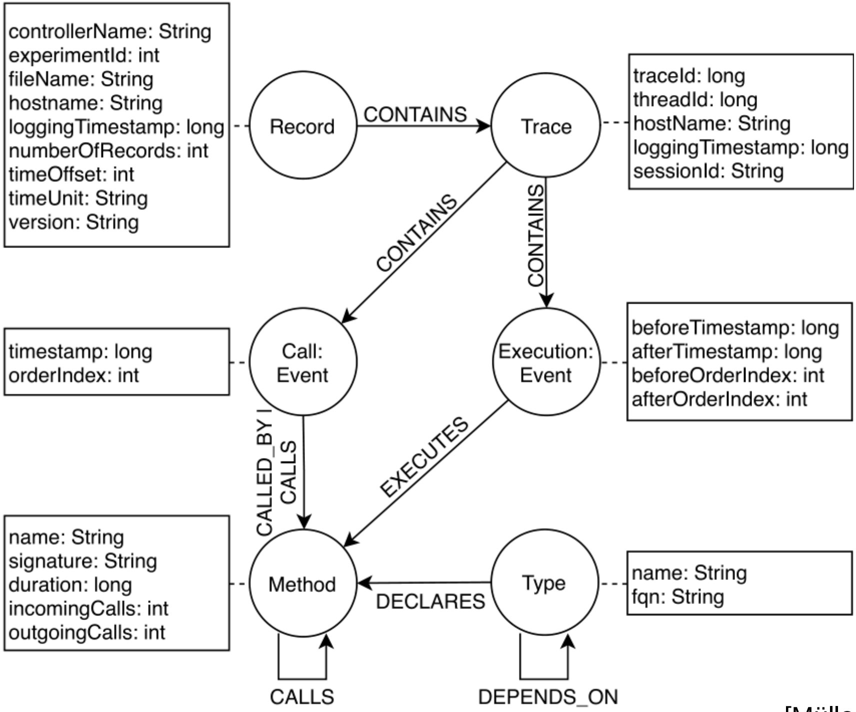
- No support for system-level information, such as CPU and system memory utilization
- High disk usage due to redundant information in the graph schema
- Long scan times due to implementation flaws
- No evaluation with regard to scalability

CONTRIBUTIONS

- Kieker plugin was extended and improved to solve the aforementioned issues
- Correctness and scalability of the revised plugin were evaluated by
  - processing data and
  - reproducing analysis results

  of two recent experiments
- A reproduction package is provided to replicate the evaluation: https://github.com/softvis-research/SSP2020

OLD KIEKER GRAPH SCHEMA



controllerName: String
experimentId: int
fileName: String
hostname: String
loggingTimestamp: long
numberOfRecords: int
timeOffset: int
timeUnit: String
version: String

traceId: long
threadId: long
hostName: String
loggingTimestamp: long
sessionId: String

timestamp: long
orderIndex: int

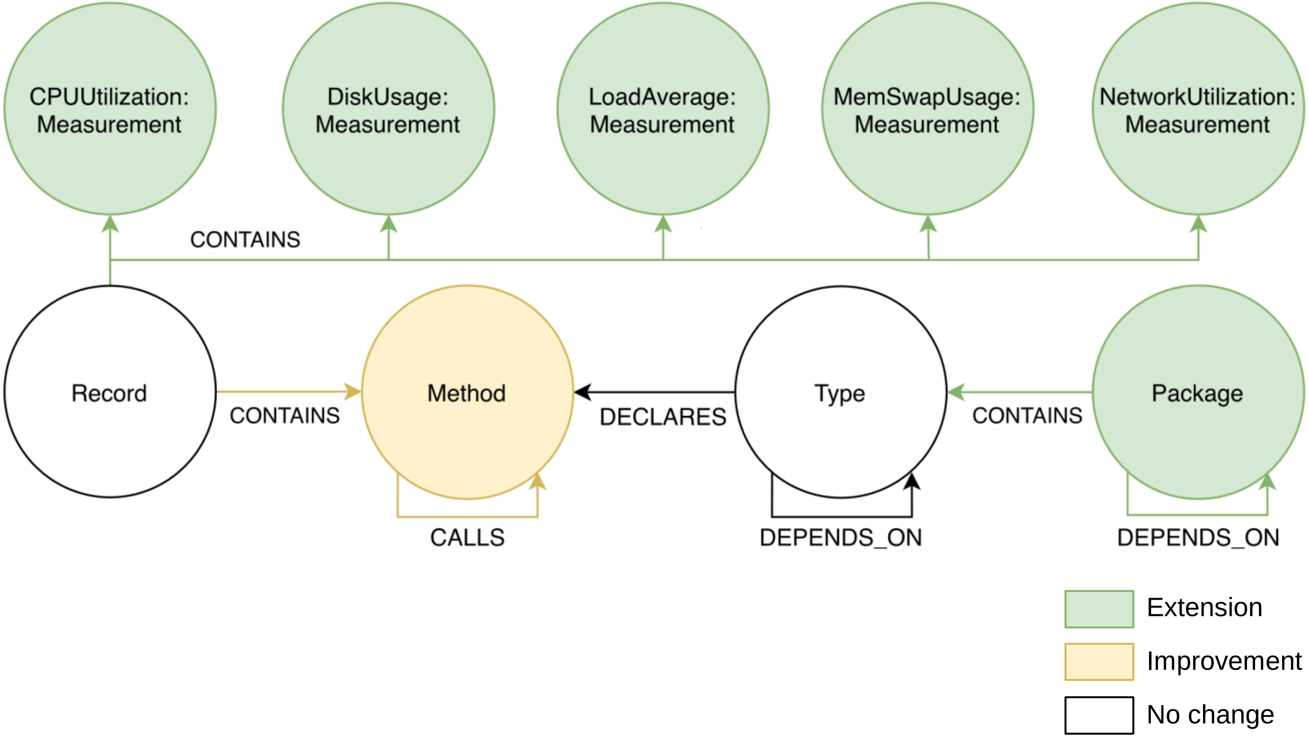
beforeTimestamp: long
afterTimestamp: long
beforeOrderIndex: int
afterOrderIndex: int

