Course Testing Techniques: An industrial perspective on model learning

Dennis Hendriks

December 16, 2022

Radboud University Nijmegen

ESI

ASML

Outline

1. Introduction
2. Industrial challenges
3. Transposition applied-research project
4. Active automata learning in industry
   - Scalability
   - Application
5. Preventing regressions for software changes
6. Conclusions

1. Introduction

Dennis Hendriks

TNO ESI

Senior Research Fellow
4 days/week

Radboud Universiteit

Researcher
1 day/week

ASML

Lead scientist Transposition applied-research project

TU/e Radboud University Nijmegen University of Stuttgart

Collaborations via PhD/MSc students

TNO-ESI: a joint-innovation center of industry, academia and TNO

Mission: Embedding leading edge methodologies into the Dutch high-tech systems industry to cope with the ever-increasing complexity of their products.

Synopsis

- ESI started in 2002
- ESI integrated in TNO (unit ICT) per January 2013
- ~55 staff members, many with extensive industrial experience
- 5 part-time Professors
- Working at industry locations: *"the industry as a lab"*

Focus

Managing complexity of high-tech systems

through

- system architecting,
- system reasoning and
- model-driven engineering

delivering

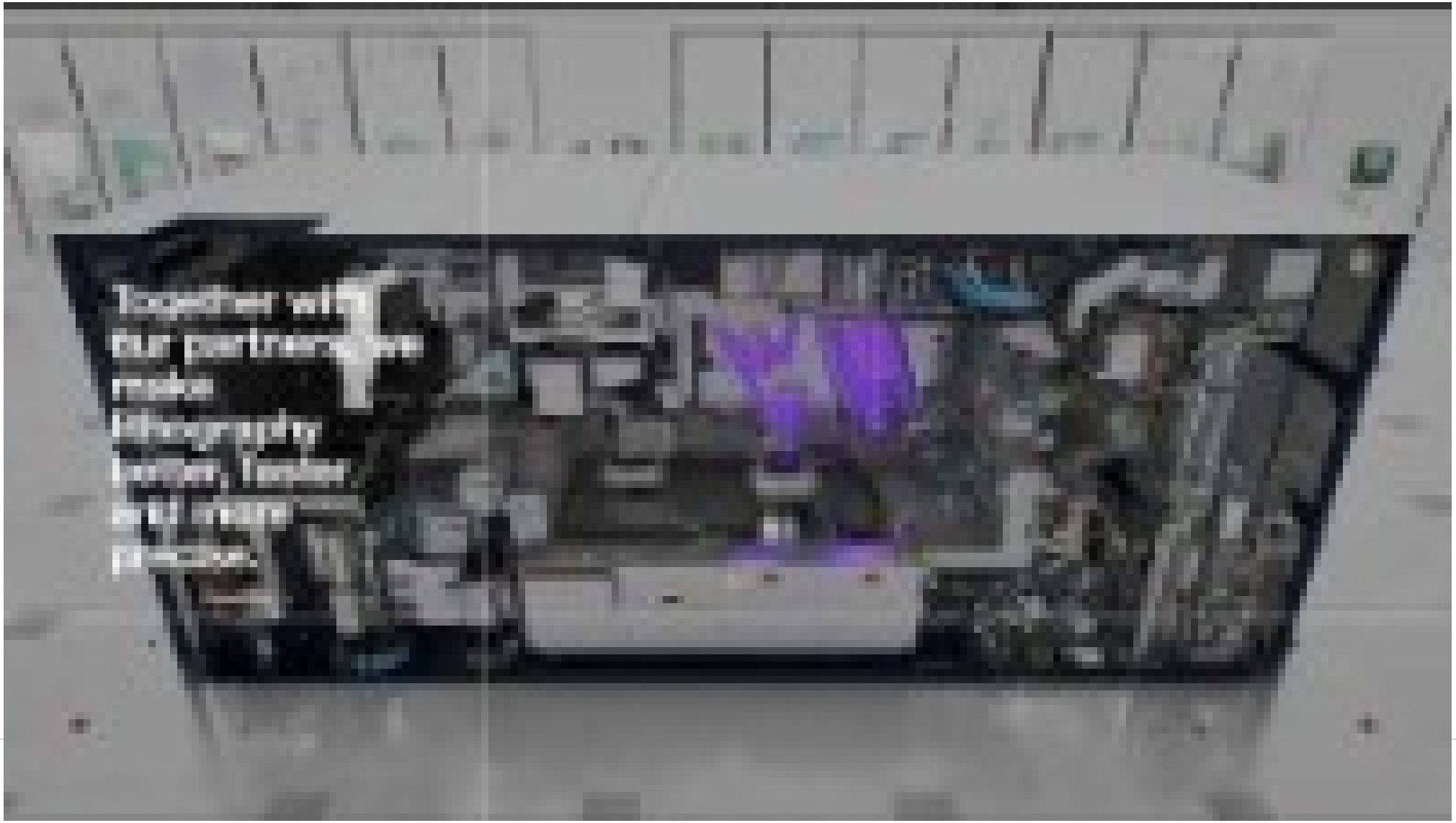
- methodologies validated in cutting-edge industrial practice

Partner Board

ASML, ThermoFisher Scientific, Philips, itec equipment + automation tech, Thales, Canon Canon Production Printing, TNO, University of Amsterdam, TU Delft Delft University of Technology, Radboud University Nijmegen, University of Twente., TU/e Eindhoven University of Technology

ASML in 1 minute

Source: https://www.youtube.com/watch?v=wI6nCmG-PpI



ASML: complex high-tech systems

NXE has over 100,000 individual parts, 3,000 cables, 40,000 bolts and 2 km of hosing…

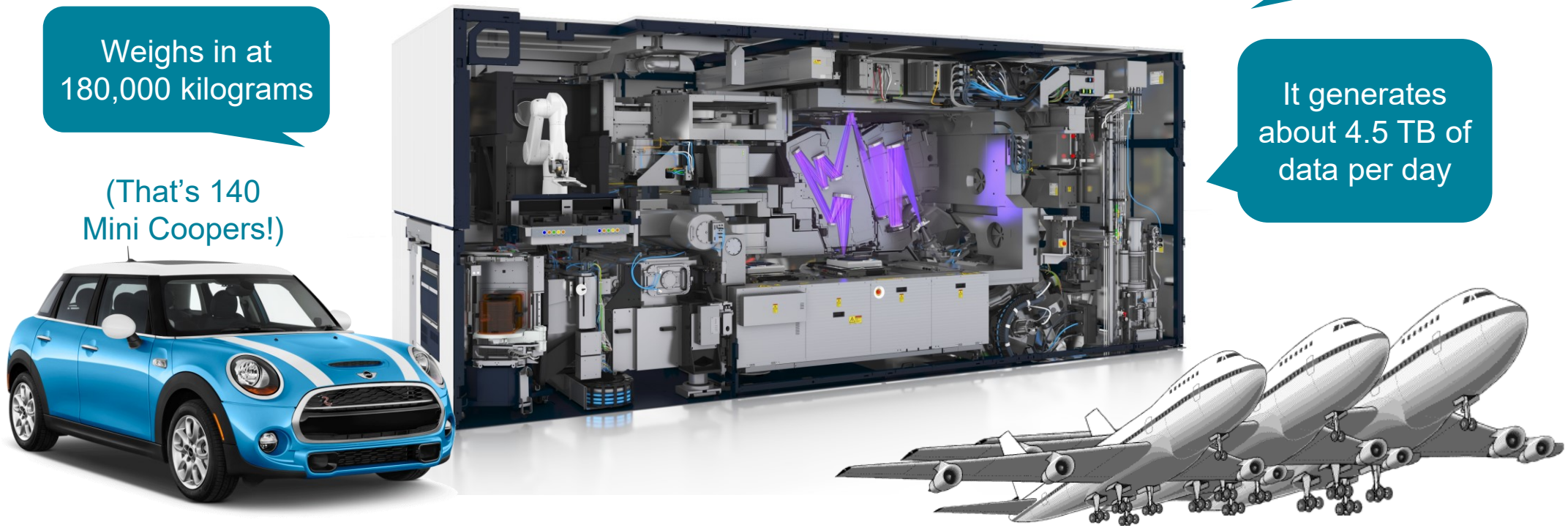
10s of millions of lines of code

Transportation takes 40 containers, 20 trucks and 3 fully loaded 747s

It has about 1,500 sensors to capture imaging data

Weighs in at 180,000 kilograms

(That's 140 Mini Coopers!)



It generates about 4.5 TB of data per day

ASML: complex high-tech systems

EUV mirrors are polished to an accuracy of ~50 picometers – less than the diameter of a silicon atom.

Blown up to the size of Germany, the biggest difference in height would be less than a millimeter.

We need to maintain a clean vacuum, but every time we expose a wafer, the photoresist releases trillions of particles

2. Industrial challenges: software behavior

Systems become more and more complex

1980s:

PAS 2000/5000

1990s:

PAS 5500

2000s:

TWINSCAN

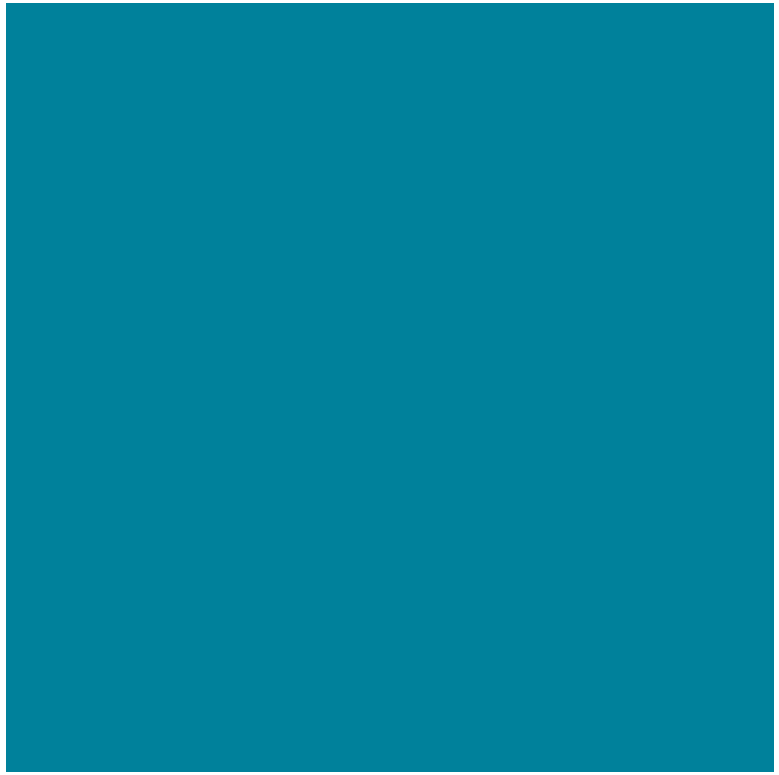
2010s:

EUV

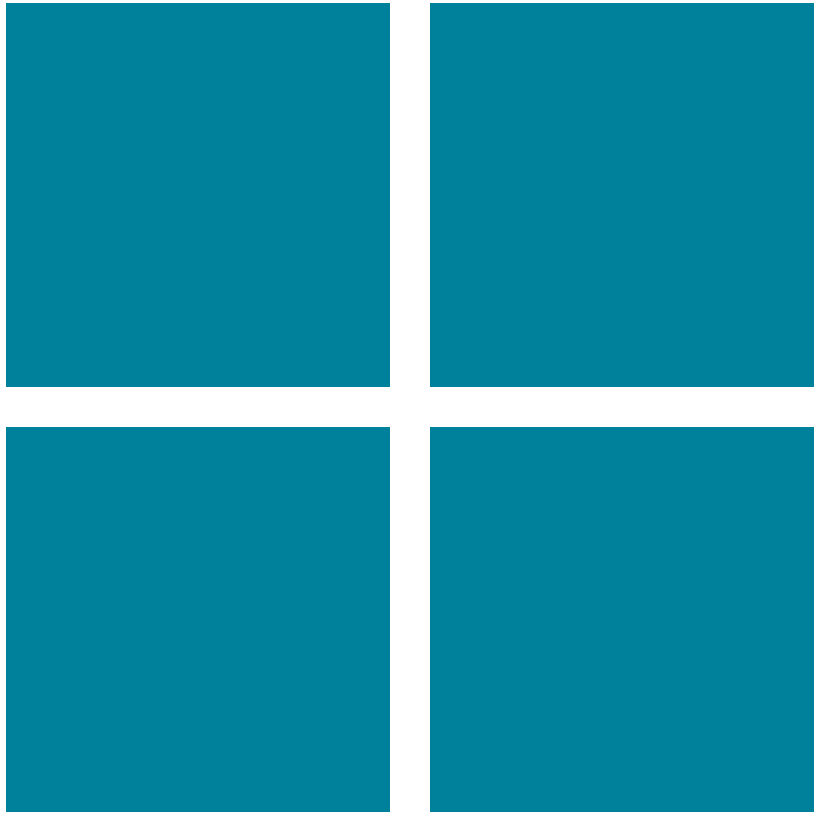
2020s:

High-NA EUV

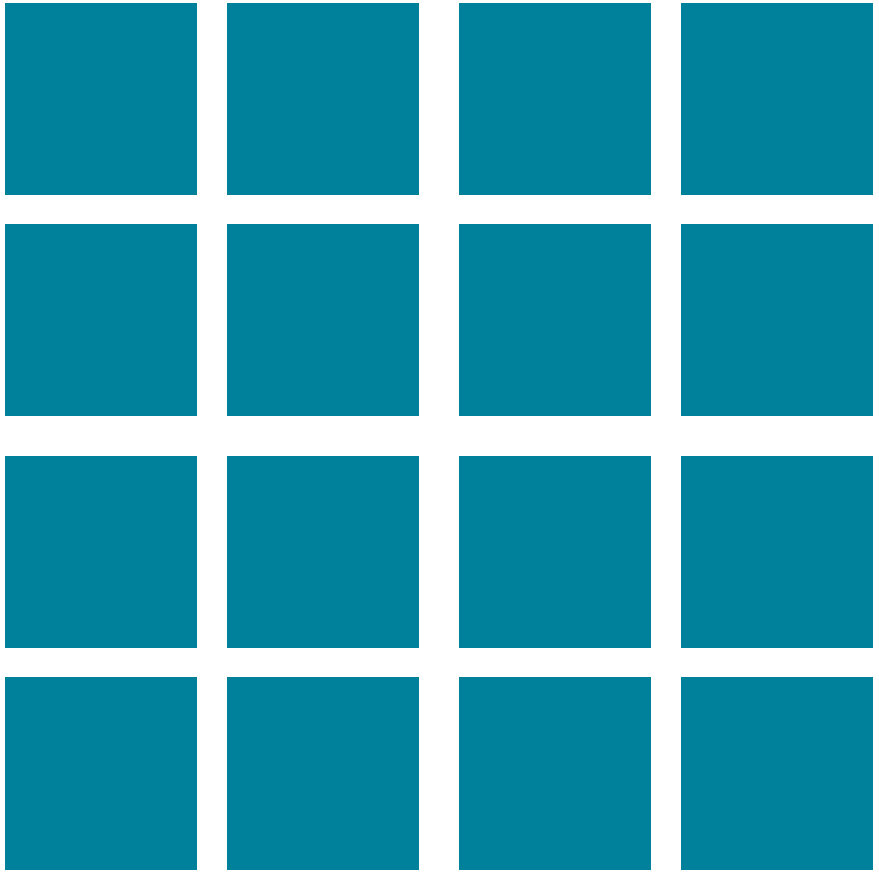
Divide and conquer: component-based software architecture



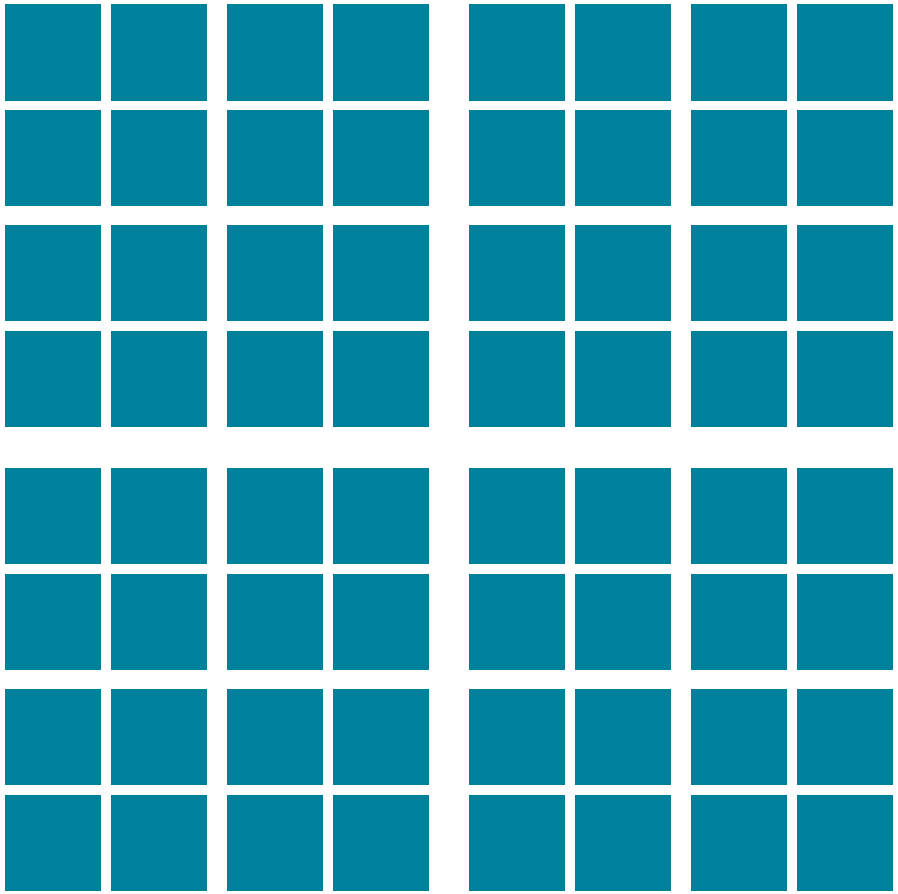
Divide and conquer: component-based software architecture



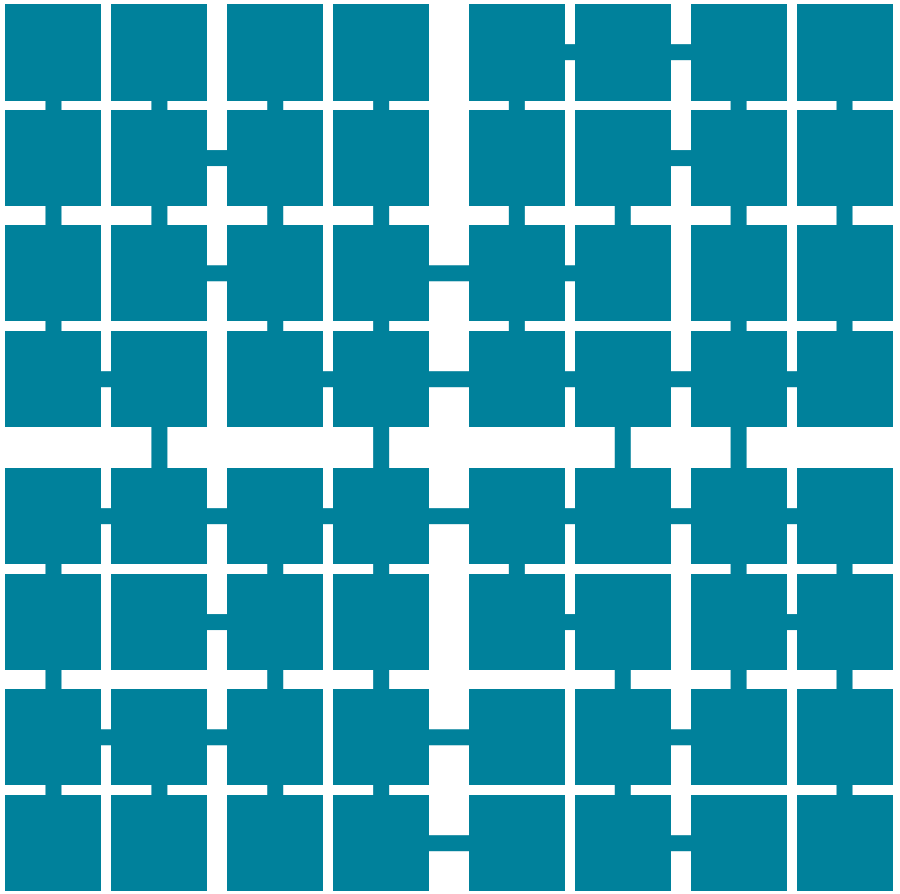
Divide and conquer: component-based software architecture



Divide and conquer: component-based software architecture



Components communicate via interfaces



Interfaces require contracts

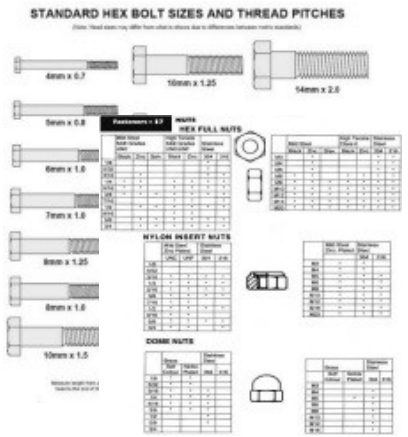
Interface contracts are common in many domains

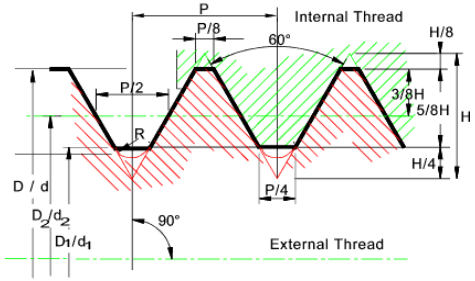
component
'bolt'

interface
'thread m6'

component
'nut'

1. Components need to fit together
2. Need standardized interfaces
3. Need to specify the interface contracts

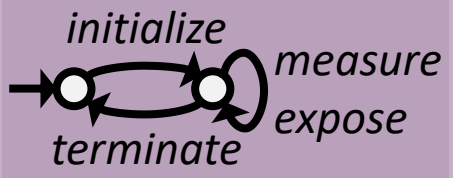
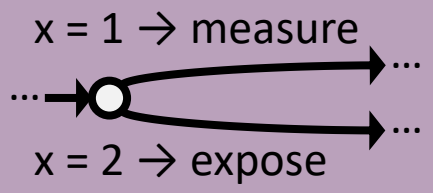
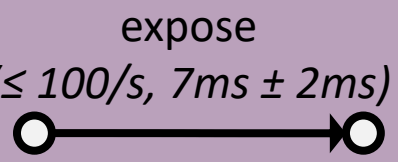




D /d =Major diameter (internal / external)
D_2/d_2 = Effective (Pitch) dia (internal /external)
D_1/d_1= Minor diameter ((internal /external)
D_2/d_2 = D - 0,64952P
D_1/d_1 = D - 1,08254P