name: String
signature: String
duration: long
incomingCalls: int
outgoingCalls: int

name: String
fqn: String

[Müller and Fischer 2019]

REVISED KIEKER GRAPH SCHEMA



1ST EXPERIMENT - HORA: ARCHITECTURE-AWARE ONLINE FAILURE PREDICTION

- Combine component failure predictors with architectural knowledge to improve failure prediction

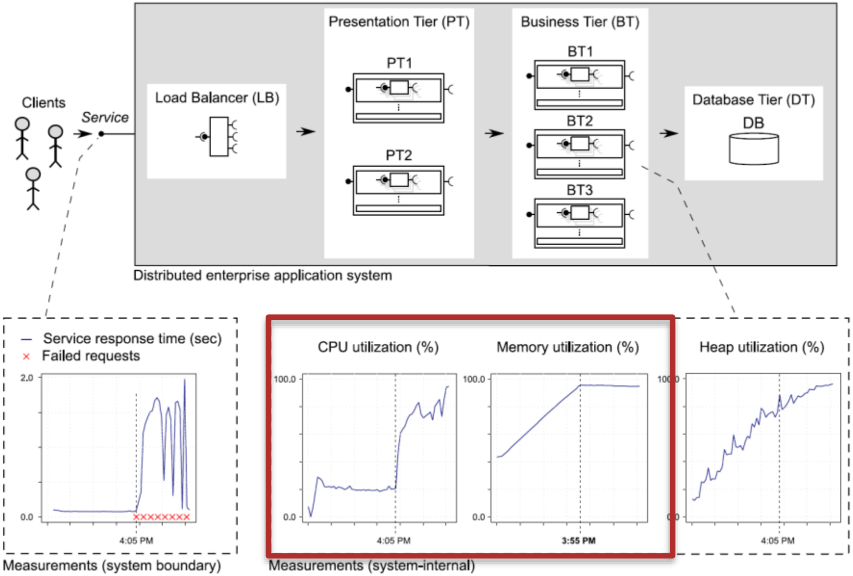
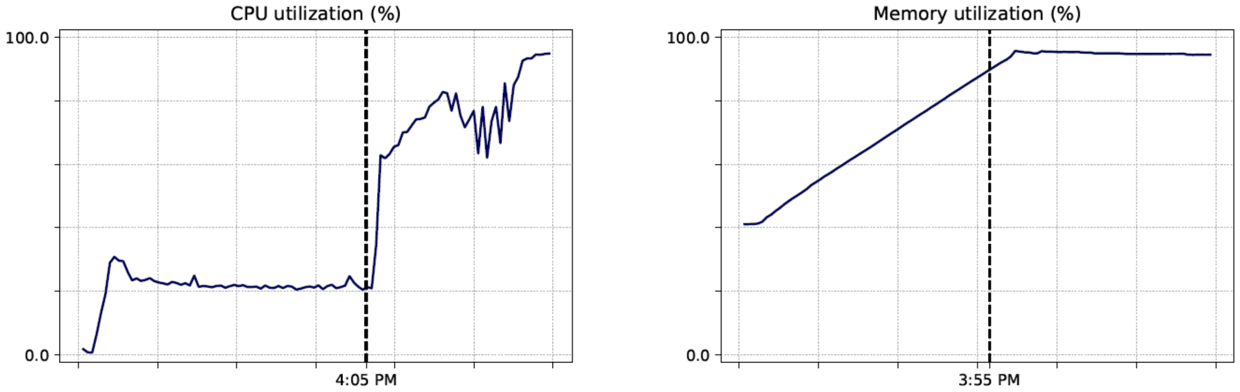


Fig. 1. Running example: high-level three-tier architecture and selected measurements.