Software components need to speak the same language

Interface contract

Syntax	Order constraints	Data constraints	Time constraints	Resource constraints
Function signatures (e.g. names, parameters, types)	Function call order (e.g. statement order in methods)	If/while guards (e.g. choices based on variable values)	Inter-call durations (e.g. min/max bound, average duration)	Platform limitations (e.g. CPU, memory, I/O limits)
initialize, measure, expose, terminate				*max 8 GB memory, max 1Gb/s disk I/O*

Software behavior models can be used for many purposes

Provide insight into software behavior

- Understand, communicate, discuss
- Cost-effective transition to Model-Driven Engineering (MDE)
- …

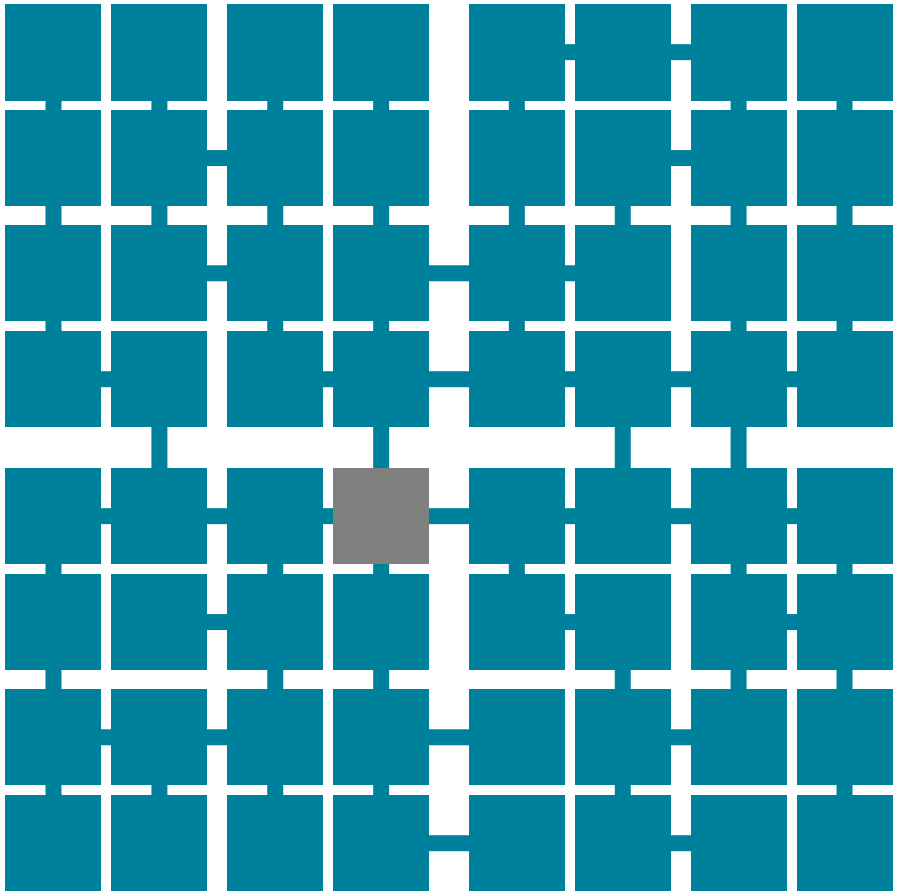
Reason about software behavior

- Model-Based Testing
- Formal verification
- Guard the architecture (monitoring/armoring)
- Change impact analysis
- …

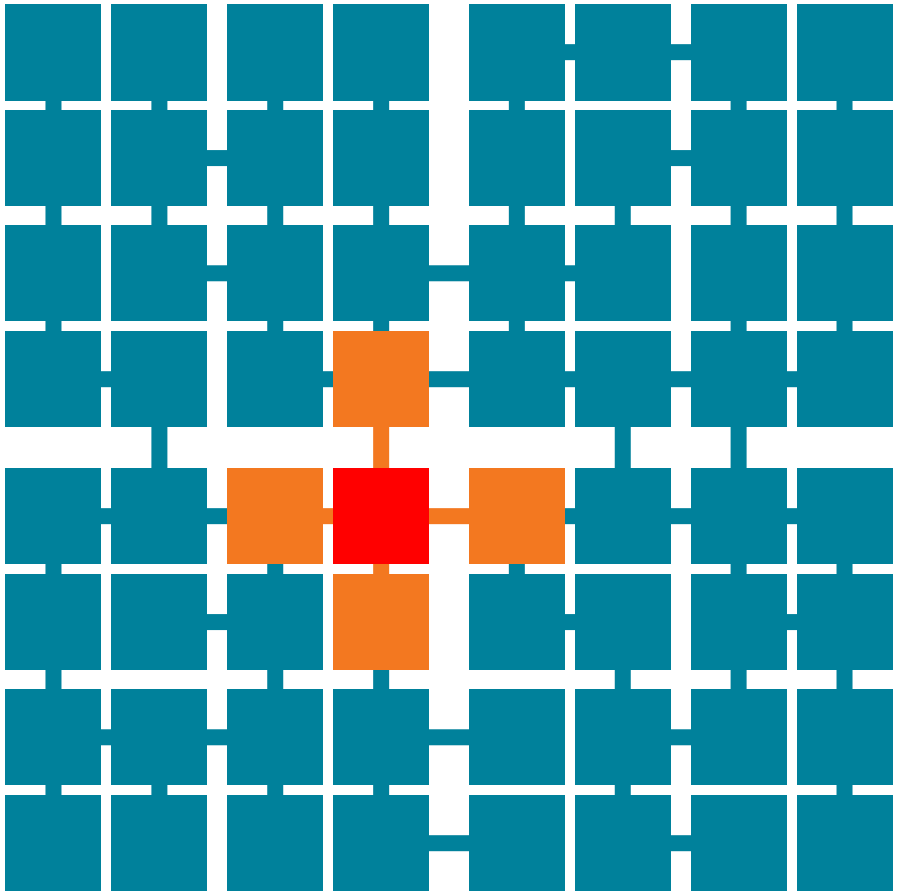
Industrial challenges: preventing regressions for software changes

Radboud University Nijmegen
ESI
ASML

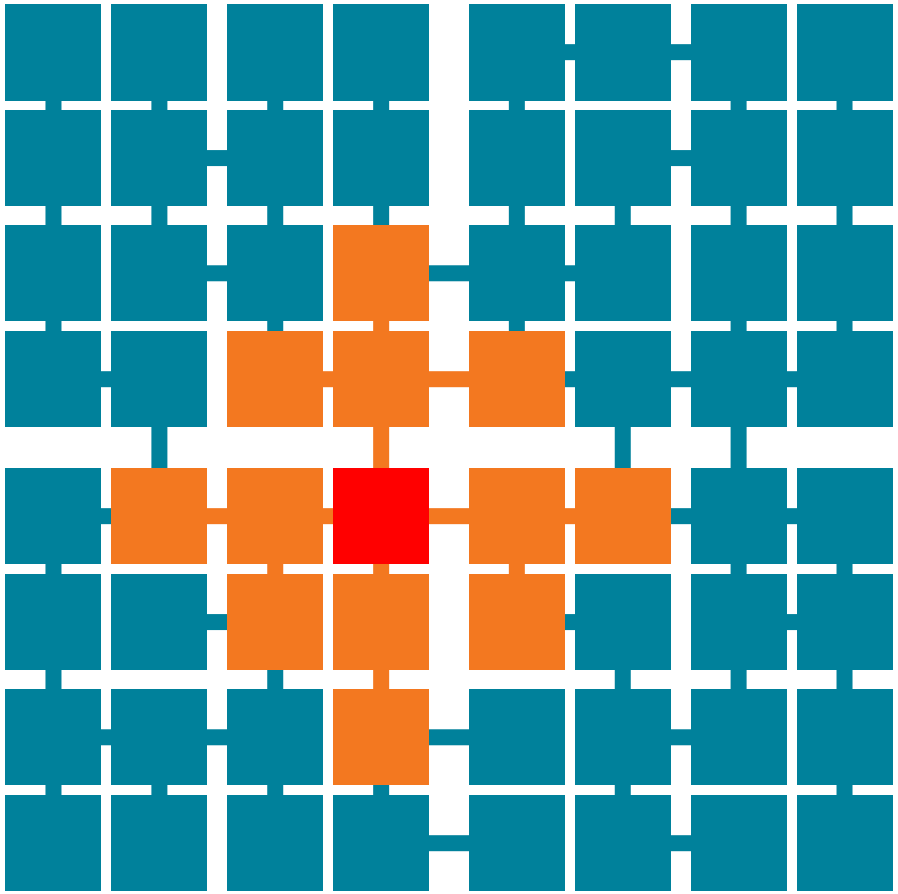
Software changes are risky



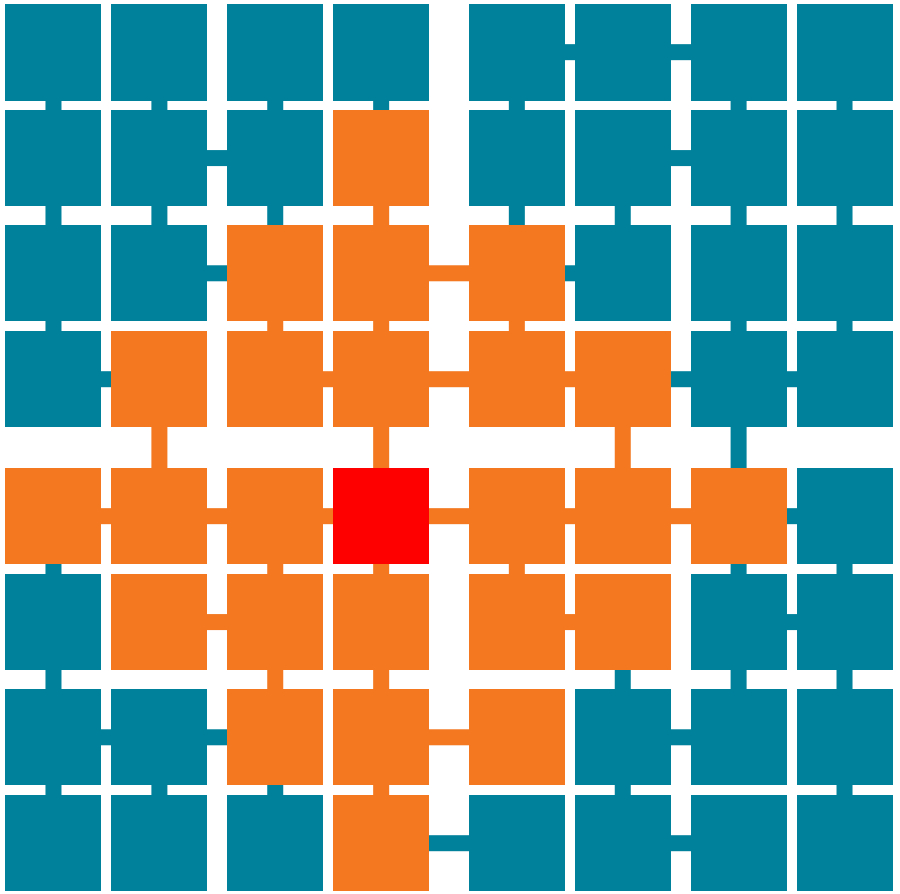
Software changes are risky



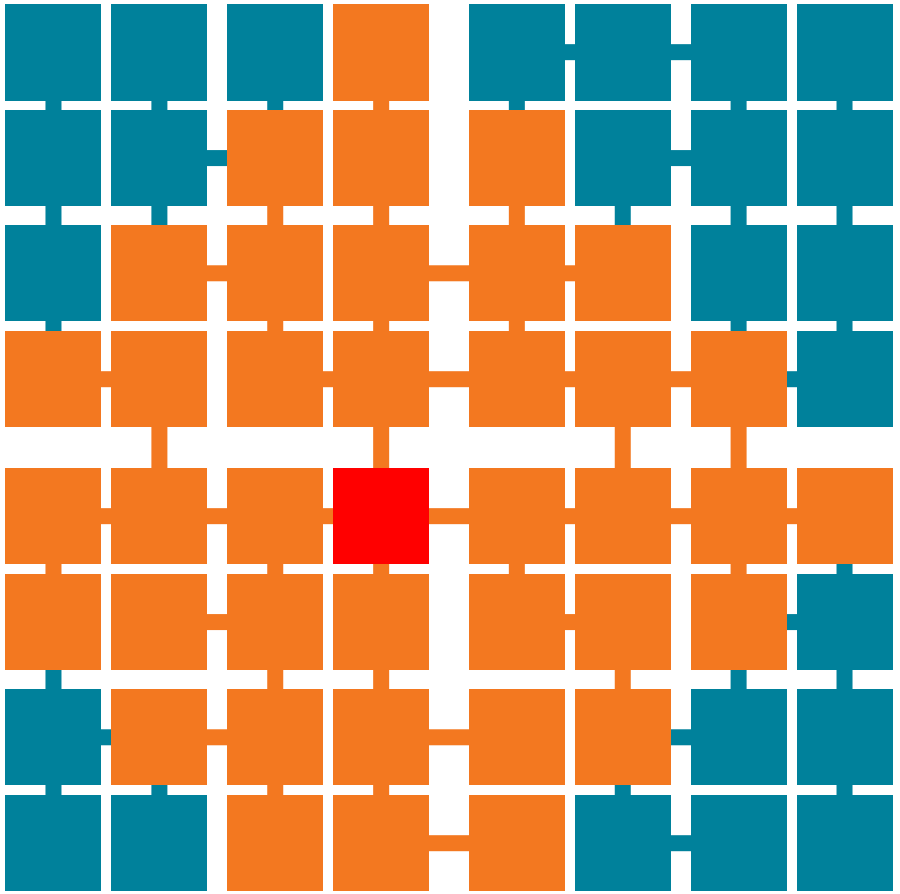
Software changes are risky



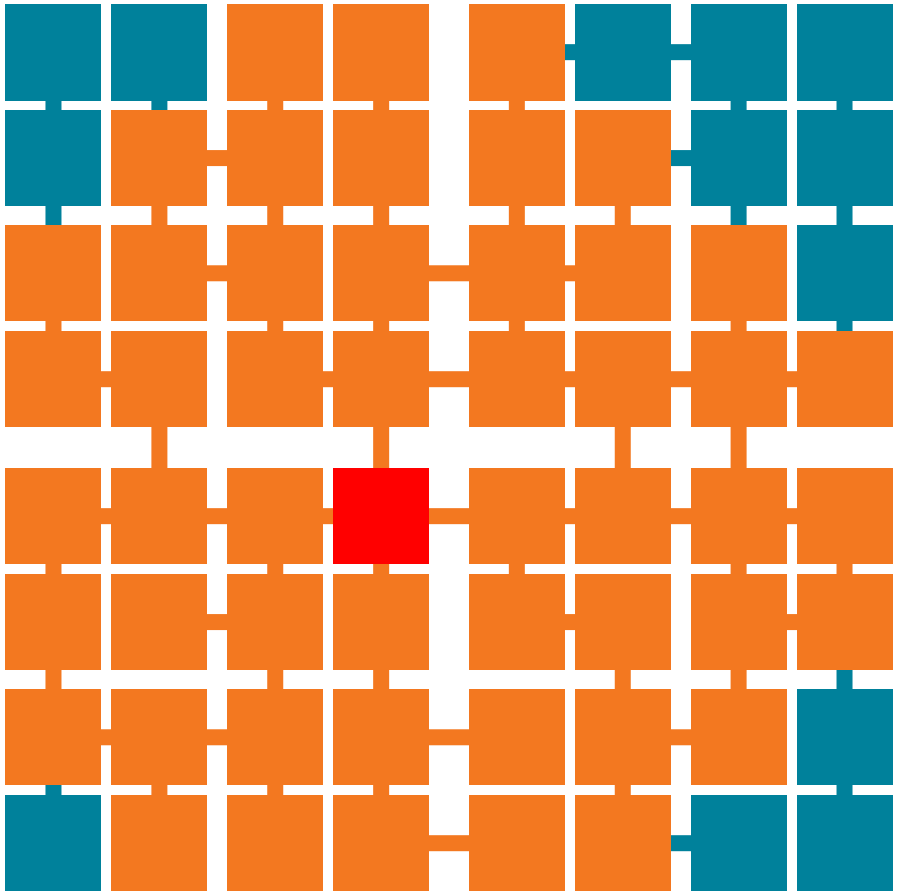
Software changes are risky



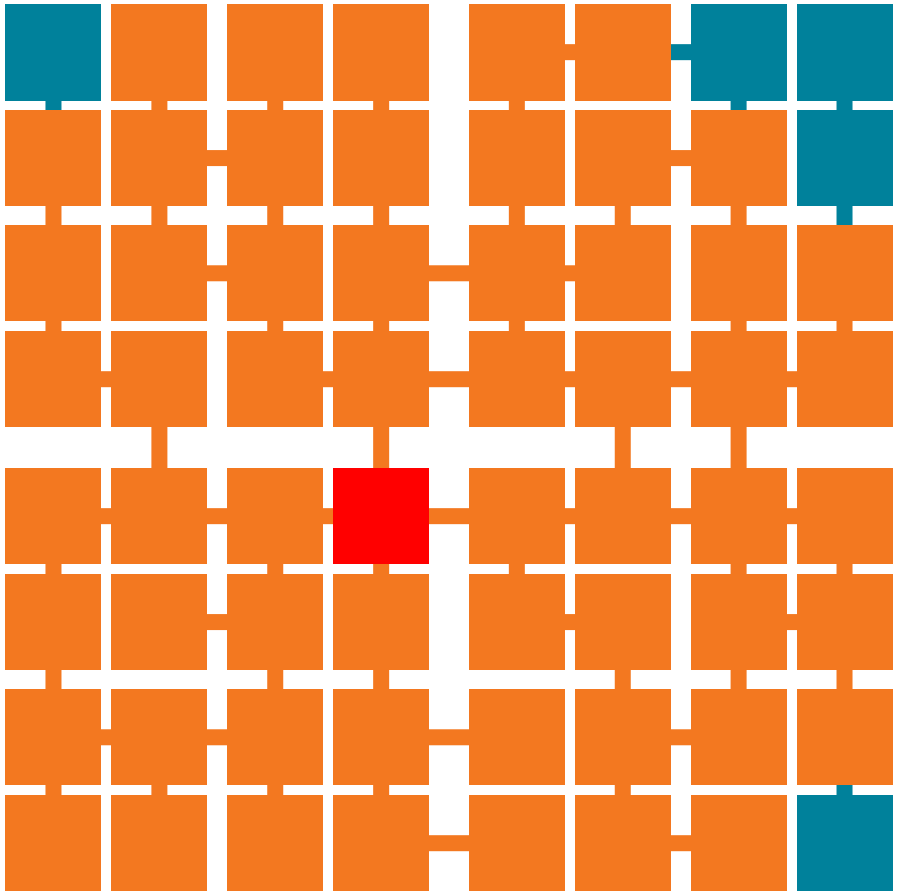
Software changes are risky



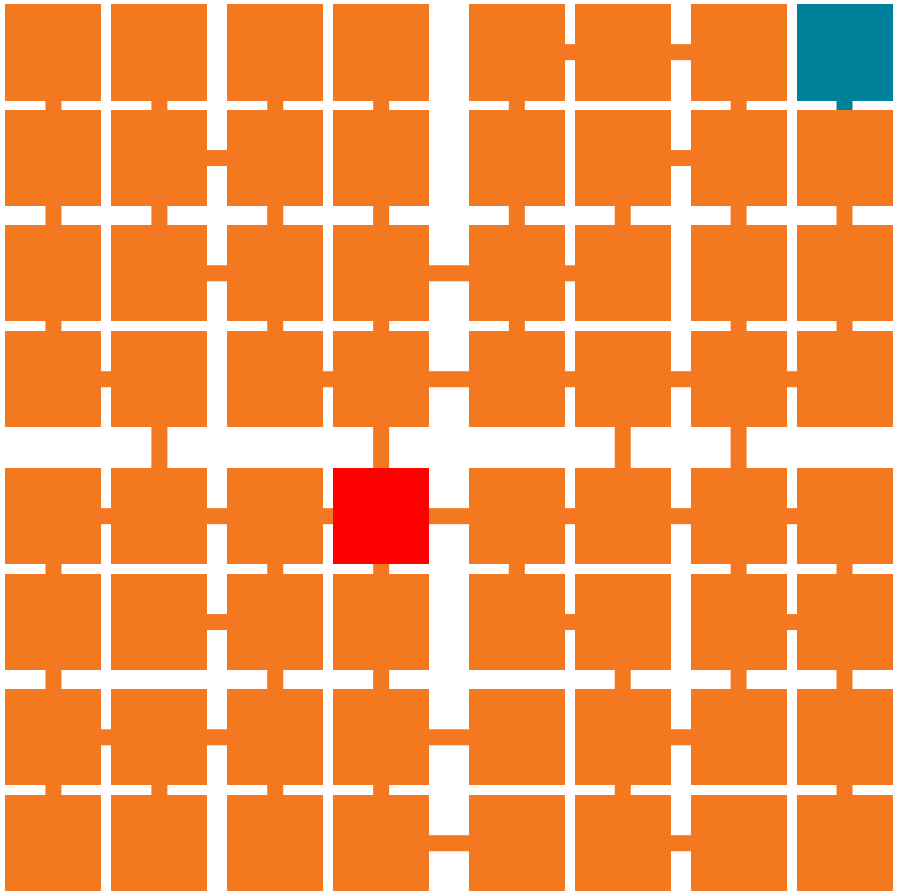
Software changes are risky



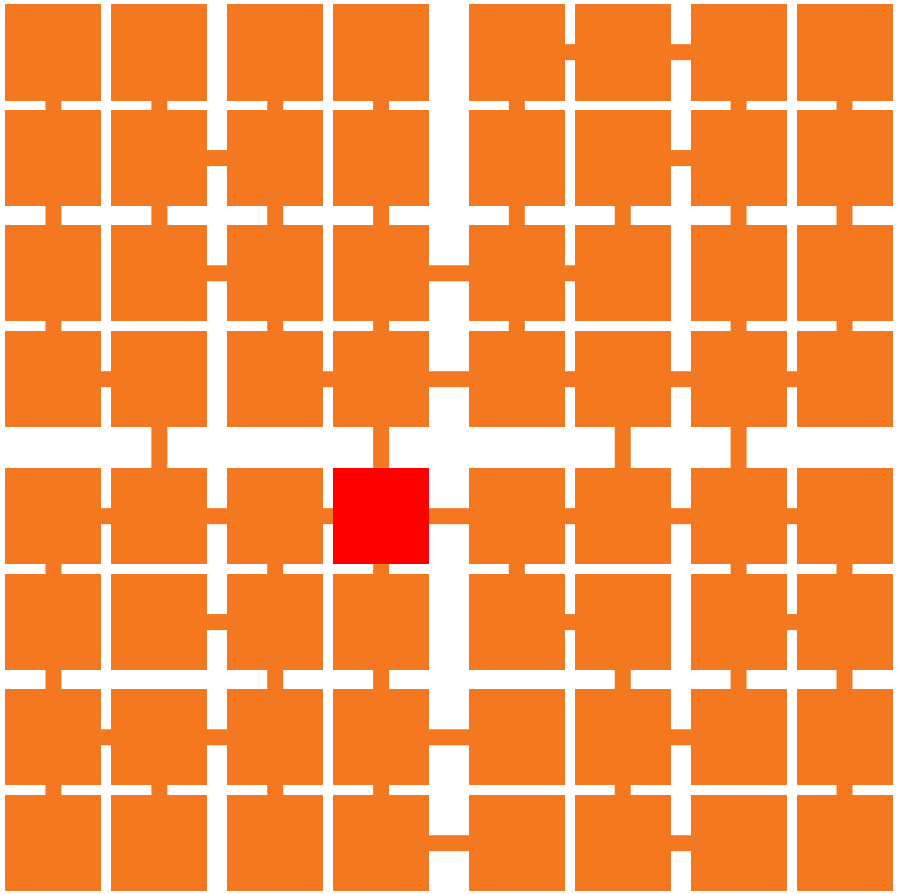
Software changes are risky



Software changes are risky



Software changes are risky



"Preventing regression impact for redesigns is critical for our customers."

Theo Engelen
(Head of ASML D&E Software Architecture)

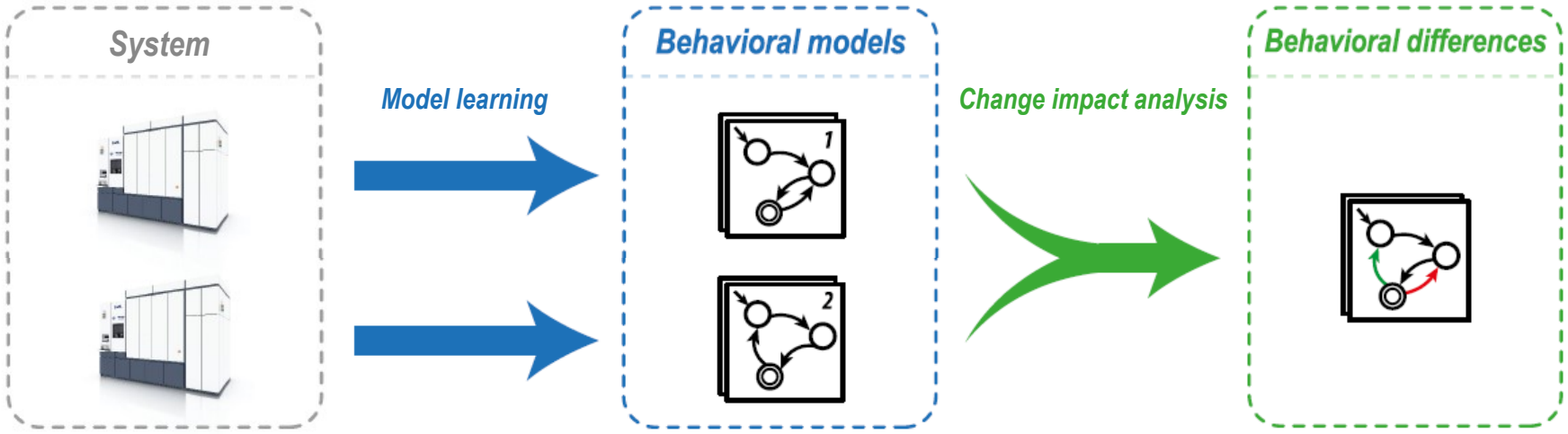
3. Transposition applied-research project

Radboud University Nijmegen

ESI

ASML

Transposition project: our approach

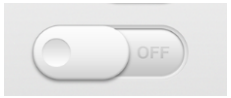


- Automatically infer behavioral models
- Provide insight into software behavior
- Essential for making correct software changes

- Automatically compare behavioral models
- Confirm expected differences and potential regressions
- Quick change impact analysis for system behavior

Improve development efficiency and reduce risks

Transposition project: different model learning techniques

Static analysis	Dynamic analysis	
		
		
Source Code Analysis	Passive Learning	Active Learning

Transposition project: the applied-research journey

2018	2019	2020	2021	2022

Active learning (scalability)
- How well does it scale in practice?
- Can we improve the scalability (enough)?

AL (application)
- How to connect to a learning tool?

Constructive Model Inference
- Can we efficiently infer intuitive models for all components of an ASML TWINSCAN system?

Change impact analysis
- Can we reduce the risks for software changes by preventing regressions?

Industrial roll-out
- What else is needed for ASML to employ all this in their daily practice?

4. Active automata learning in industry: Scalability

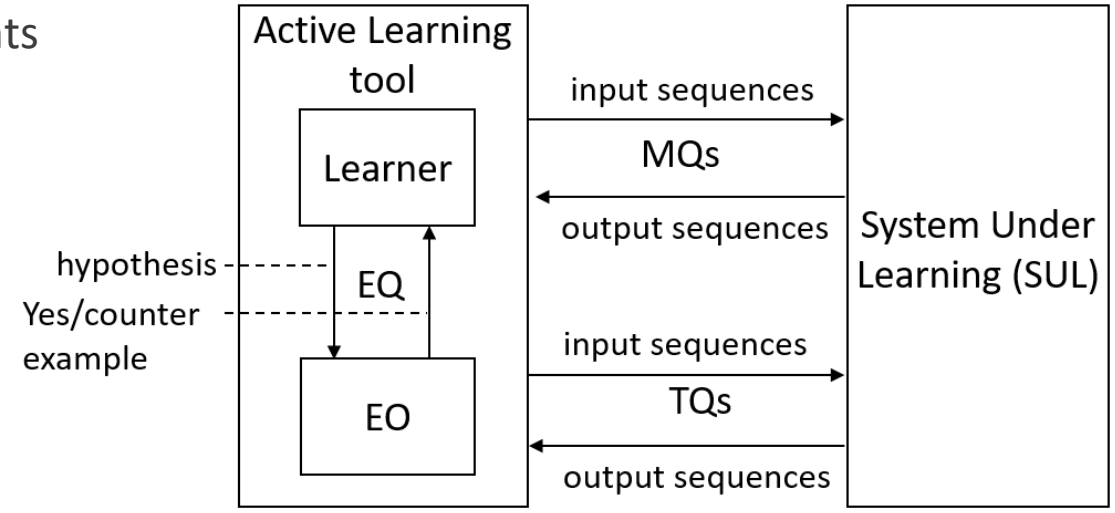
Applied research on scalability of active automata learning

Active (automata) learning is known to have scalability issues

- Investigations in literature often use generated models
- Or are limited to only a single real/industrial (sub)component

Goal

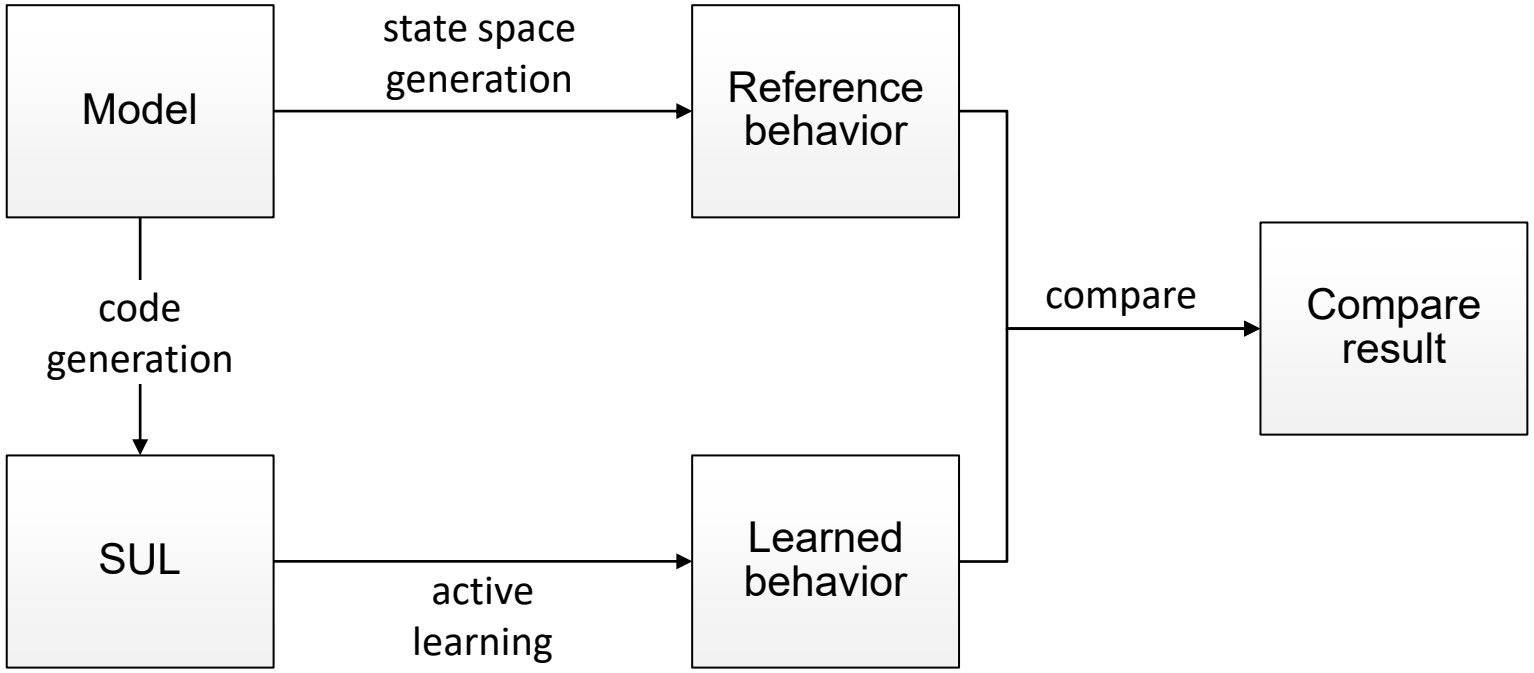
- See how well AL scales on large set of components
- Better understand scalability of AL
- Improve scalability of AL



Experimental approach

Experimental setup to apply active learning to MDE-based components

- Two independent toolchains
- Formal verification (behavioral comparison)



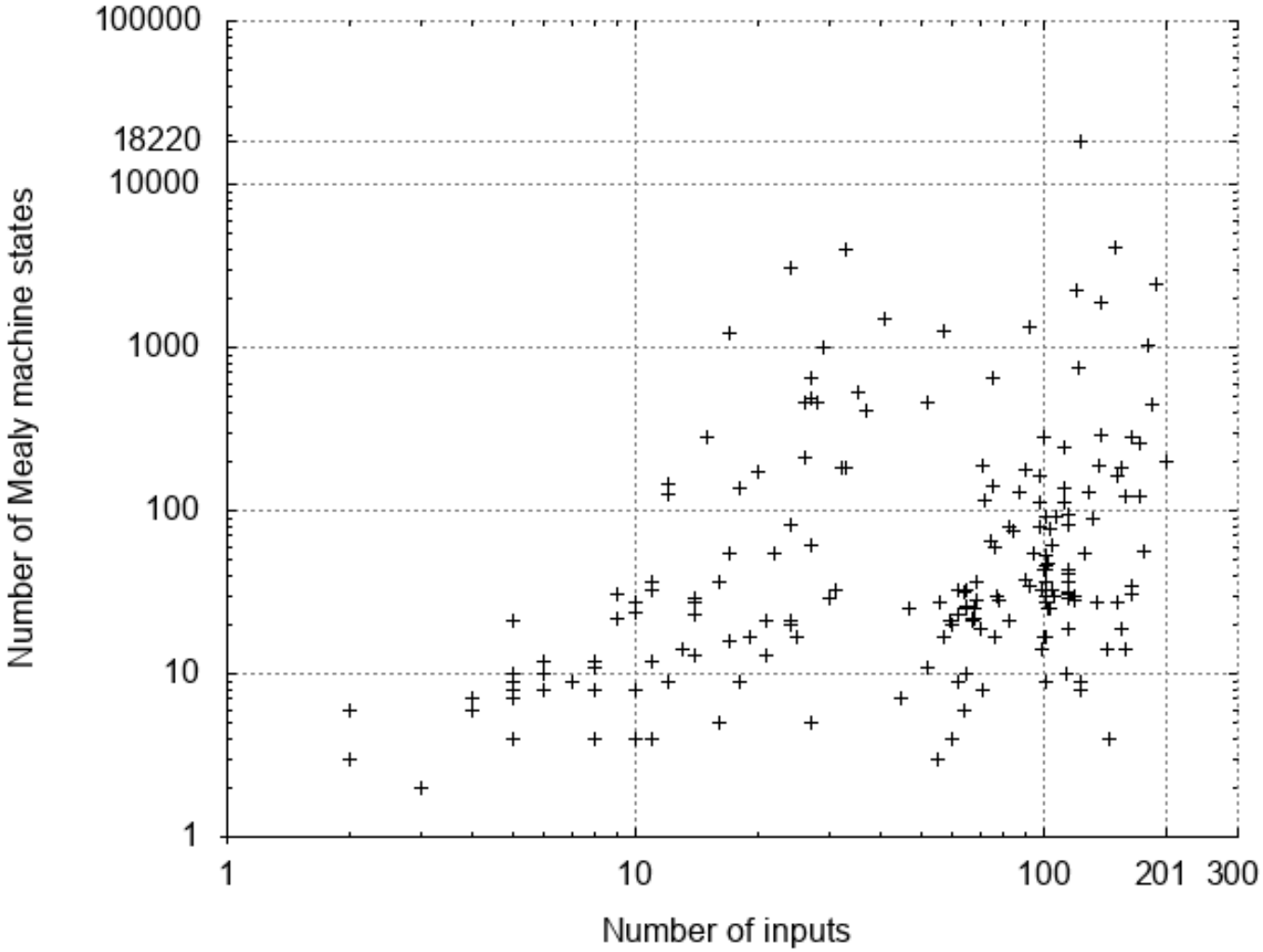
Input models

Validation

- Use of MDE-based components enables validation of the approach
- Applied to 218 real ASML sub-components
- Validation successful

Scalability

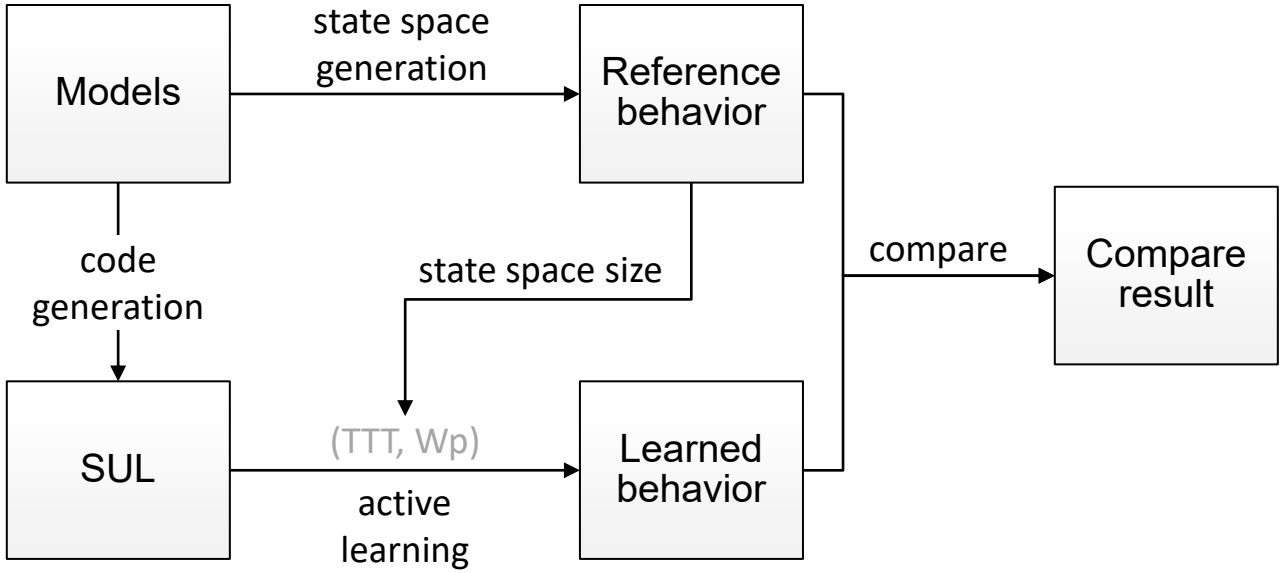
- 200+ models of various complexity; suitable for scalability analysis
- Each model/sub-component used as a separate SUL



Scalability baseline analysis

Baseline

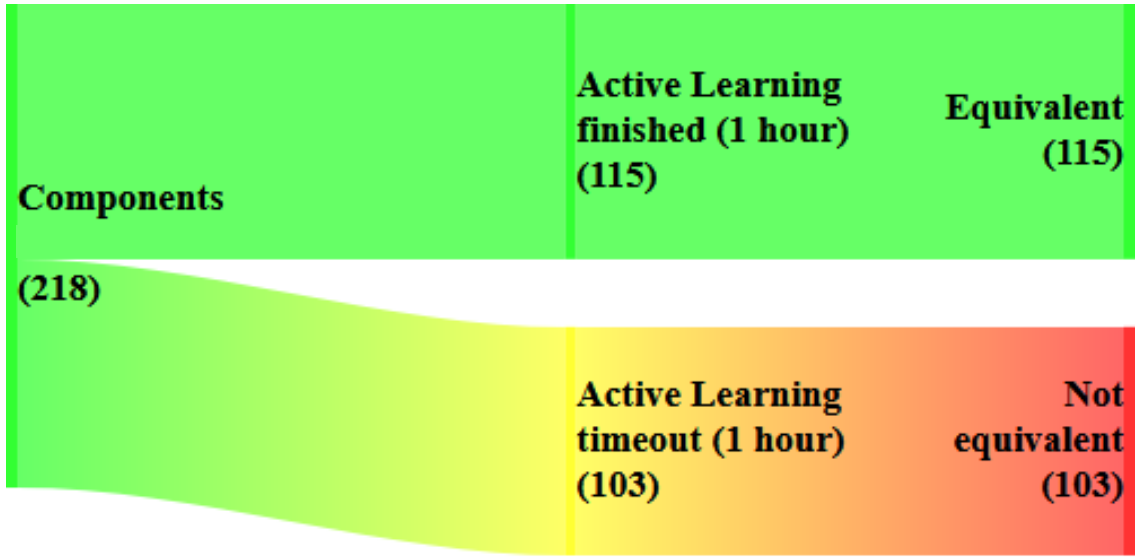
- TTT learning algorithm: state of the art algorithm, redundancy-free data structures
- Wp testing algorithm: guarantees completeness, requires size of the state space
- Active learning timeout: 60 minutes per component



Scalability baseline analysis

Baseline results

- 115/218 components learned (≈ 53%)
- Remaining 103/218 components timed out, and not completely learned



Scalability baseline analysis

Scalability insights

- Increasing timeout does not help much

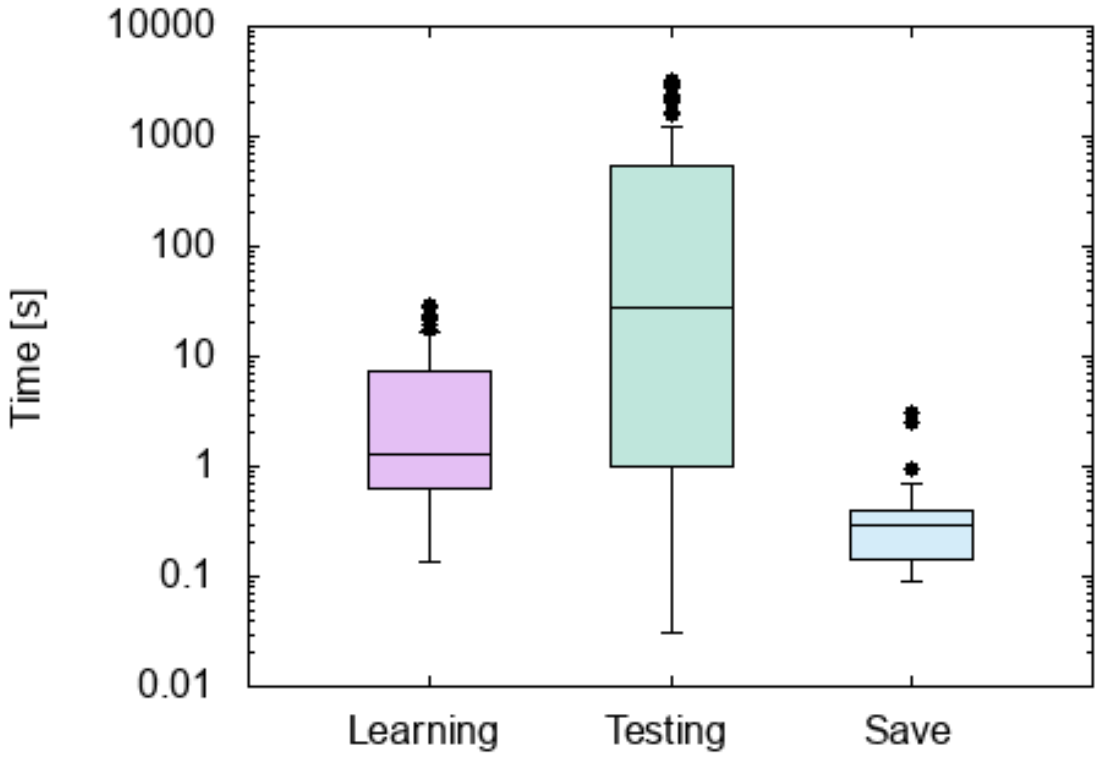
Timeout per component	Components learned
1 hour	115/218 (≈ 53%)
8 hours	121/218 (≈ 56%)
48 hours (2 days)	121/218 (≈ 56%)
192 hours (8 days)	121/218 (≈ 56%)

Scalability baseline analysis

Scalability insights

- Increasing timeout does not help much
- Most of the time spend on testing, not learning
  - As expected, due to theoretical worst-case bounds
  - Confirmed in practice, by experimental results