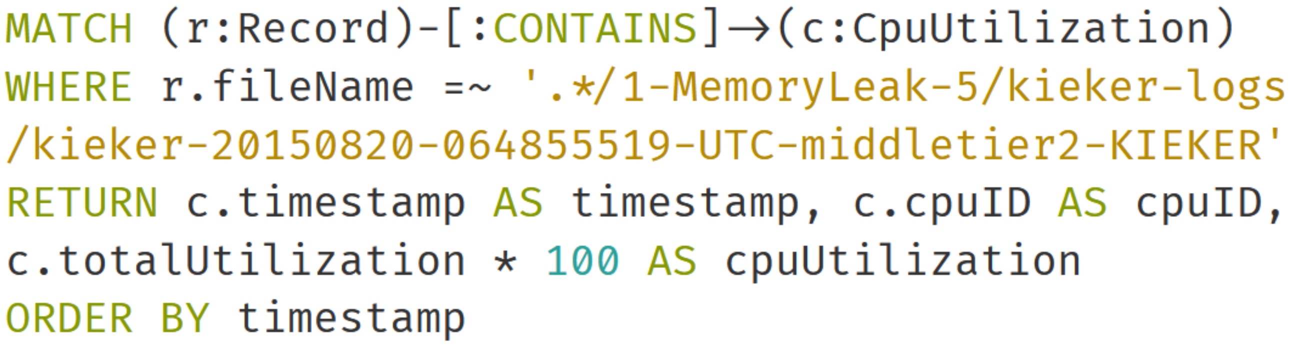
[Pitakrat et al. 2018]

PERFORMANCE ANALYSIS AT SYSTEM-LEVEL

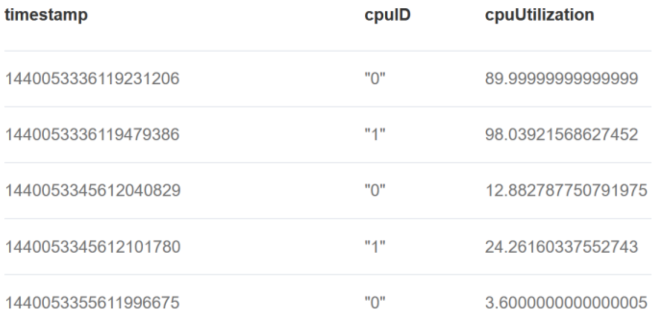
- Reproduce two line charts showing the system-level measures CPU and system memory utilization of the second business-tier instance from the first experiment [Pitakrat et al. 2018]



CYPHER QUERY FOR CPU UTILIZATION

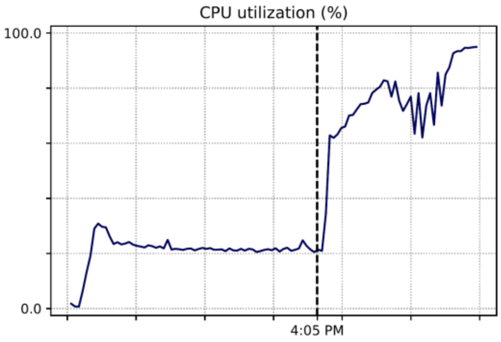


```
MATCH (r:Record)-[:CONTAINS]->(c:CpuUtilization)
WHERE r.fileName =~ '.*/1-MemoryLeak-5/kieker-logs
/kieker-20150820-064855519-UTC-middletier2-KIEKER'
RETURN c.timestamp AS timestamp, c.cpuID AS cpuID,
c.totalUtilization * 100 AS cpuUtilization
ORDER BY timestamp
```



timestamp	cpuID	cpuUtilization
1440053336119231206	"0"	89.99999999999999
1440053336119479386	"1"	98.03921568627452
1440053345612040829	"0"	12.882787750791975
1440053345612101780	"1"	24.26160337552743
1440053355611996675	"0"	3.6000000000000005