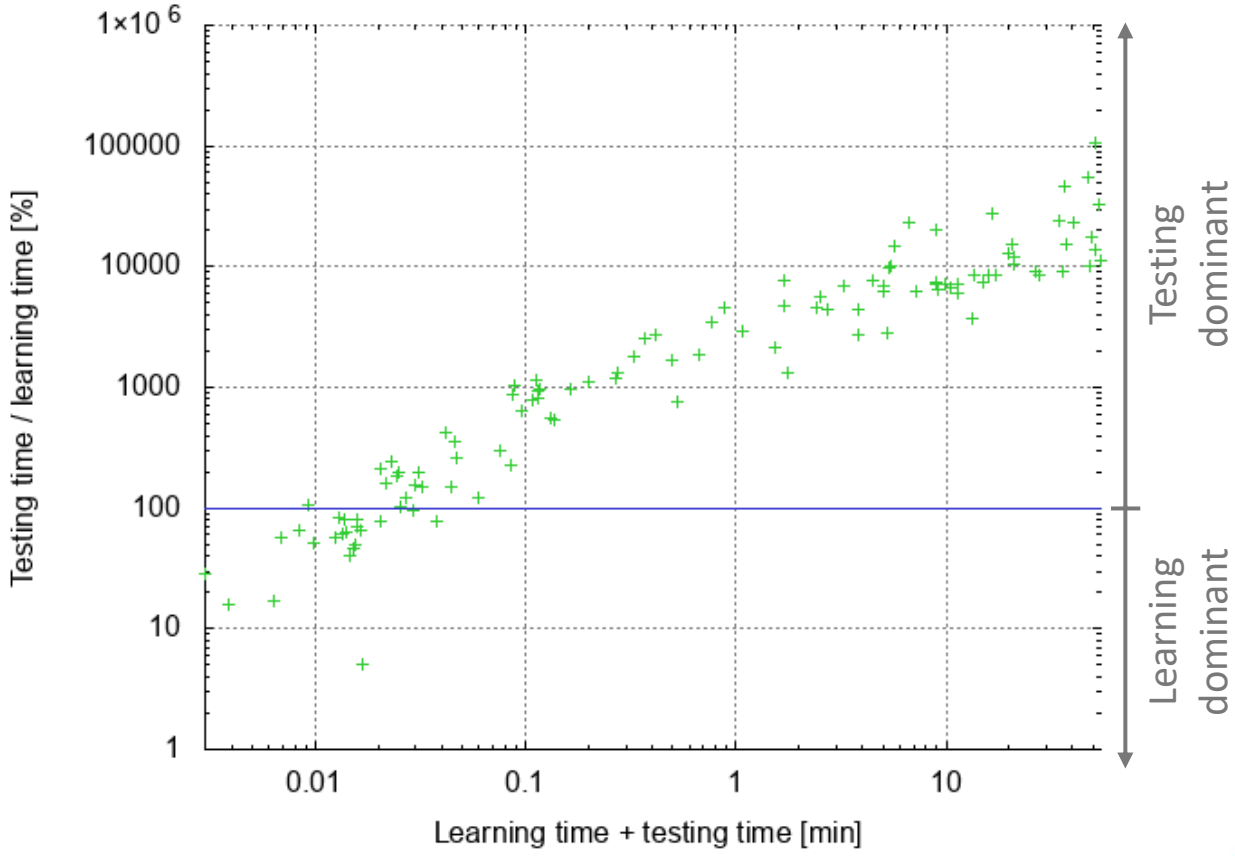
Phase	Theoretical worst-case bound	Duration
Learning	Polynomial	< 30 seconds
Testing	Exponential	< 54 minutes
Saving results	Linear	< 4 seconds



Scalability baseline analysis

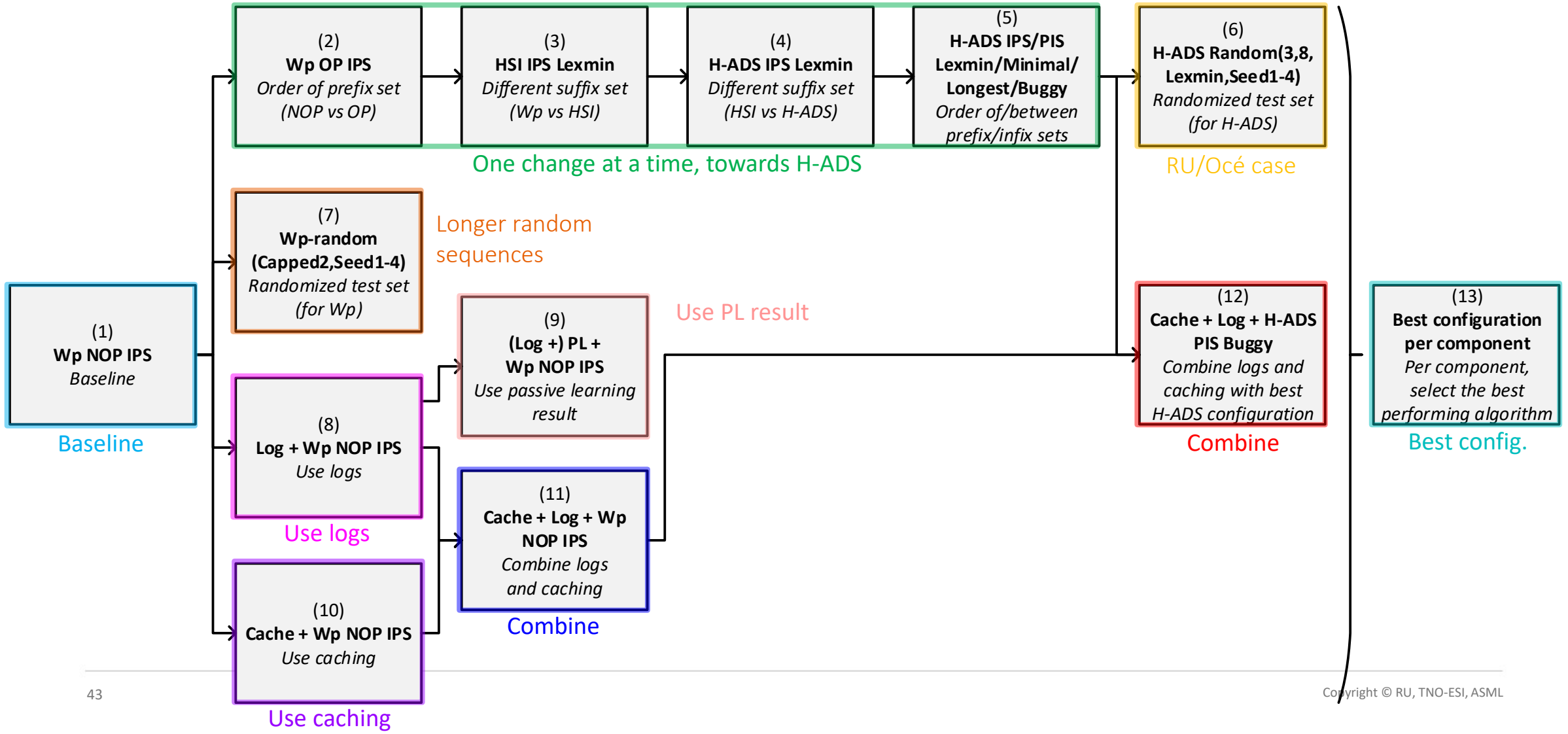
Scalability insights

- Increasing timeout does not help much
- Most of the time spend on testing, not learning
- Testing more dominant for complex components
  - As expected, due to theoretical worst-case bounds
  - Confirmed in practice, by experimental results



→ Look into alternative testing algorithms

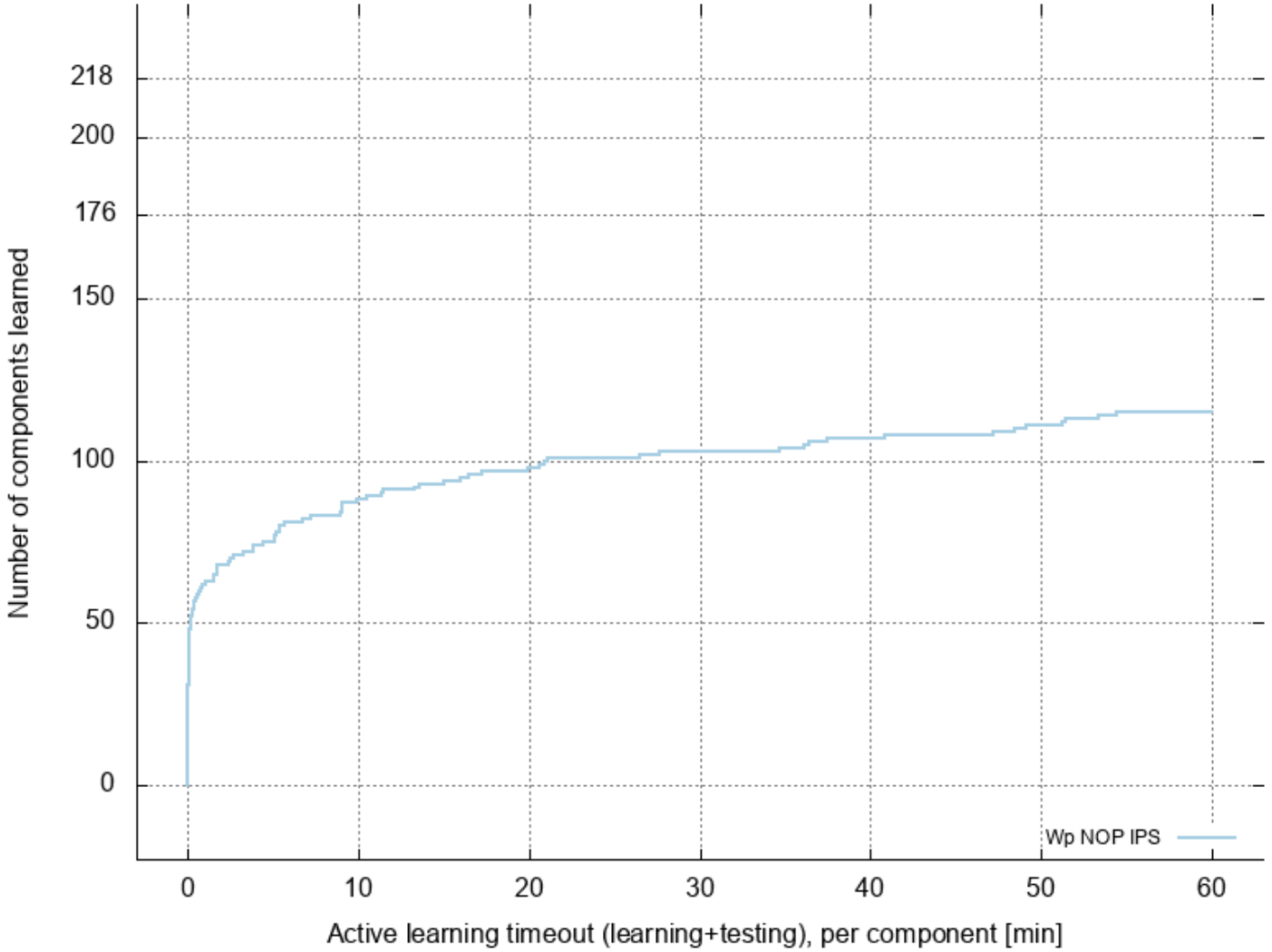
Test algorithms



Scalability improvements

1) Wp NOP IPS

- Baseline
- 115/218 components learned (≈ 53%)

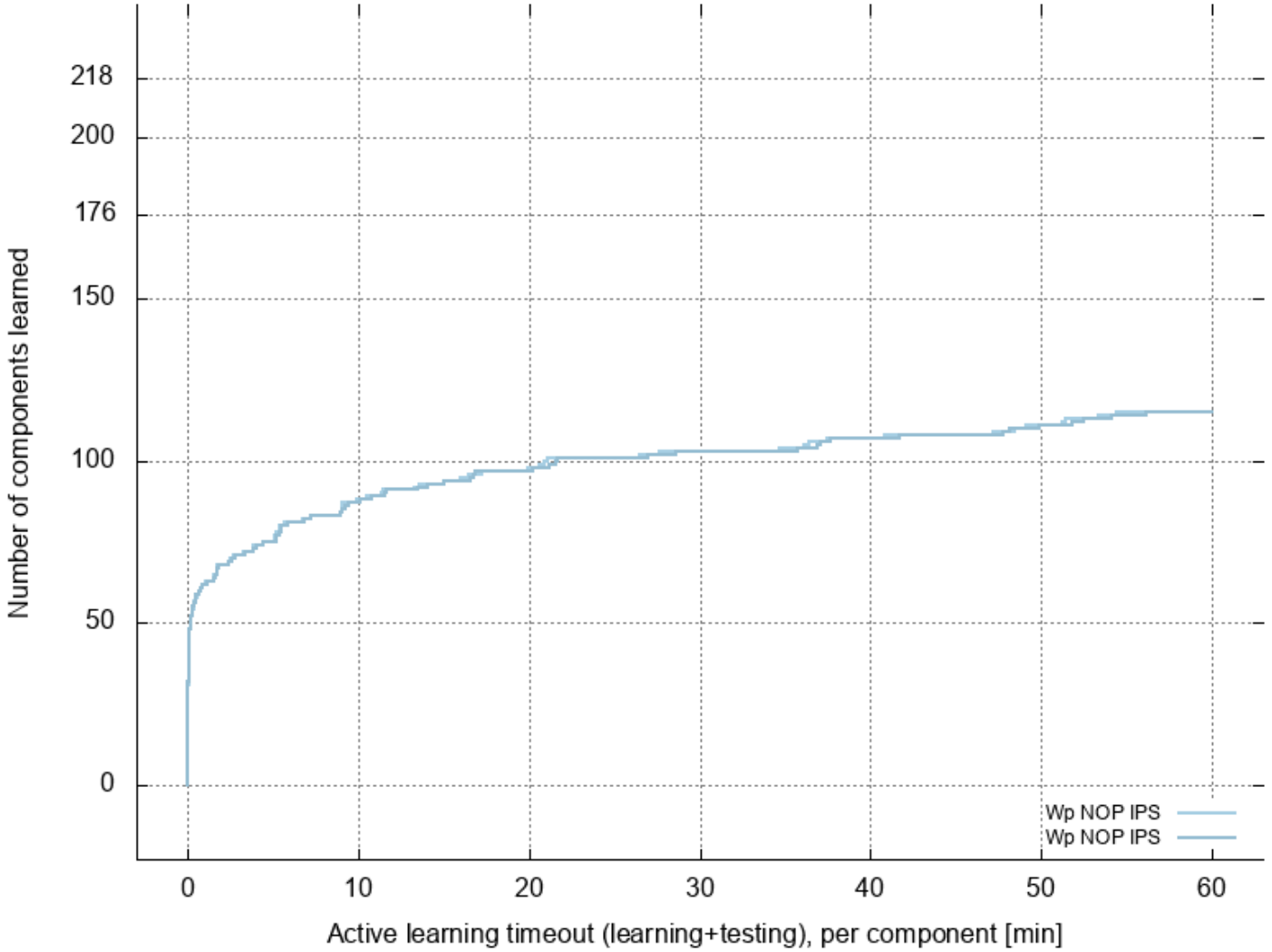


Scalability improvements

1) Wp NOP IPS

- Baseline
- 115/218 components learned (≈ 53%)

- Another baseline
- Same number of components learned
- Nearly indistinguishable in plot



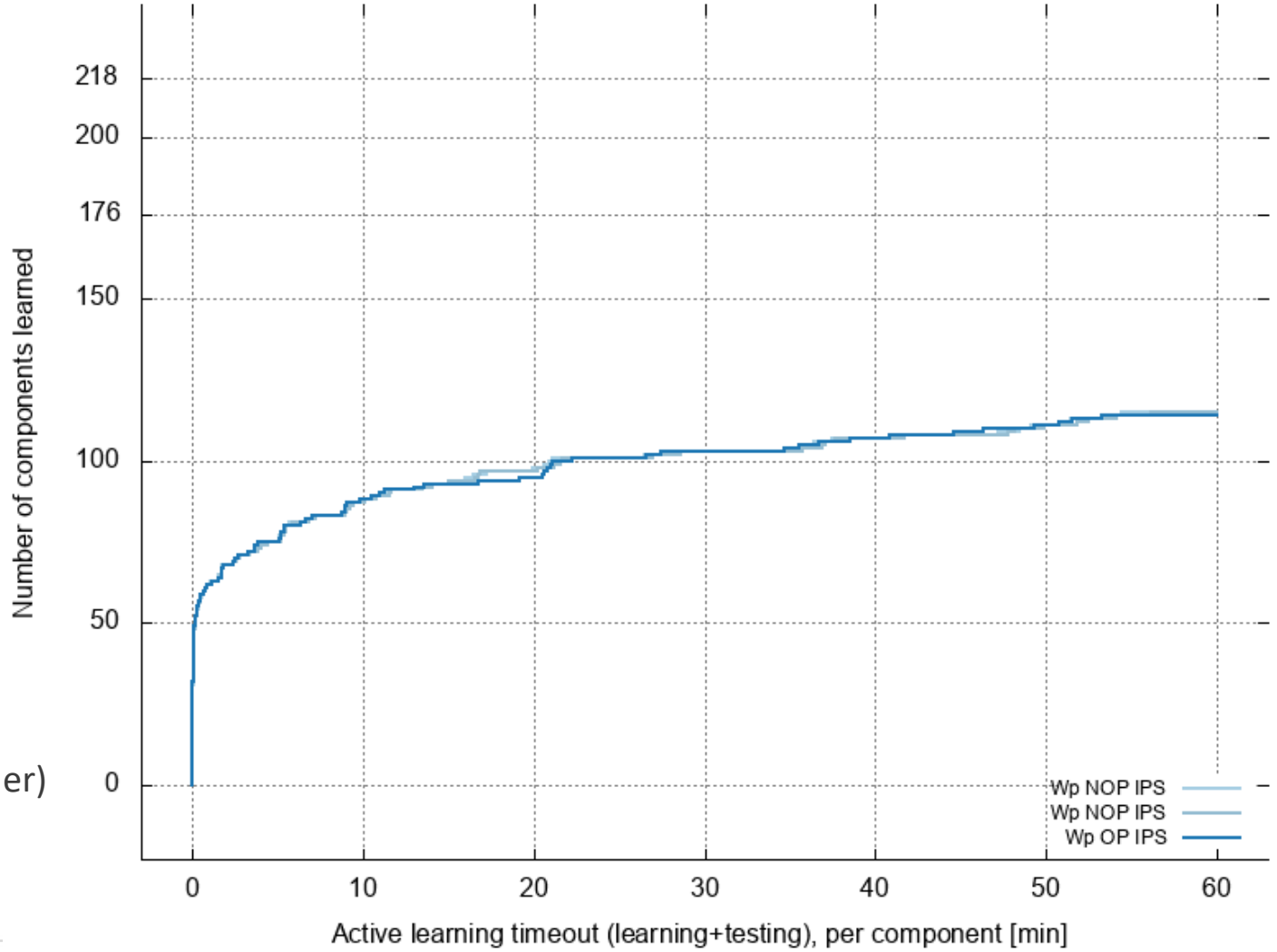
Scalability improvements

2) Wp OP IPS

- One less component learned than baseline
- Nearly indistinguishable in plot

Details:

- Wp NOP → Wp OP
- Only order of prefix set is changed
- Both BFS with given alphabet order
- OP = Order Preserving (insertion order)
- NOP = Non Order Preserving (undefined order)
- 114/218 components learned (≈ 52%)



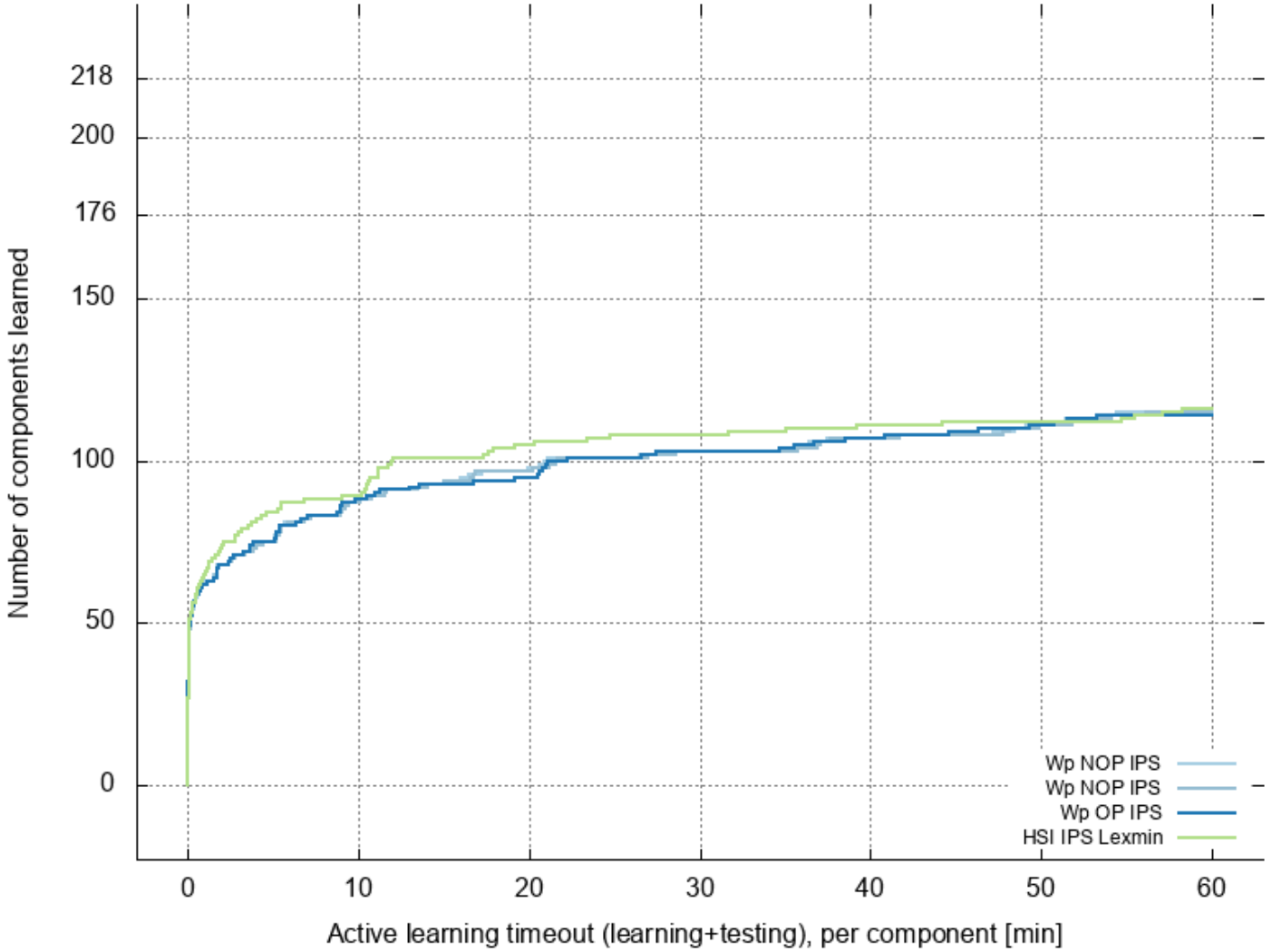
Scalability improvements

3) HSI IPS Lexmin

- Slightly faster than Wp for simpler components
- One more component learned than with Wp

Details:

- Wp → HSI
- Only suffix set is changed
- 116/218 components learned (≈ 53%)



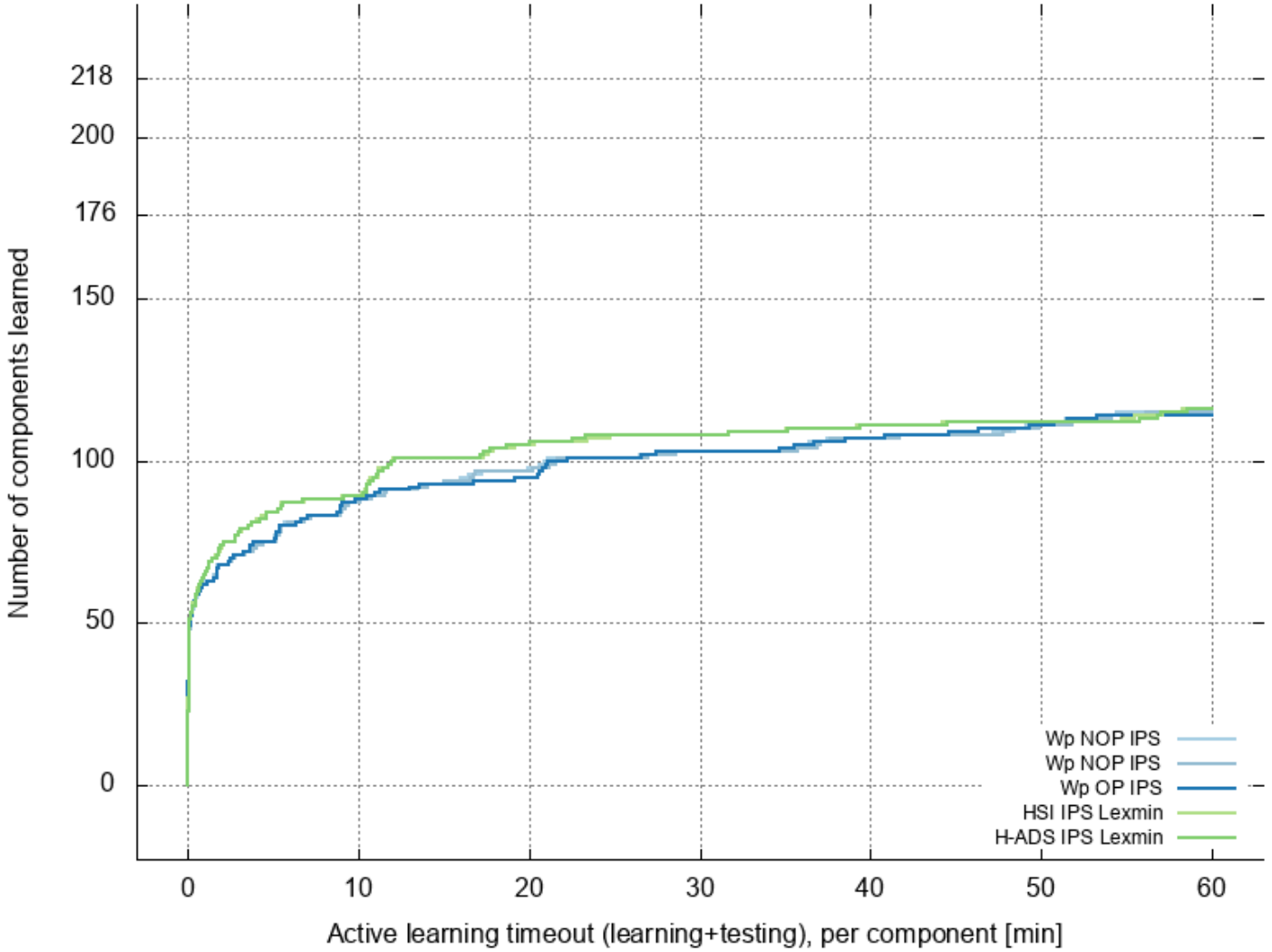
Scalability improvements

4) H-ADS IPS Lexmin

- Same number of components learned as with HSI
- Nearly indistinguishable in plot

Details:

- HSI → H-ADS
- Only suffix set is changed
- Uses an ADS if it exists, otherwise fallback to HSI
- 116/218 components learned (≈ 53%)



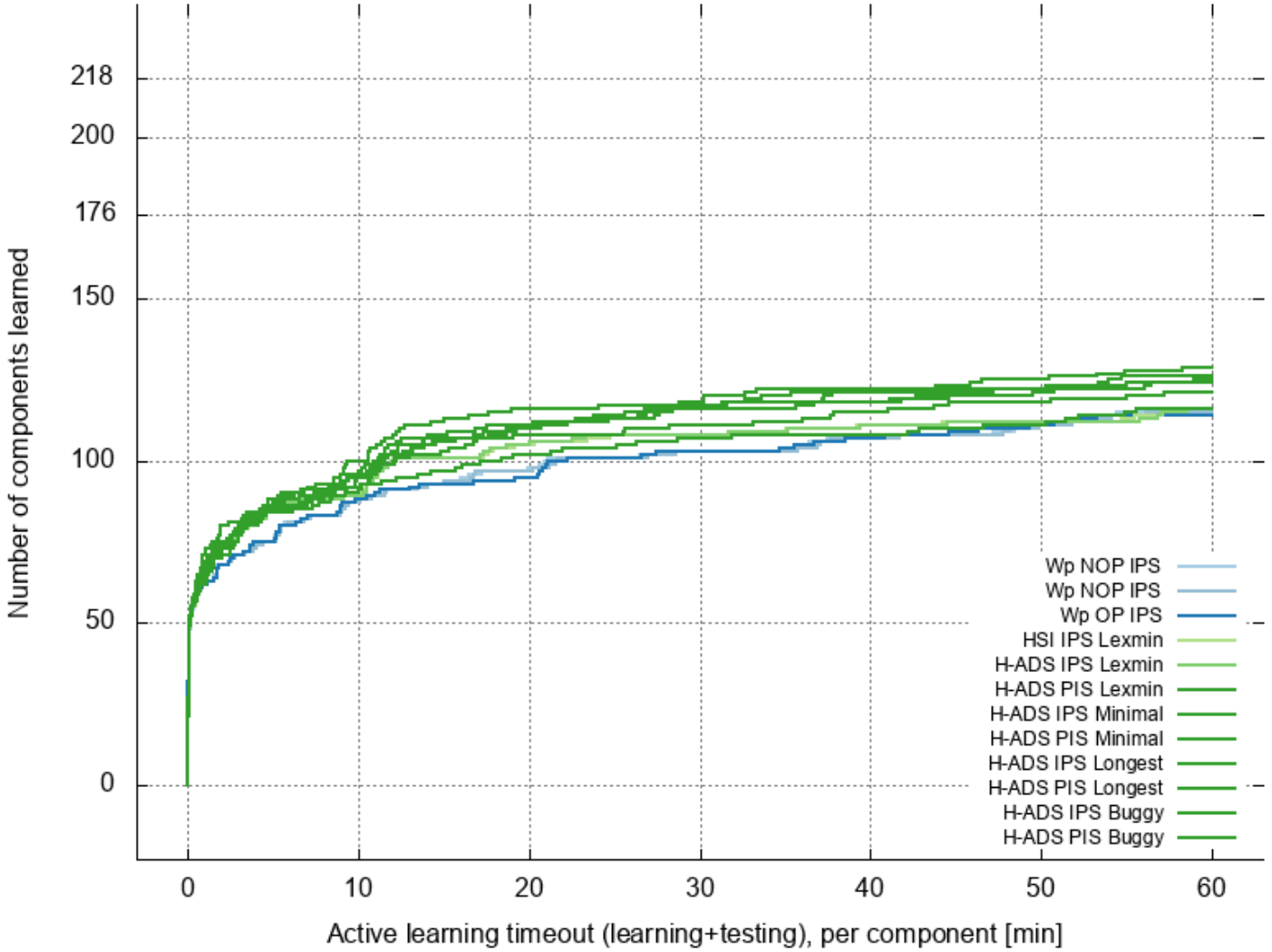
Scalability improvements

5) H-ADS IPS/PIS Lexmin/Minimal/Longest/Buggy

- Varying ordering can help to learn more components

Details:

- 4 prefix orders / both IPS and PIS
- 8 test algorithm configurations in total
- Between 116 and 129 components learned
- Generally PIS performs better than IPS
- H-ADS PIS Buggy performs best, with 129/218 components learned (≈ 59%)



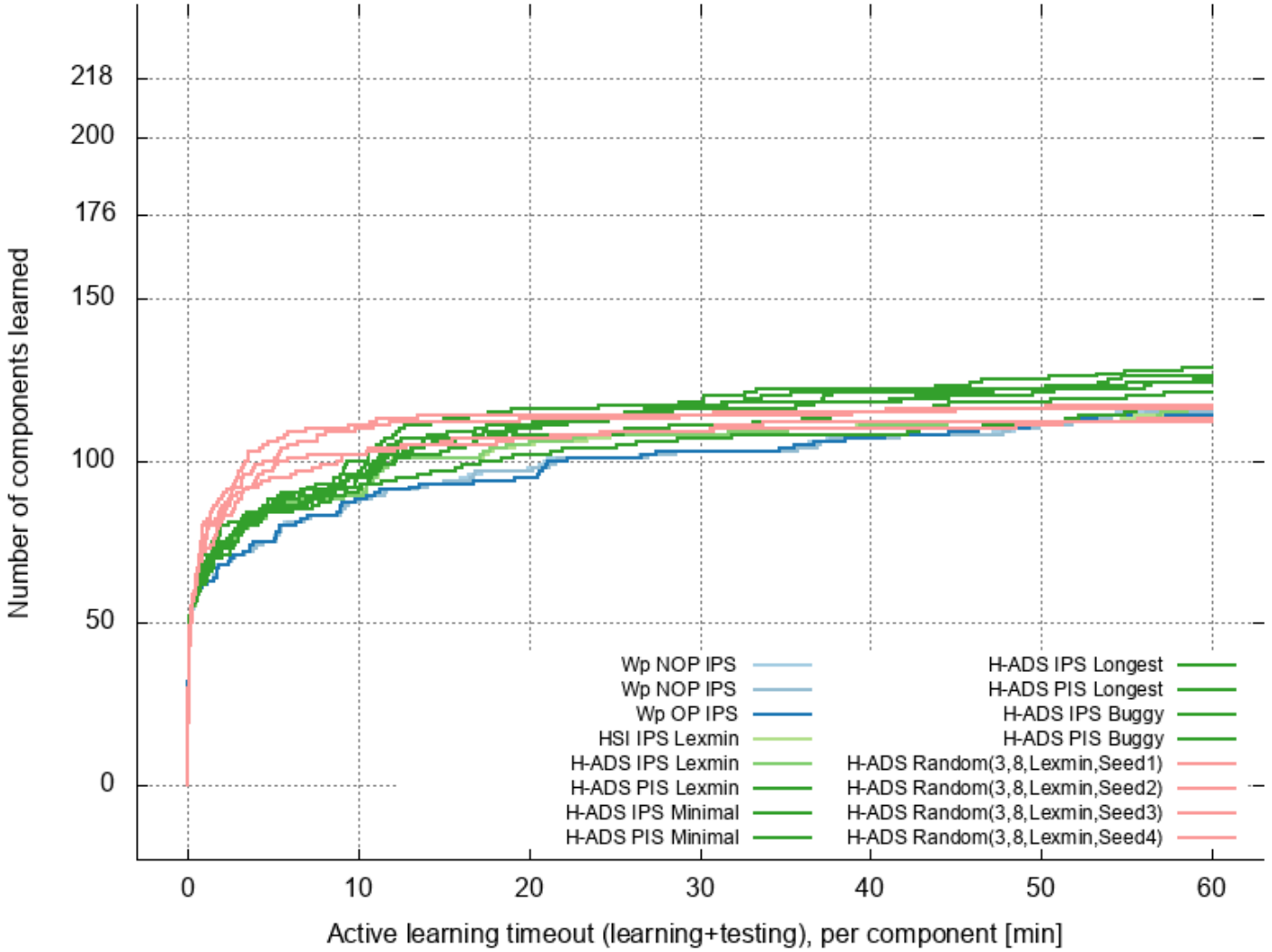
Scalability improvements

6) H-ADS Random(3,8) LastRound(Lexmin) Seed(1-4)

- Faster for simpler components
- Learns less complex components (conjecture: due to random length limited to 8)

Details:

- Randomized test set order
- Infix fixed length 2, random length 8; same as for RU/Océ case
- Last round Lexmin to ensure termination
- Executed using 4 different seeds
- 112-117/218 comp. learned (≈ 51-54%)



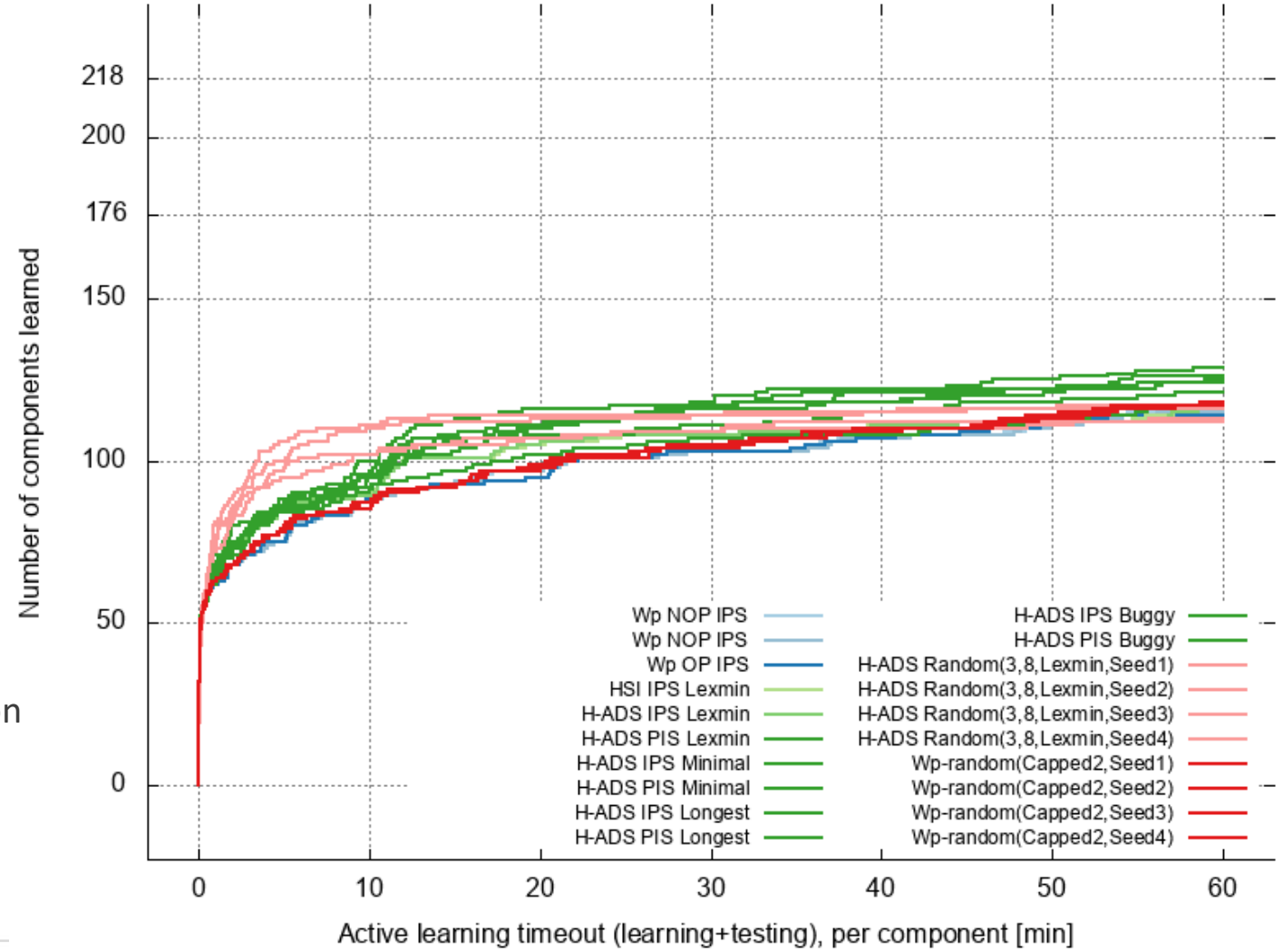
Scalability improvements

7) Wp-random NOP IPS Capped(2) Seed(1-4)

- Slower than H-ADS random for simpler components
- One more component learned than with H-ADS random

Details:

- Randomized test set order
- Infix non-random capped at length 2, remainder randomized and not capped
- Last round normal Wp to ensure termination
- 118/218 components learned (≈ 54%)
- Very little variation between the 4 runs



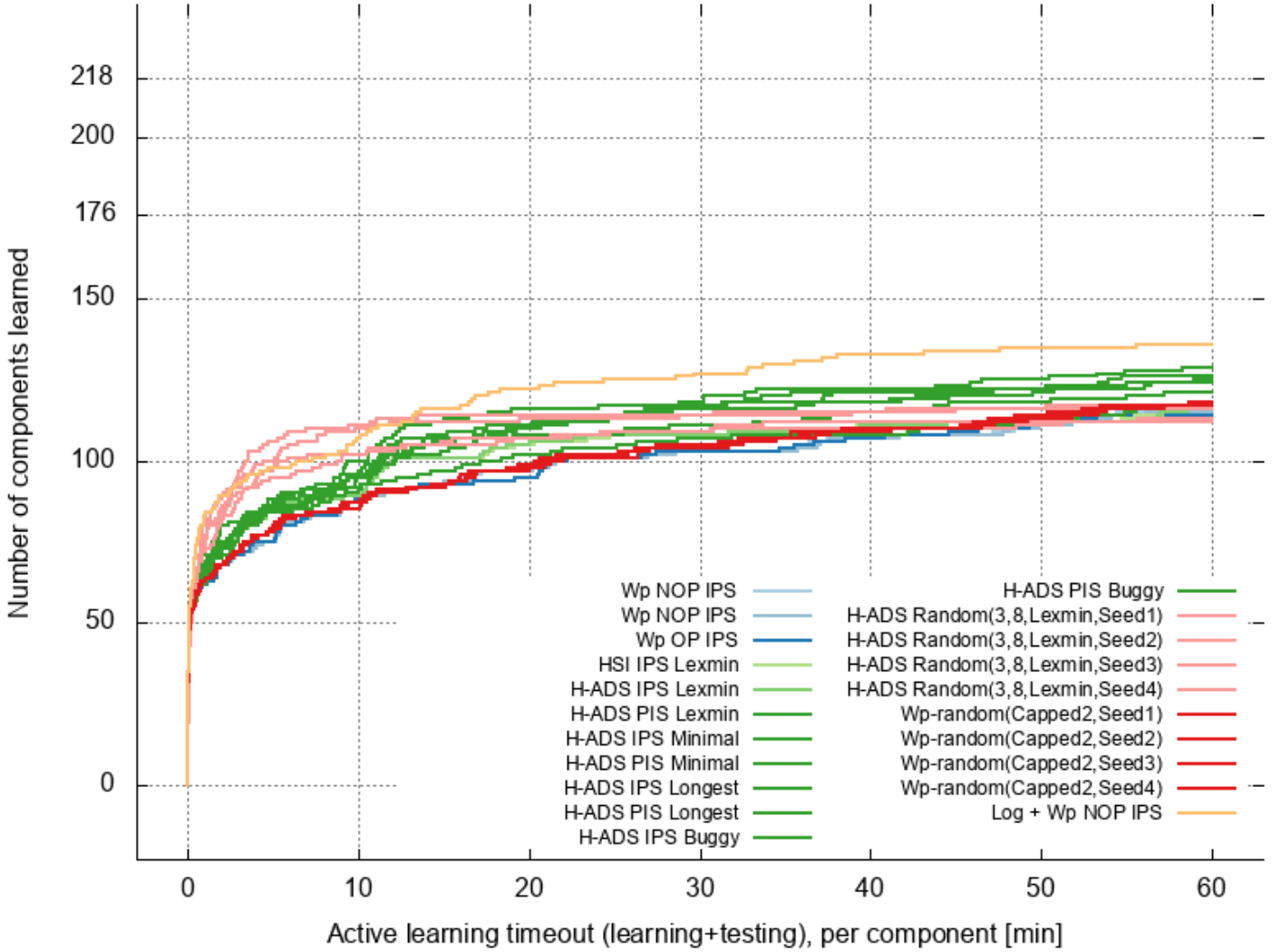
Scalability improvements

8) Log + Wp NOP IPS

- 21 more components learned than baseline
- Cheap to use, great benefit

Details:

- Execution logs contain behavior that the component has exhibited
- Any behavior in the logs that is not in the hypothesis is a counter example
- Easily find counter examples without consulting the SUL
- Benefit depends greatly on quality of logs
- 136/218 components learned (≈ 62%)



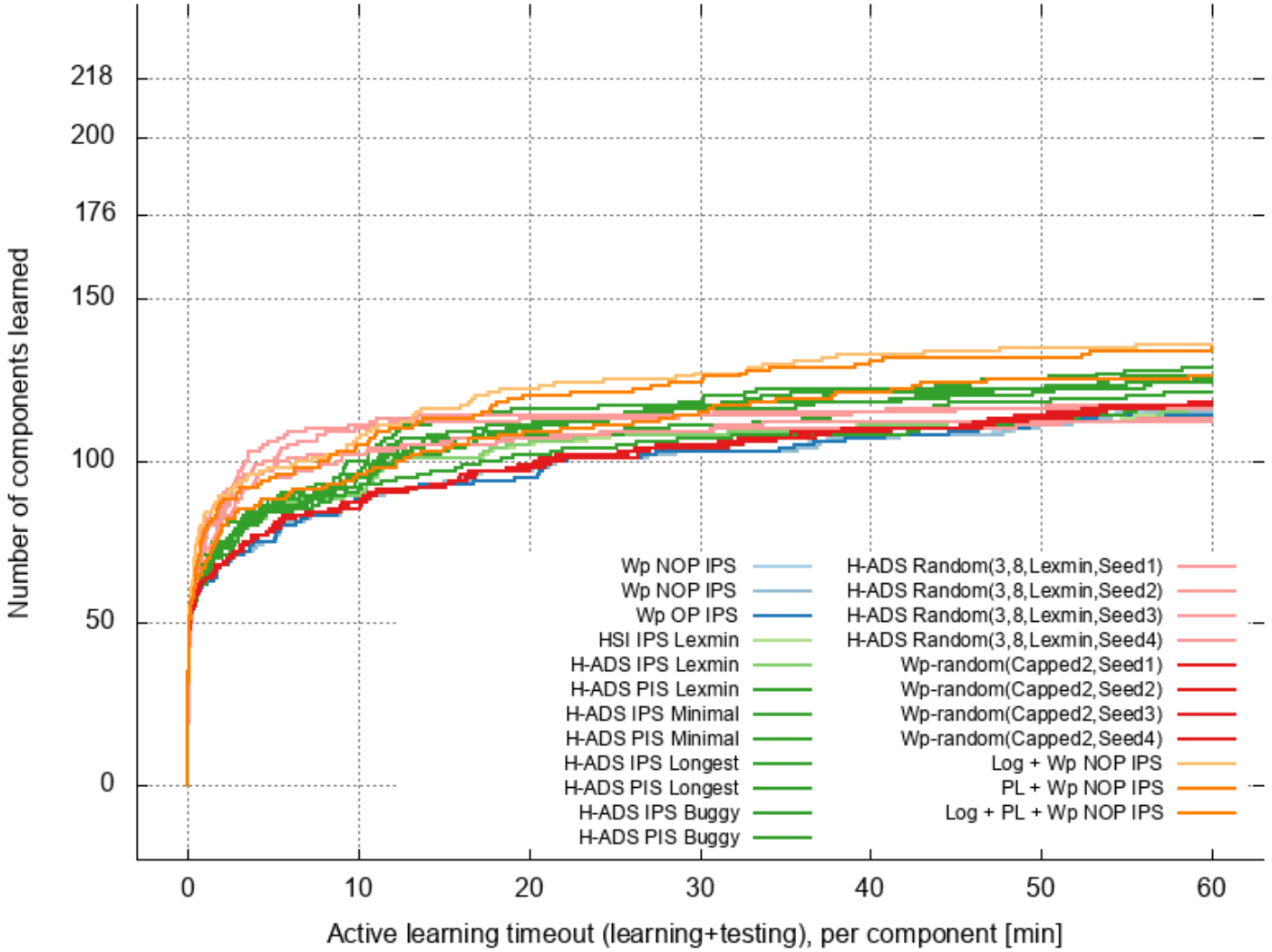
Scalability improvements

9) (Log +) PL + Wp NOP IPS

- Passive learning greatly underperforms wrt Log
- Log+PL slightly underperforms wrt Log
- Passive learning does not seem beneficial