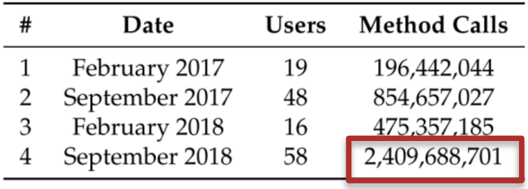
➔



2ND EXPERIMENT - COMPARING STATIC AND DYNAMIC WEIGHTED SOFTWARE COUPLING METRICS

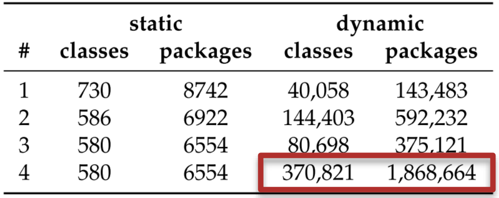
- Investigate how weighted dynamic coupling measurements can support software engineers to evaluate the architectural quality of software systems

Table 1. Numbers of users and monitored calls.



#	Date	Users	Method Calls
1	February 2017	19	196,442,044
2	September 2017	48	854,657,027
3	February 2018	16	475,357,185
4	September 2018	58	2,409,688,701

Table 10. Average Coupling Degrees in our four Experiments.



	static		dynamic	
#	classes	packages	classes	packages
1	730	8742	40,058	143,483
2	586	6922	144,403	592,232
3	580	6554	80,698	375,121
4	580	6554	370,821	1,868,664

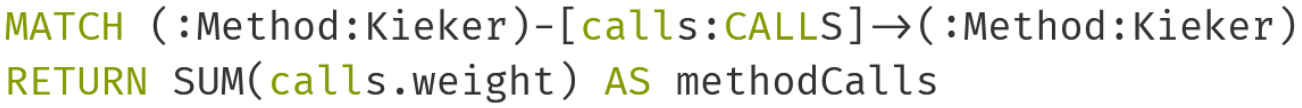
[Schnoor and Hasselbring 2020]

PERFORMANCE ANALYSIS AT APPLICATION-LEVEL

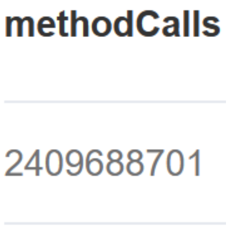
- Plugin processes 2,409,688,701 method calls and reproduces the weighted dynamic dependency graphs at class and package level from the second experiment [Schnoor and Hasselbring 2020]
- Disk usage
  - Original tar.xz file: 8.89 GB
  - Graph database: 110 MB*
- Scan and graph creation time
  - 1h 38min 29s

* This reduction is mainly due to omitting the node types `Event` and `Trace` including their properties.

CYPHER QUERY FOR METHOD CALLS



```
MATCH (:Method:Kieker)-[calls:CALLS]→(:Method:Kieker)
RETURN SUM(calls.weight) AS methodCalls
```



methodCalls
2409688701

=

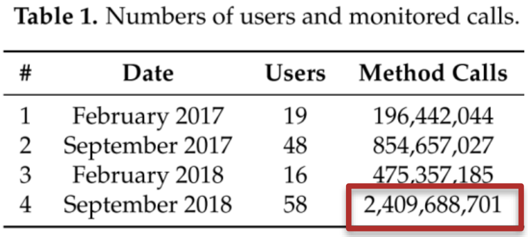
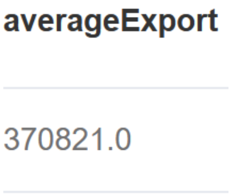


Table 1. Numbers of users and monitored calls.

#	Date	Users	Method Calls
1	February 2017	19	196,442,044
2	September 2017	48	854,657,027
3	February 2018	16	475,357,185
4	September 2018	58	2,409,688,701

CYPHER QUERY FOR AVERAGE EXPORT COUPLING DEGREE ON CLASS LEVEL

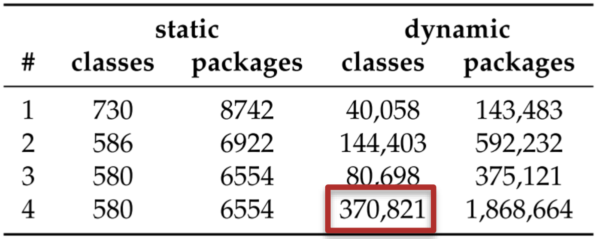
```
MATCH (t:Type:Kieker)
WHERE (t)-[:DEPENDS_ON]→() OR ()-[:DEPENDS_ON]→(t)
WITH t
OPTIONAL MATCH (t)-[out:DEPENDS_ON]→()
WITH t, SUM(out.weight) AS import
OPTIONAL MATCH ()-[in:DEPENDS_ON]→(t)
WITH t, import, SUM(in.weight) AS export
RETURN ROUND(AVG(export)) AS averageExport
```