Details:

- PL result is compared to hypothesis; diffs checked against SUL as potential counter examples
- Hope to benefit from generalization of PL algorithms; reject invalid generalization
- More expensive to compute than log-based oracle; PL-oracle needs to query SUL
- AL and PL employ similar state merging
- 126/218 (≈ 58%) and 135/218 (≈ 62%) components learned



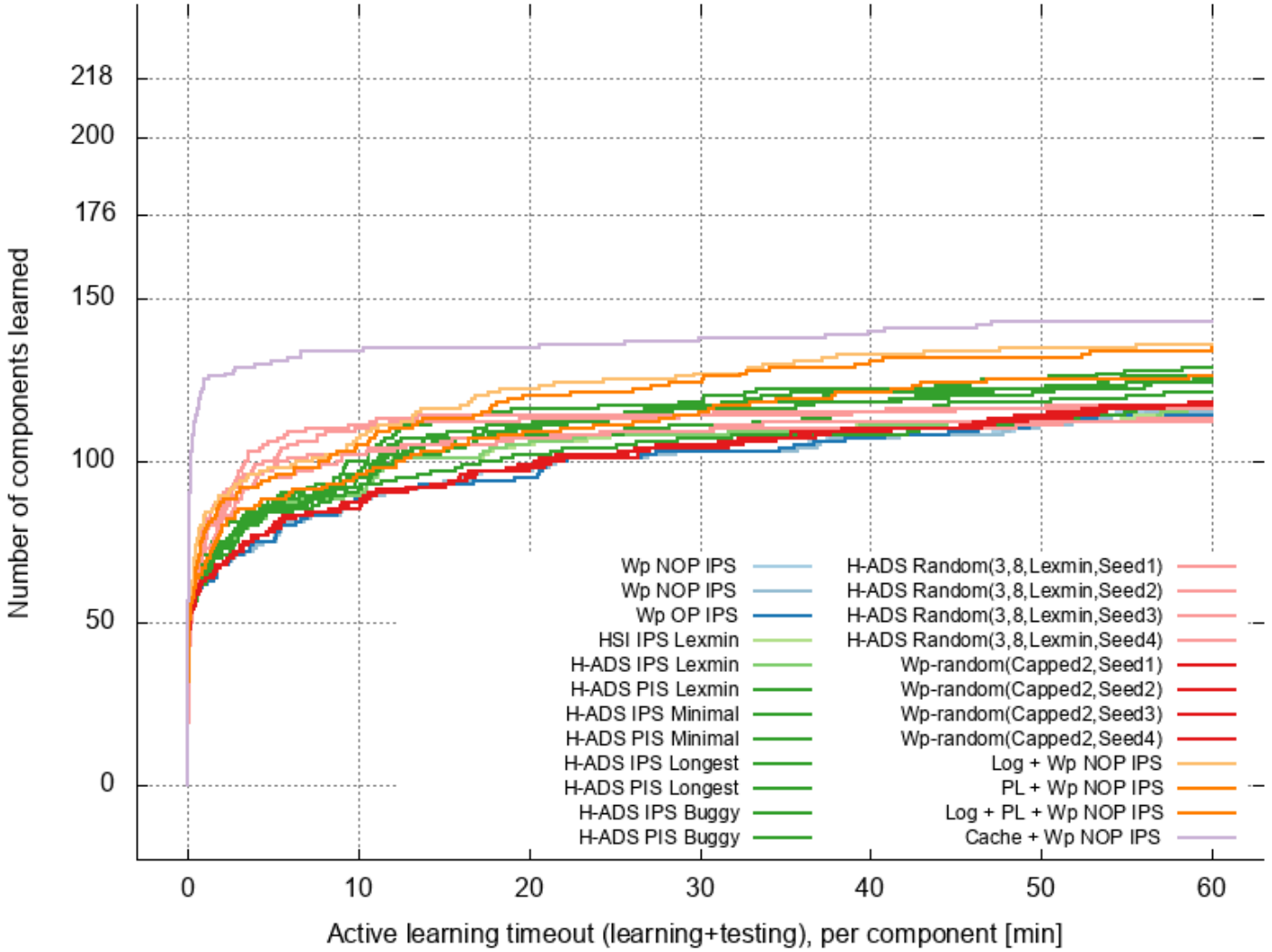
Scalability improvements

10) Cache + Wp NOP IPS

- Caching + sink state optimization; affects both learning and testing
- 28 more components learned than baseline
- Great benefit, enable it always

Details:

- Use caching for SUL responses
- Also includes sink state optimization; once in a sink state, the output is known
- Affects both membership queries (learning) and test queries (testing)
- 143/218 components learned (≈ 66%)



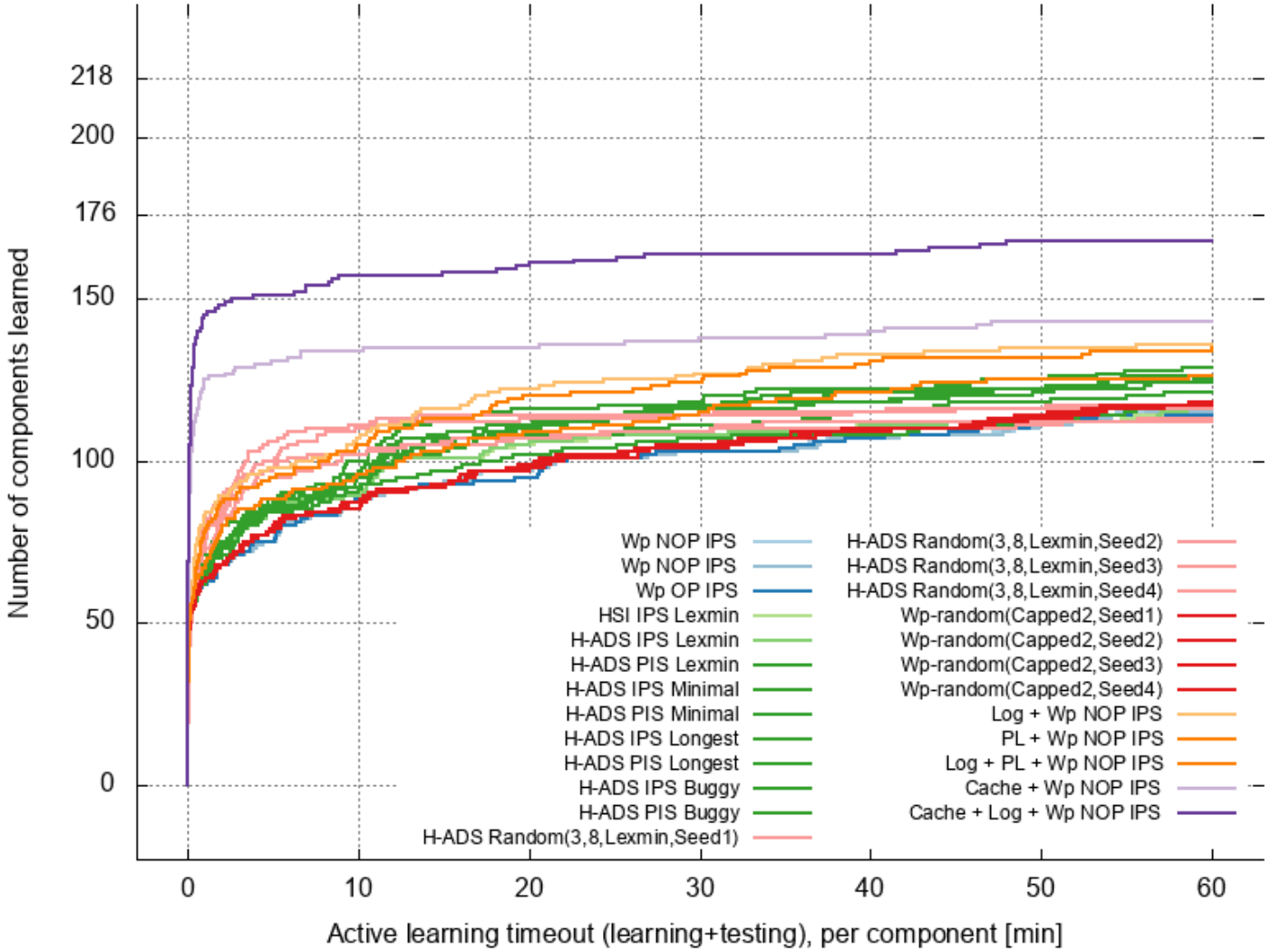
Scalability improvements

11) Cache + Log + Wp NOP IPS

- 53 more components learned than baseline
- Caching and logs seem to strengthen each other

Details:

- Combine using logs and caching
- 168/218 components learned (≈ 77%)



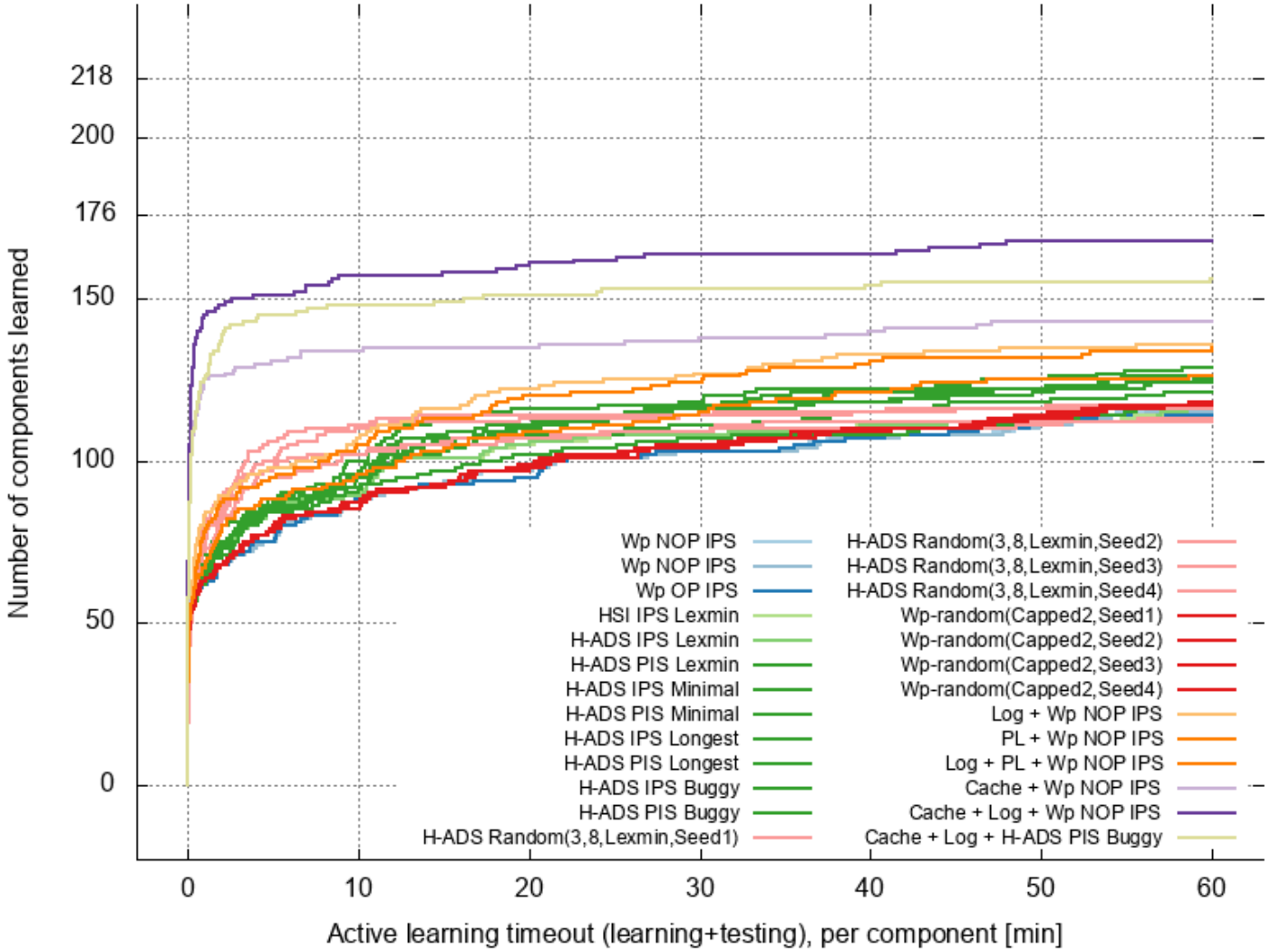
Scalability improvements

12) Cache + Log + H-ADS PIS Buggy

- Caching and logs have less impact on best H-ADS configuration
- 12 less components learned than with Wp

Details:

- Combine logs and caching with best H-ADS configuration
- 156/218 components learned (≈ 72%)



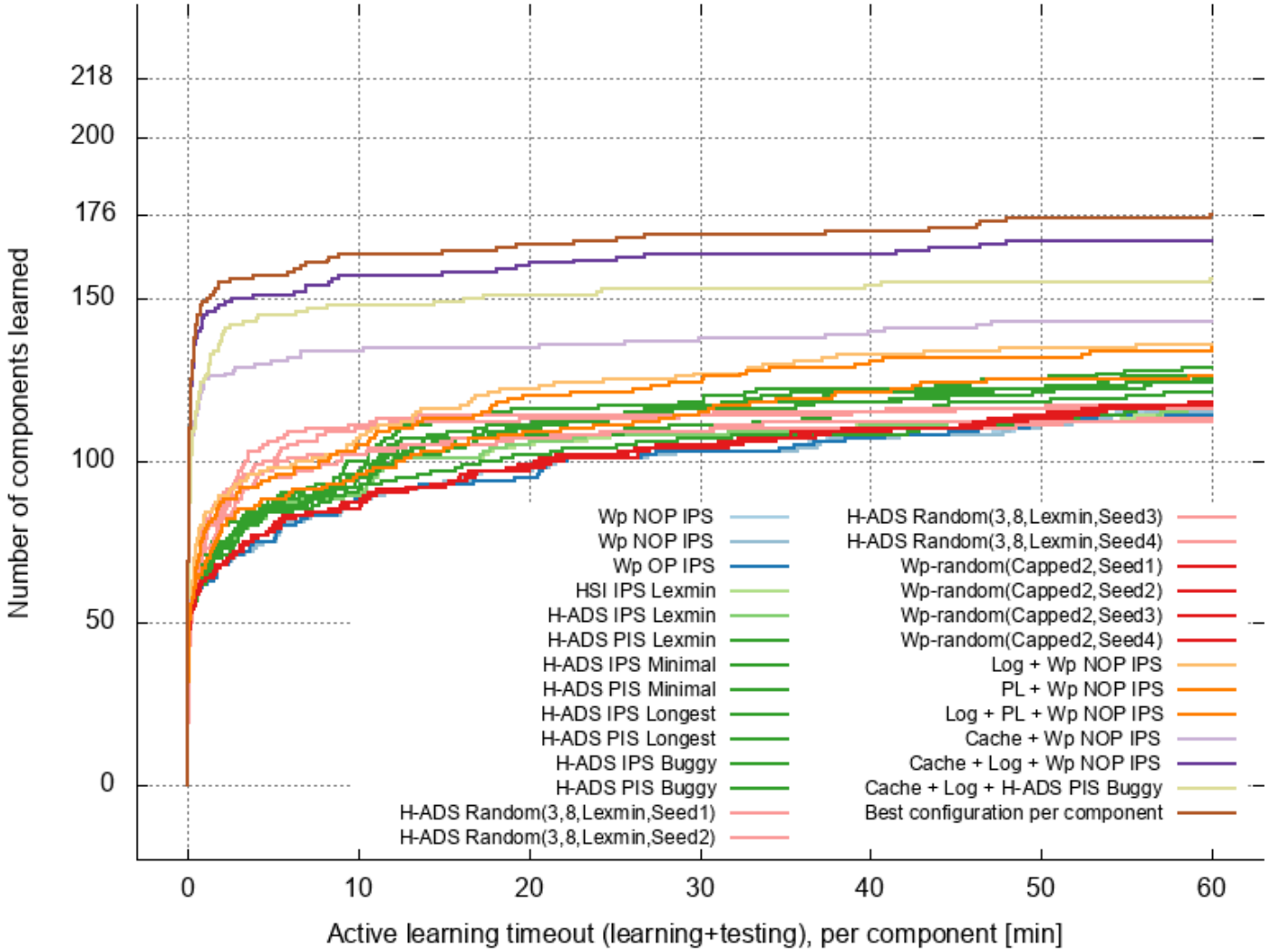
Scalability improvements

13) Best configuration per component

- 61 more components learned than baseline
- Best blackbox results so far; almost 81% learned

Details:

- Per component, select the test algorithm and configuration that works best
- Results in different configurations for different components
- 176/218 components learned (≈ 81%)



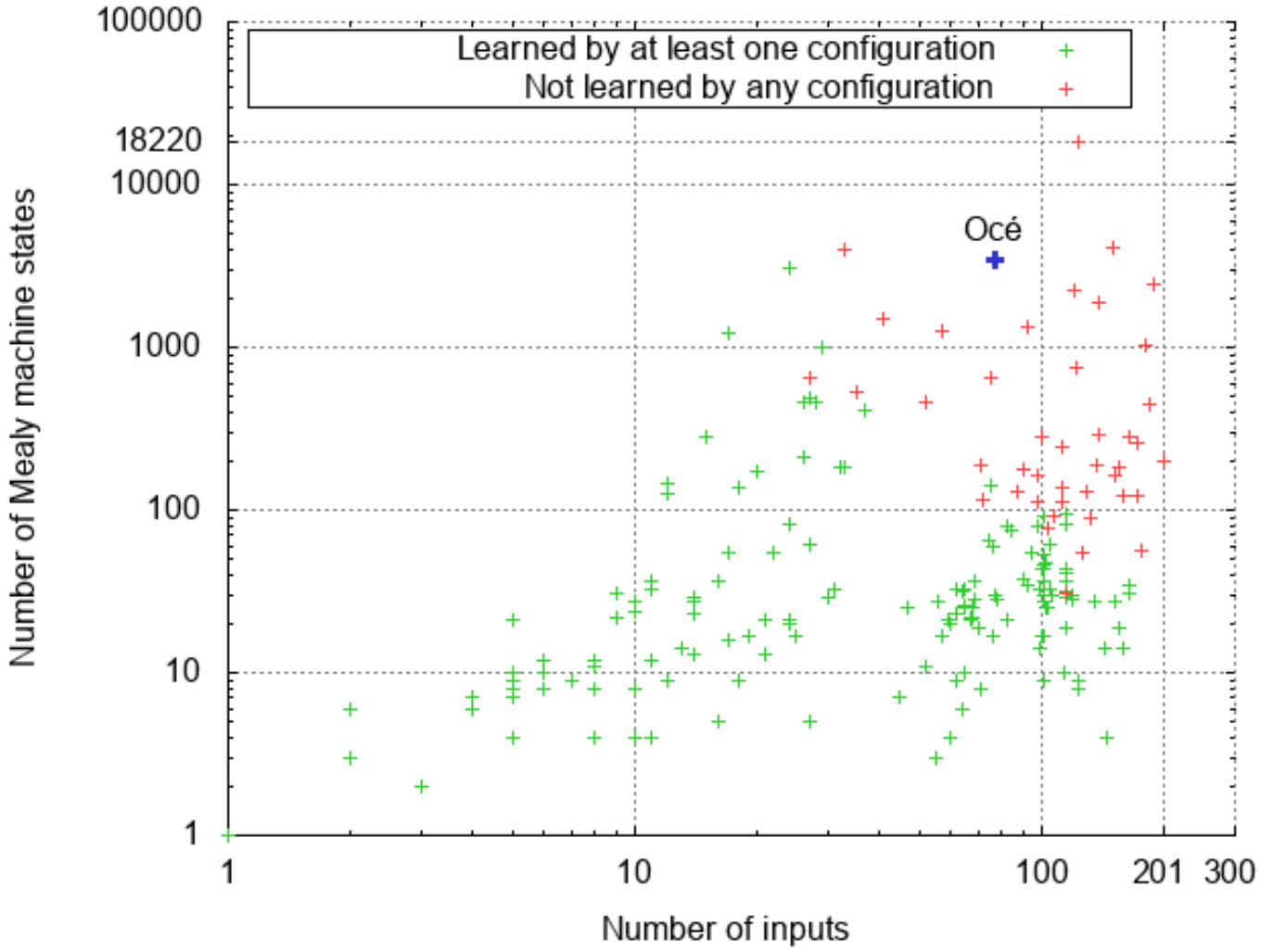
Scalability results

Overview

- Best blackbox results so far
- 176/218 components learned (≈ 81%)

RU/Océ case*

- In 8 hours, 26 minutes
- Different hardware/setup
- Also applied component-specific abstractions, etc

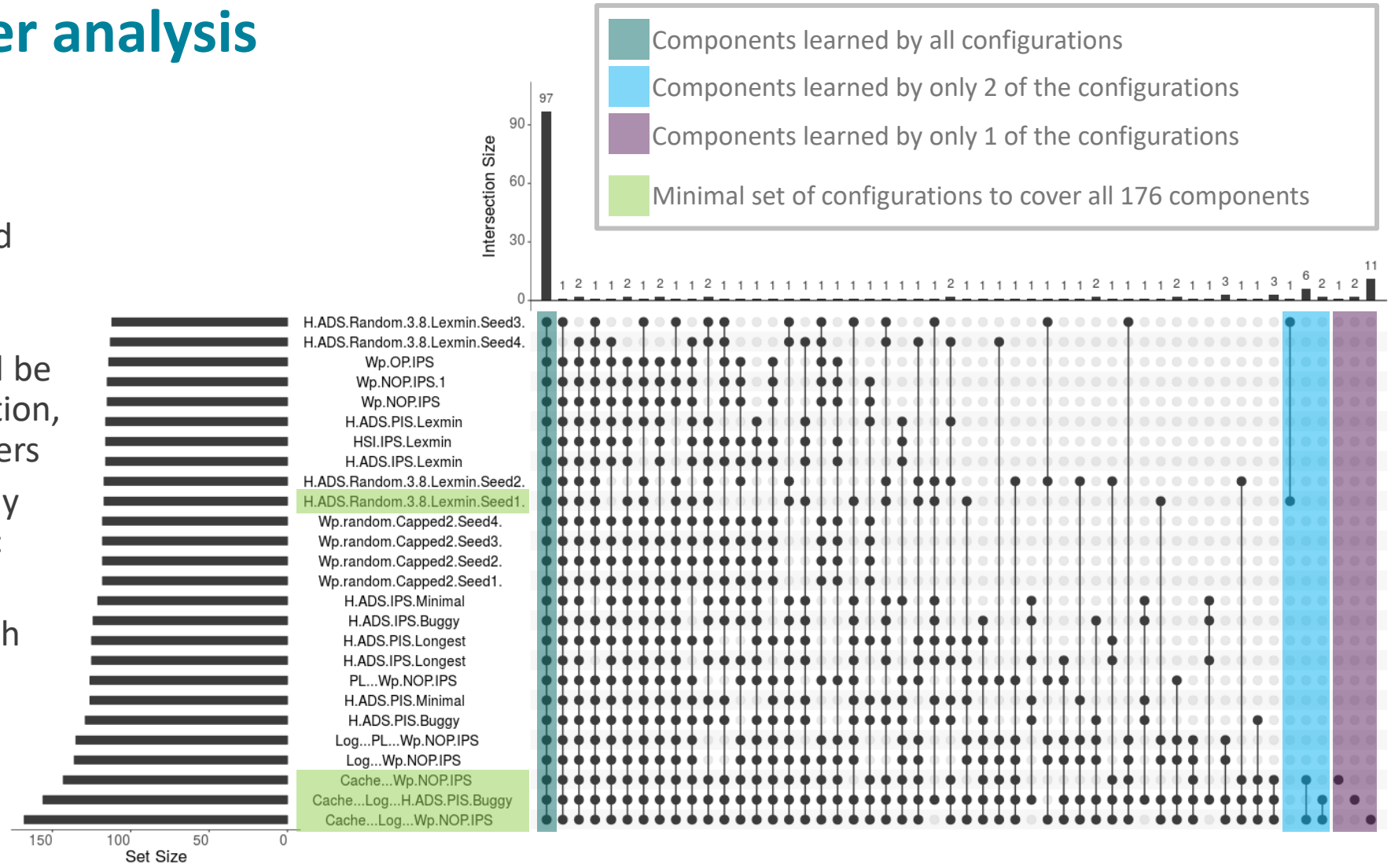


*) RU/Océ case: W. Smeenk, J. Moerman, F. Vaandrager, and D. N. Jansen, "Applying Automata Learning to Embedded Control Software," in International Conference on Formal Engineering Methods 2015.

Scalability further analysis

Unique components

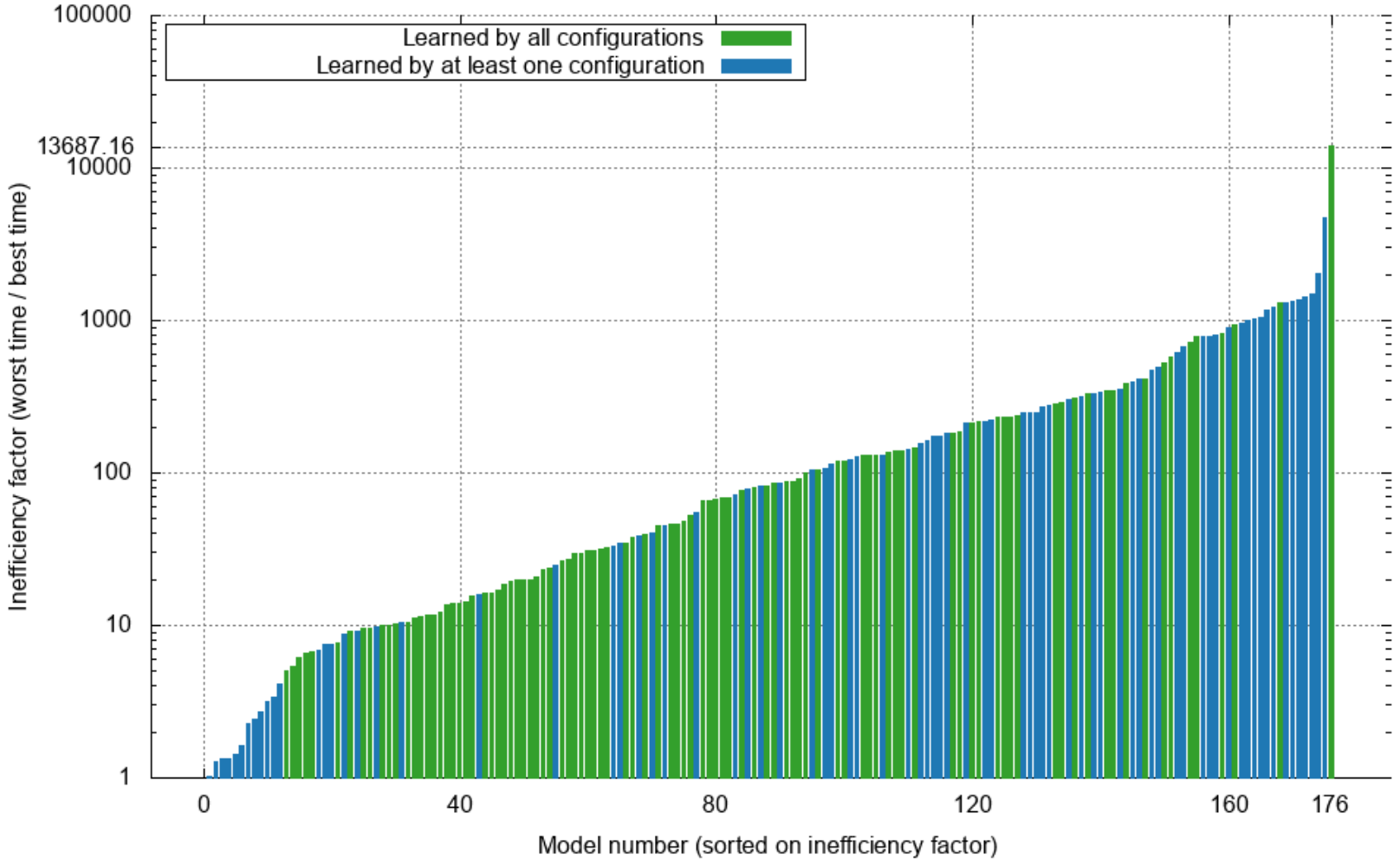
- 97 components learned by all configurations
- For remaining 79 components that could be learned by a configuration, the configuration matters
- 14 components are only learned by one specific configuration
- 4 configurations enough to cover the 176 components



Scalability further analysis

Best vs worst configuration:

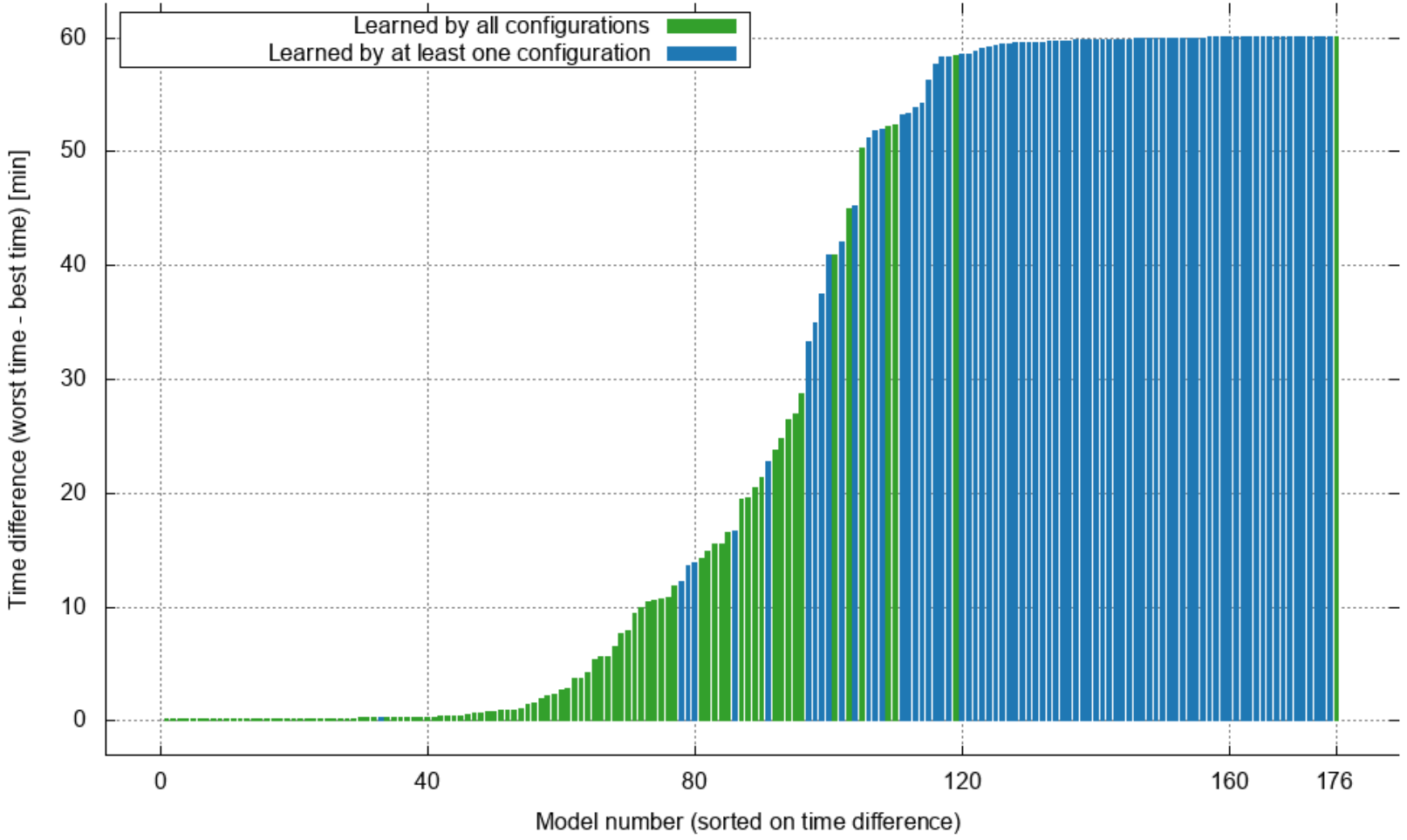
- One component takes over 13,000 times as long to learn using worst configuration than best configuration



Scalability further analysis

Best vs worst configuration:

- One component takes over 13,000 times as long to learn using worst configuration than best configuration
- One component can be learned in ≈ 3 seconds using best configuration and in ≈ 58 minutes without timeout using worst configuration
- For individual components, the choice of configuration can matter a lot!



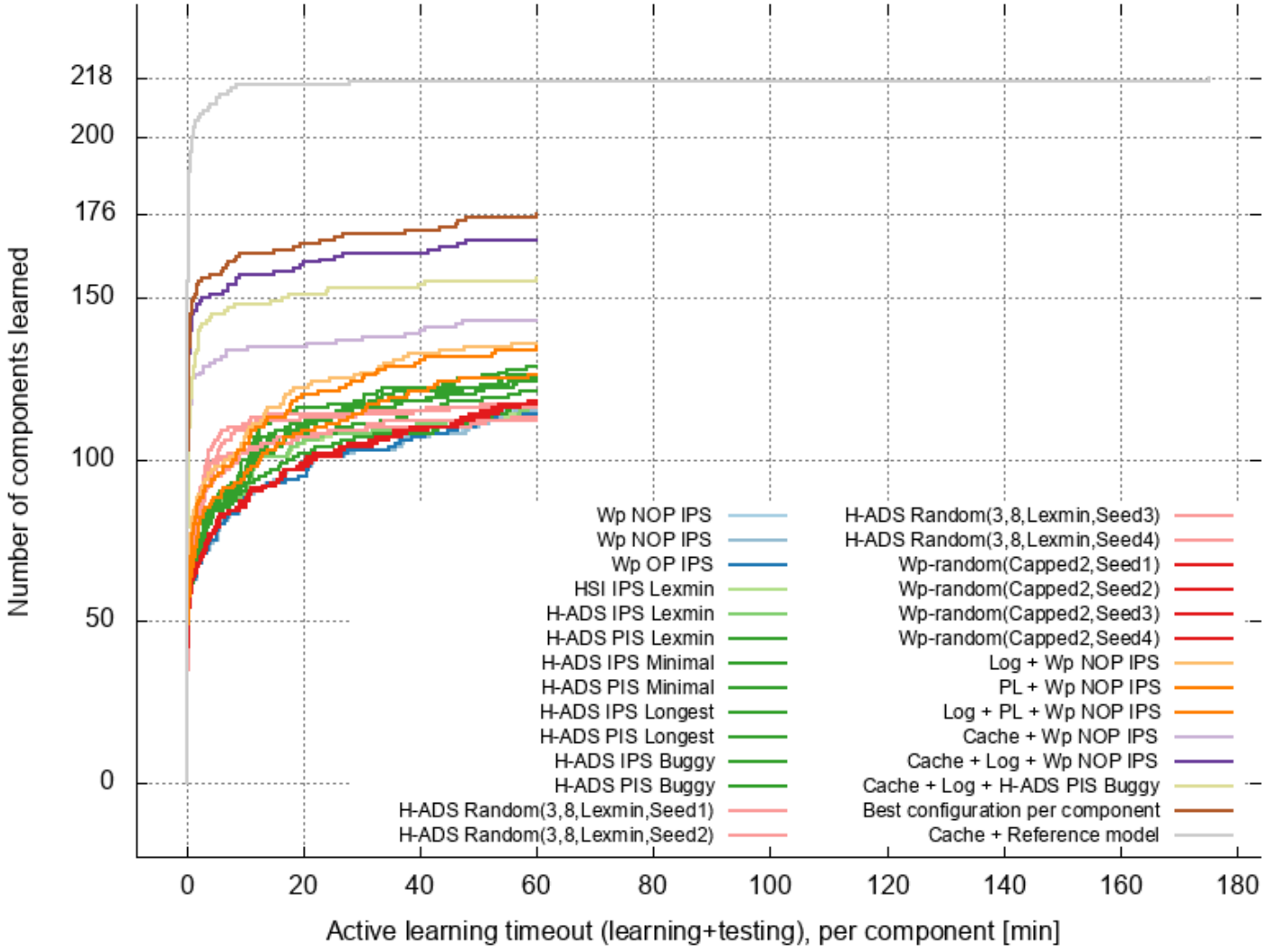
Scalability further analysis

The limit: how far can we get?

- With perfect testing, can learn all components; learning fast enough
- 216/218 learned in under 9 mins; 217/218 learned in under 28 mins; 218/218 learned in under 3 hours

Details:

- Test oracle: Cache + Reference model
- ‘Perfect testing’, by computing difference between hypothesis and reference state space; any difference is a counter example; no test queries to SUL
- Only testing is changed; learning is performed normally, including membership queries to the SUL
- Can’t be faster than this for testing

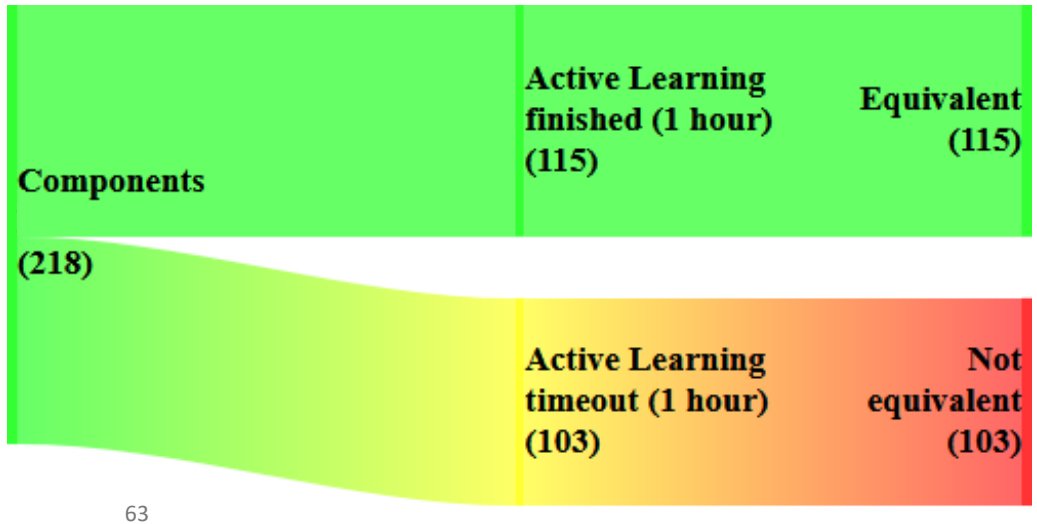


Scalability conclusions

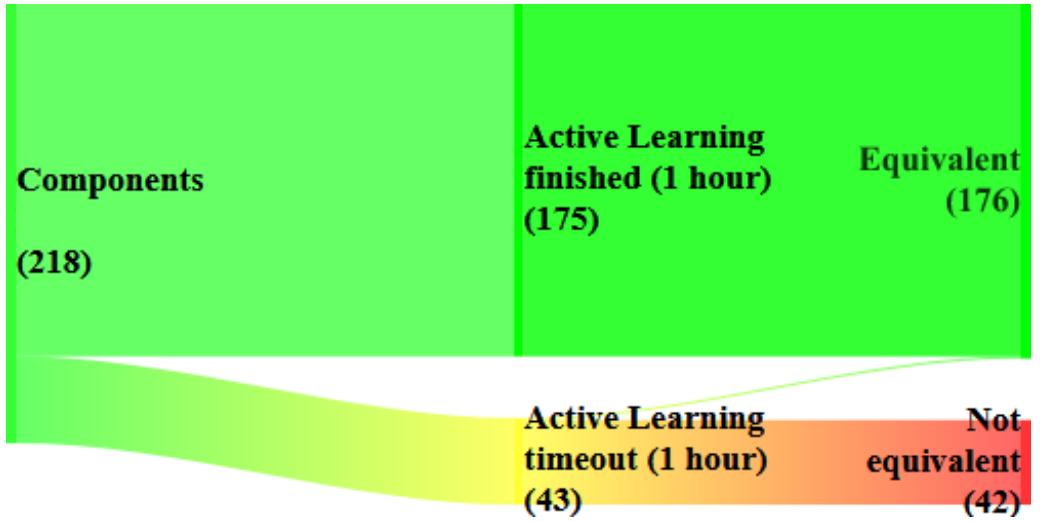
Conclusions

- Testing is bottleneck for AL
- Test algorithm choice has great impact on AL performance
- No single algorithm/configuration is best; greatly depends on component
- With best configuration per component: from ≈ 53% to ≈ 81% of components learned

Baseline:



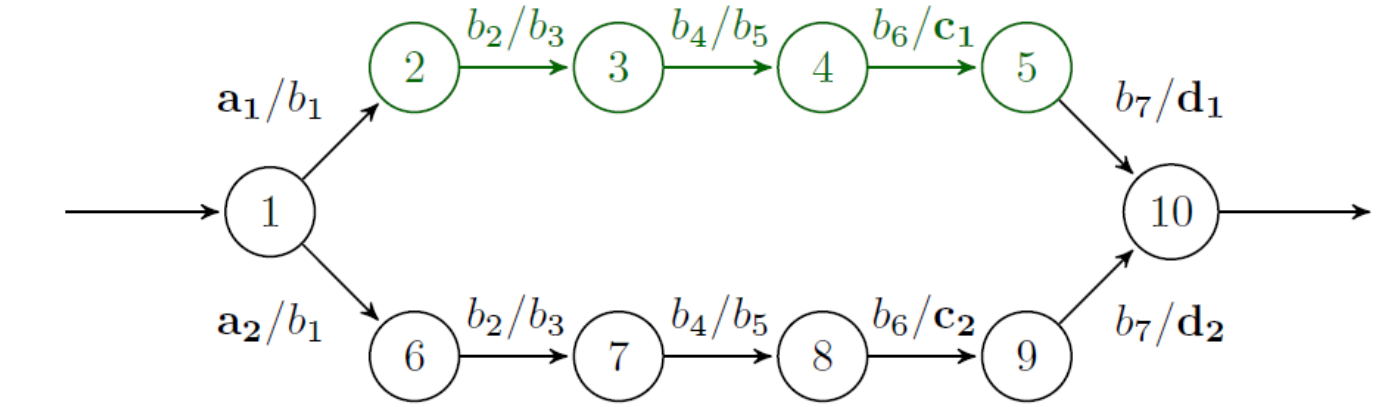
Best configuration per component:



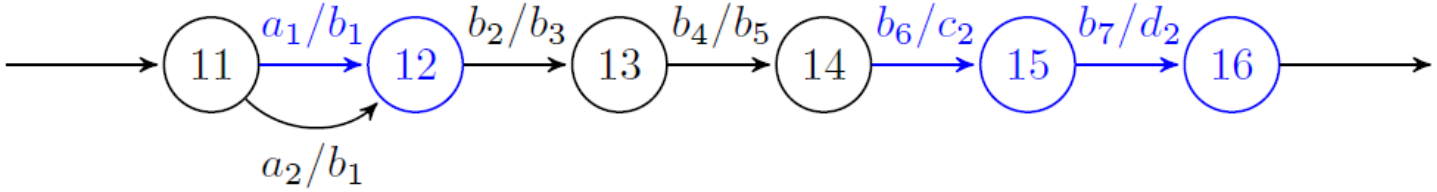
Black-box approaches struggle to find counter-examples

Far output distinction behavior

- Hypothesis has only bottom path
  - a1/b1, …, b6/**c2**, b7/**d2** is possible
  - b6/**c1** and b7/**d1** never possible
- One specific sequence finds this
  - From state 11, a1/b2/b4/b6
- Needle in a haystack
  - 14 states, length 4, 144 inputs,
  - 14 * 144^4 = 6,019,743,744
  - 1 out of 6+ billion sequences...



(a) Reference model



(b) Learned model

Scalability next steps

There are many other things you can try

- Abstractions
- Checkpointing
- Verification
- Parallel testing
- Greybox testing
- Whitebox testing
- …

And other things to consider

- But how well does it scale in practice? And is that enough?
- What else is needed to apply it in practice?