averageExport
370821.0

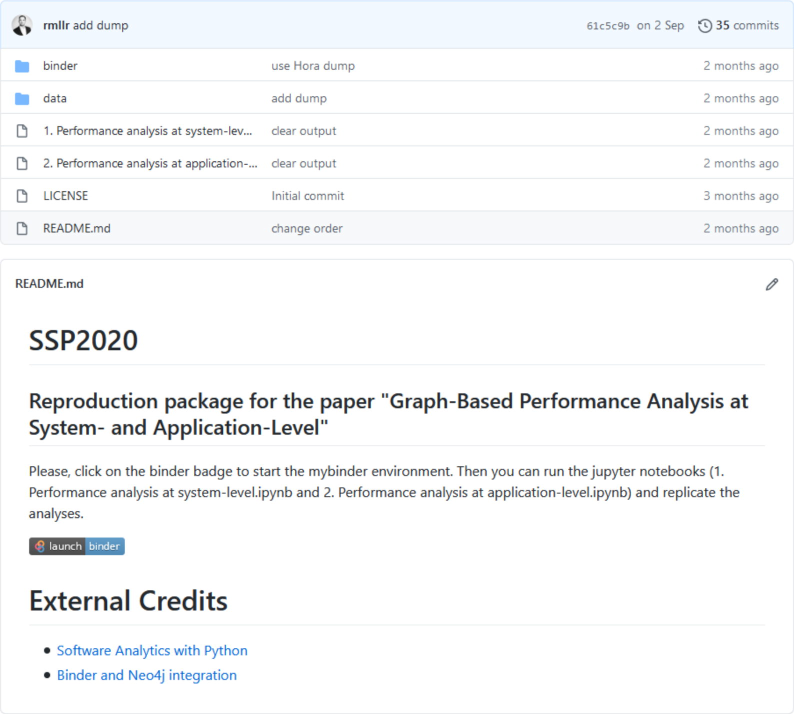
=

Table 10. Average Coupling Degrees in our four Experiments.



	static		dynamic	
#	classes	packages	classes	packages
1	730	8742	40,058	143,483
2	586	6922	144,403	592,232
3	580	6554	80,698	375,121
4	580	6554	370,821	1,868,664

REPRODUCTION PACKAGE



rmllr add dump 61c5c9b on 2 Sep 35 commits

binder	use Hora dump	2 months ago
data	add dump	2 months ago
1. Performance analysis at system-lev...	clear output	2 months ago
2. Performance analysis at application-...	clear output	2 months ago
LICENSE	Initial commit	3 months ago
README.md	change order	2 months ago

README.md

SSP2020

Reproduction package for the paper "Graph-Based Performance Analysis at System- and Application-Level"

Please, click on the binder badge to start the mybinder environment. Then you can run the jupyter notebooks (1. Performance analysis at system-level.ipynb and 2. Performance analysis at application-level.ipynb) and replicate the analyses.

External Credits

- Software Analytics with Python
- Binder and Neo4j integration

[https://github.com/softvis-research/SSP2020]

FUTURE WORK

- Replicate the complete experiment from [Schnoor and Hasselbring 2020]
- Kieker plugin will be used to generate dynamic dependency graphs
- Java bytecode scanner plugin will be used to generate static dependency graphs

[https://github.com/jQAssistant/jqa-java-plugin]

REFERENCES

- W. Hasselbring and A. van Hoorn. "Kieker: A monitoring framework for software engineering research". In: Software Impacts 5 (Aug. 2020), pp. 1-5.
- R. Müller and M. Fischer. "Graph-Based Analysis and Visualization of Software Traces". In: 10th Symposium on Software Performance: Joint Developer and Community Meeting of Descartes/Kieker/Palladio. Würzburg, Germany, 2019.
- T. Pitakrat et al. "Hora: Architecture-aware online failure prediction". In: Journal of Systems and Software 137 (2018), pp. 669-685.
- H. Schnoor and W. Hasselbring. "Comparing Static and Dynamic Weighted Software Coupling Metrics". In: Computers 9.2 (Mar. 2020), p. 24.

UNIVERSITÄT
LEIPZIG

THANK YOU.

Richard Müller
Information Systems Institute, Chair of Software Engineering, Leipzig University
Tom Strempel
Master student in Computer Science, Leipzig University

✉ rmueller@wifa.uni-leipzig.de
@rimllr
https://github.com/softvis-research
http://softvis.wifa.uni-leipzig.de