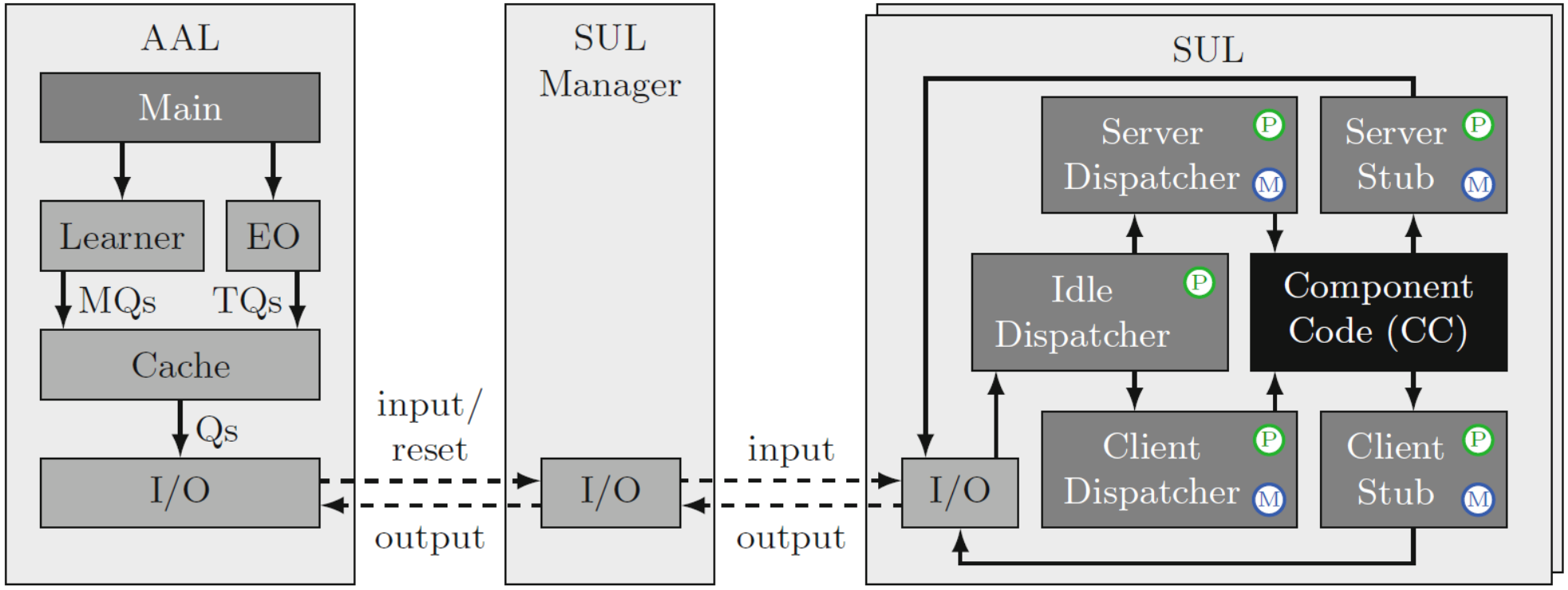
4. Active automata learning in industry: Application

Radboud University Nijmegen

ESI

ASML

Isolating components and connecting a learner



Legend:

OS process

Reusable code (configurable)

Generated code (based on IDL)

Original code

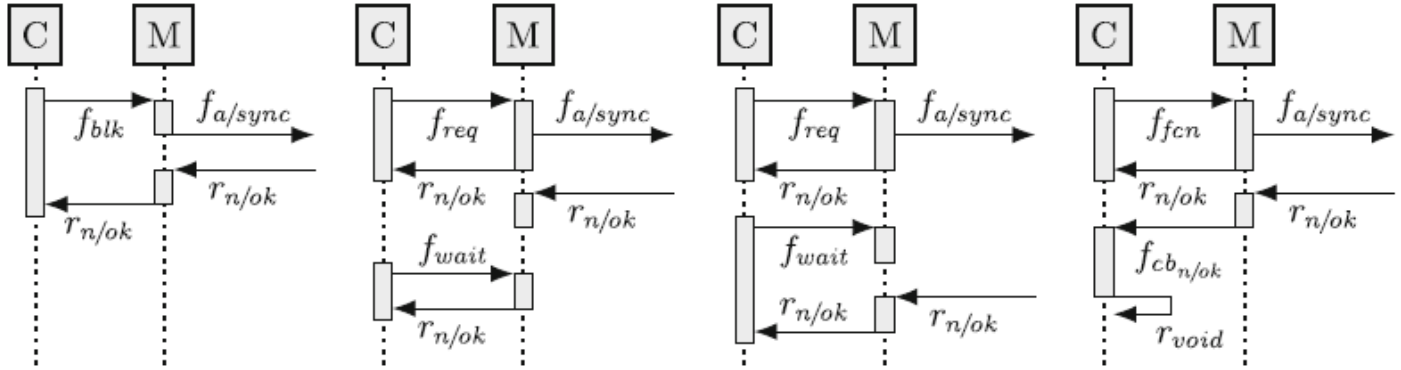
Socket I/O (one way)

Function call (with its return in opposite direction)

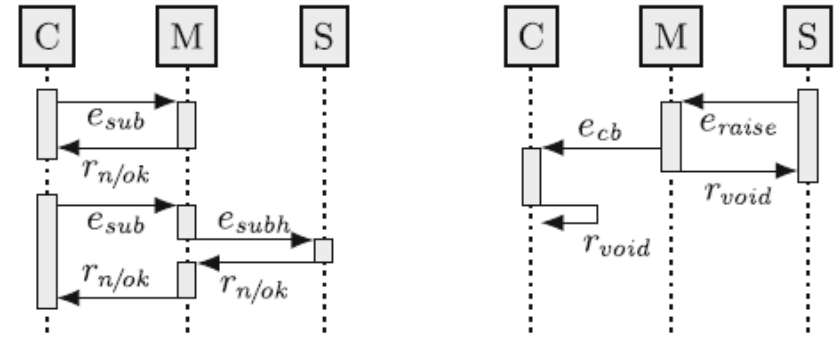
P Interfacing protocol

M Mapper

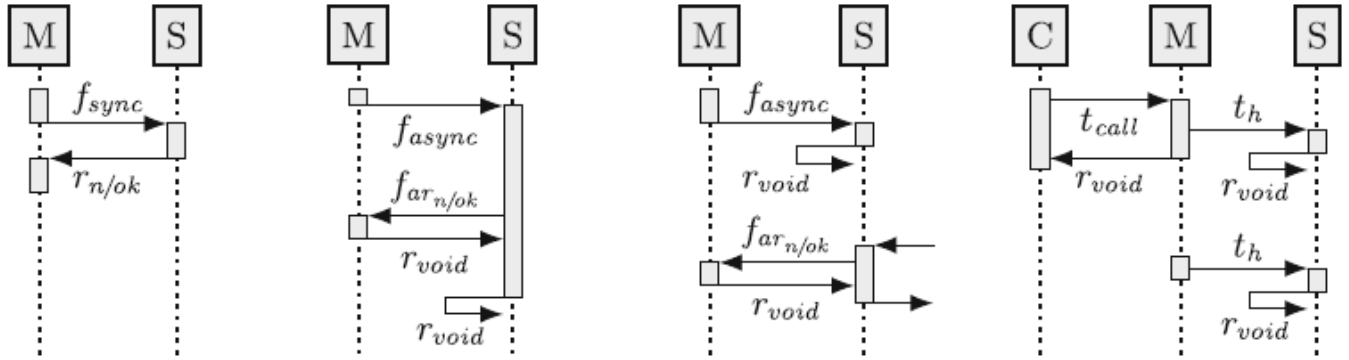
Many different forms of communications



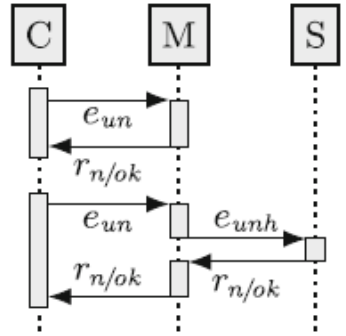
(a) Blocking call (b) Request/wait call (server finishes first) (c) Request/wait call (client waits first) (d) FCN call



(i) Event subscription (j) Event raise



(e) Synchronous handler (f) Asynchronous handler (direct) (g) Asynchronous handler (delayed) (h) Trigger



(k) Event unsubscription

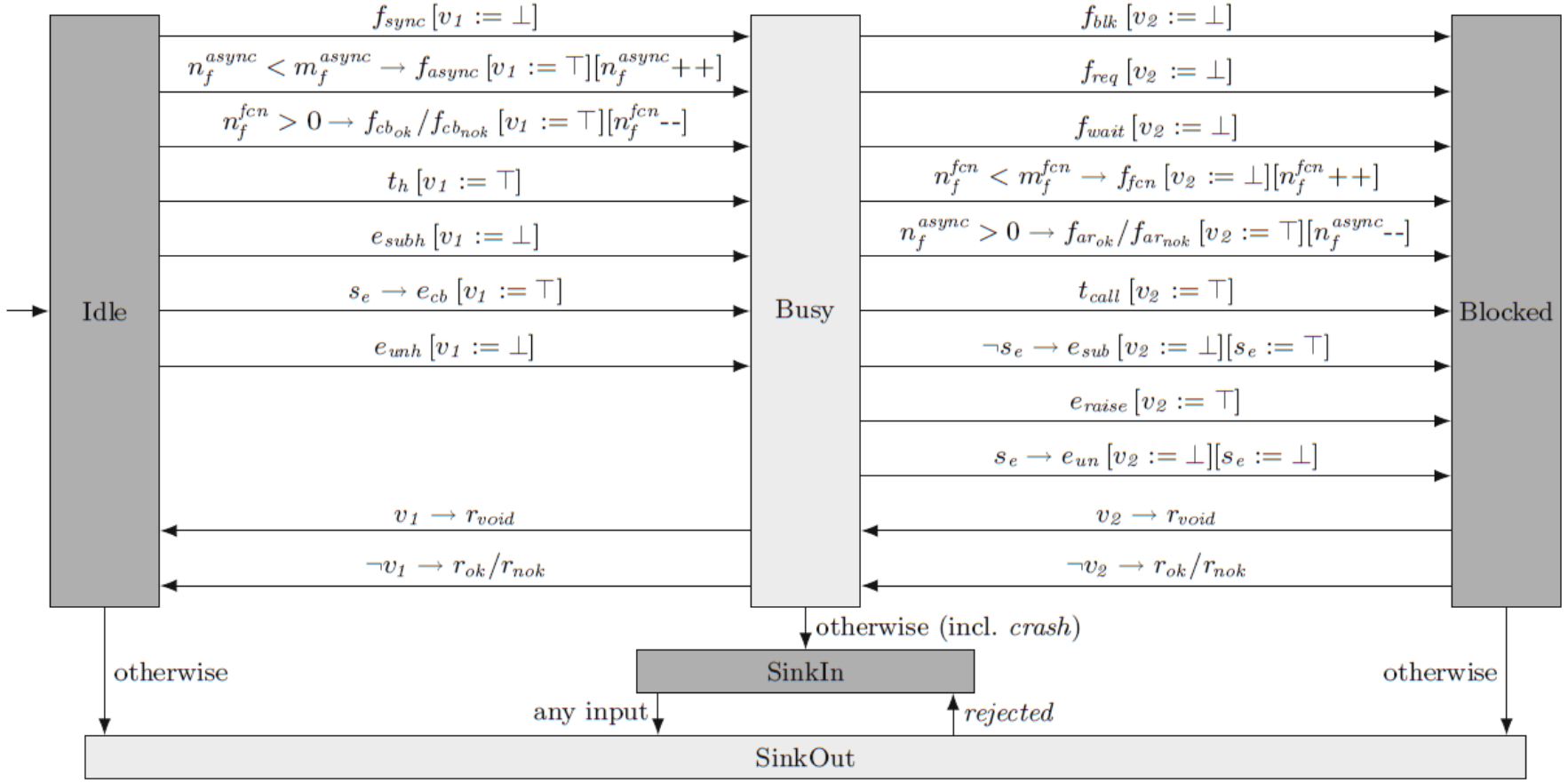
Mapping communication patterns to inputs/outputs

Communication pattern	Role of SUL	Message	Input/Output
(i) Blocking call	Client	f_{blk}	Output
		r_{ok} / r_{nok}	Input
(ii) Request/wait call	Client	f_{req}	Output
		r_{ok} / r_{nok}	Input
		f_{wait}	Output
		r_{ok} / r_{nok}	Input
(iii) FCN call	Client	f_{fcn}	Output
		r_{ok} / r_{nok}	Input
		$f_{cb_{ok}}$ / $f_{cb_{nok}}$	Input
		r_{void}	Output
(iv) Synchronous handler	Server	f_{sync}	Input
		r_{ok} / r_{nok}	Output
(v) Asynchronous handler	Server	f_{async}	Input
		r_{void}	Output
		$f_{ar_{ok}}$ / $f_{ar_{nok}}$	Output
		r_{void}	Input
(vi) Trigger	Client	t_{call}	Output
		r_{void}	Input
	Server	t_h	Input
		r_{void}	Output
(vii) Event	Client	e_{sub}	Output
		r_{ok} / r_{nok}	Input
		e_{cb}	Input
		r_{void}	Output
		e_{un}	Output
		r_{ok} / r_{nok}	Input
	Server	e_{subh}	Input
		r_{ok} / r_{nok}	Output
		e_{raise}	Output
		r_{void}	Input
		e_{unh}	Input
		r_{ok} / r_{nok}	Output

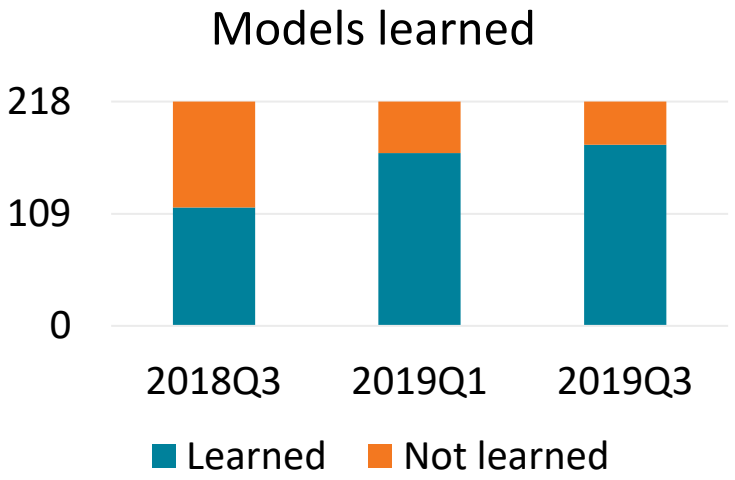
Interfacing protocol

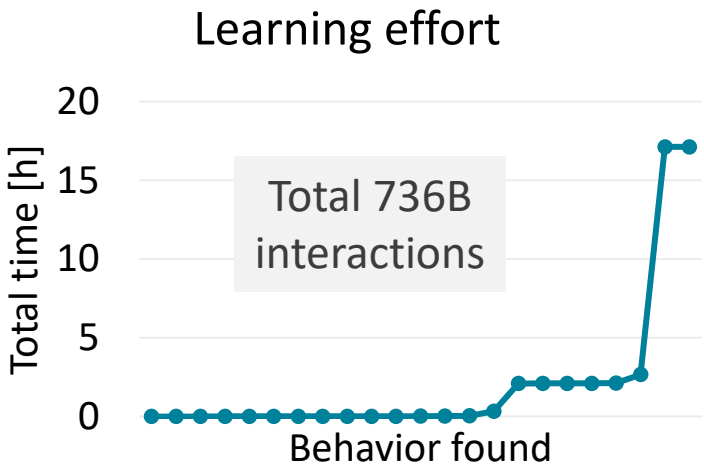
Responsibilities:

- Don’t send impossible inputs to the SUL
- Handling no/multiple outputs
- Ensure a finite learning process



Transposition project: the half-way point





State-of-the-art scalability

- Controlled scalability analysis
- Adopted/improved state-of-the-art
- But is it enough in practice?

Industrial application

- Application to real case
- Identified bottlenecks
- Further research needed

Shifting direction

- What can we do already now?
- Passive learning scales better
- Useful, even though incomplete

Want to know more about our model learning work?

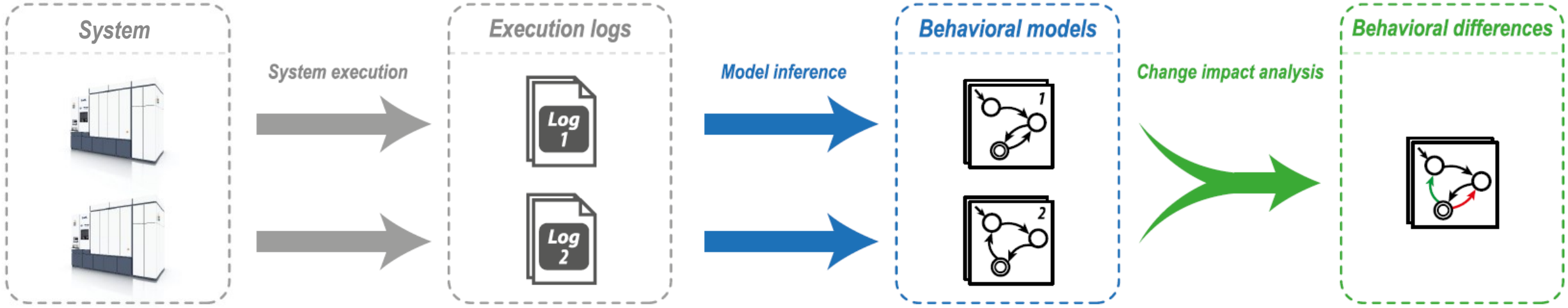
Kousar Aslam, "Deriving Behavioral Specifications of Industrial Software Components", PhD thesis, TU/e, 2021

Hendriks and Aslam, "A Systematic Approach for Interfacing Component-Based Software with an Active Automata Learning Tool", ISoLA, 2022

Hooimeijer et al., "Constructive Model Inference: Model Learning for Component-based Software Architectures", ICSOFT, 2022

5. Preventing regressions for software changes

Approach to prevent regressions for software changes



- Automatically infer behavioral models
- Provide insight into software behavior
- Essential for making correct software changes

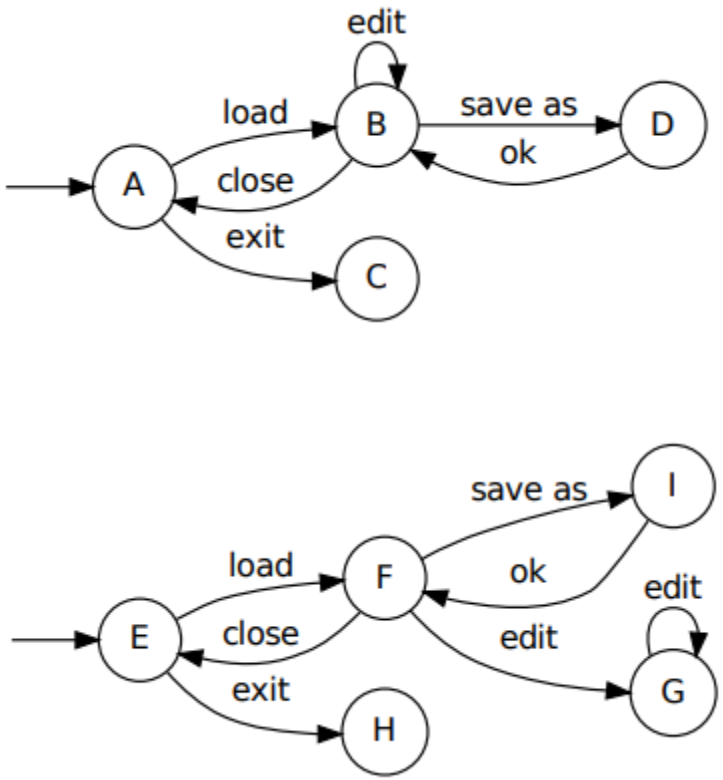
- Automatically compare behavioral models
- Confirm expected differences and potential regressions
- Quick change impact analysis for system behavior

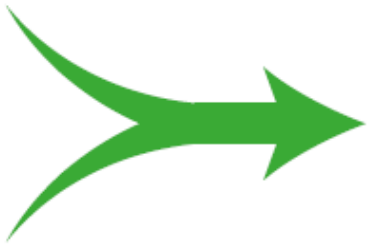
Improve development efficiency and reduce risks

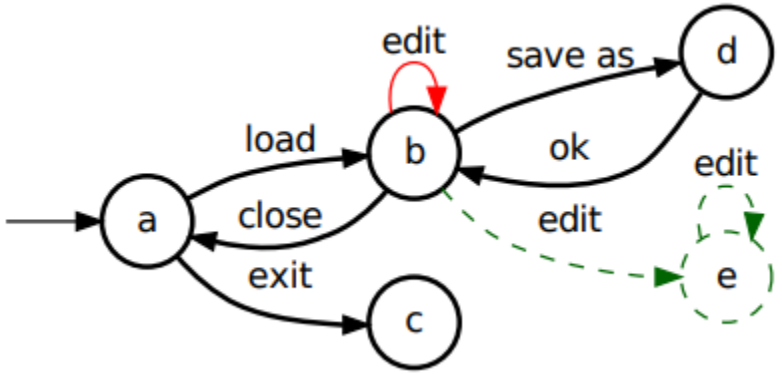
Want to know more about this?

Hendriks et al., "A Multi-level Methodology for Behavioral Comparison of Software-Intensive Systems", FMICS, 2022

Behavioral comparison of state machine structures: example

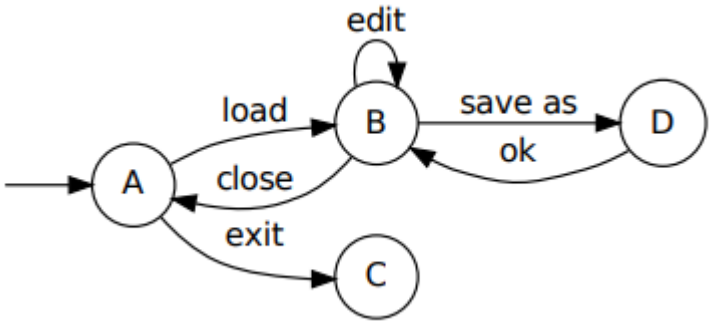


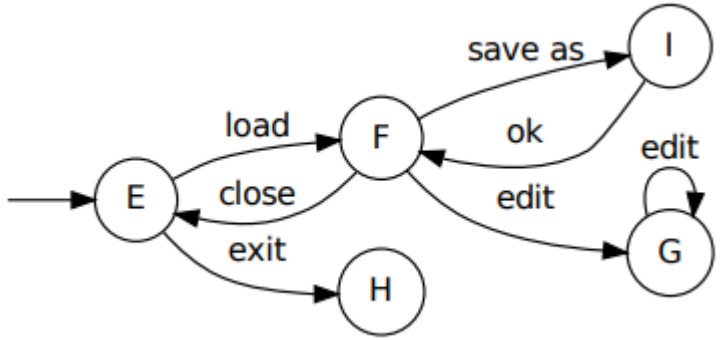




Behavioral comparison of state machine structures: LTSDiff

1. **Compute similarity scores**
   a) For every pair of states in source and target state machines
   b) Local scores based on incoming and outgoing edges
   c) Global scores with attenuation factor for further away edges
2. **Compute a matching**
   a) Start with key pairs (landmarks), e.g. top 25% best scoring pairs
   b) Fallback to initial states
   c) Match surrounding areas, moving outwards
3. **Compute difference state machine**
   a) Matching states and transition vs unmatched ones

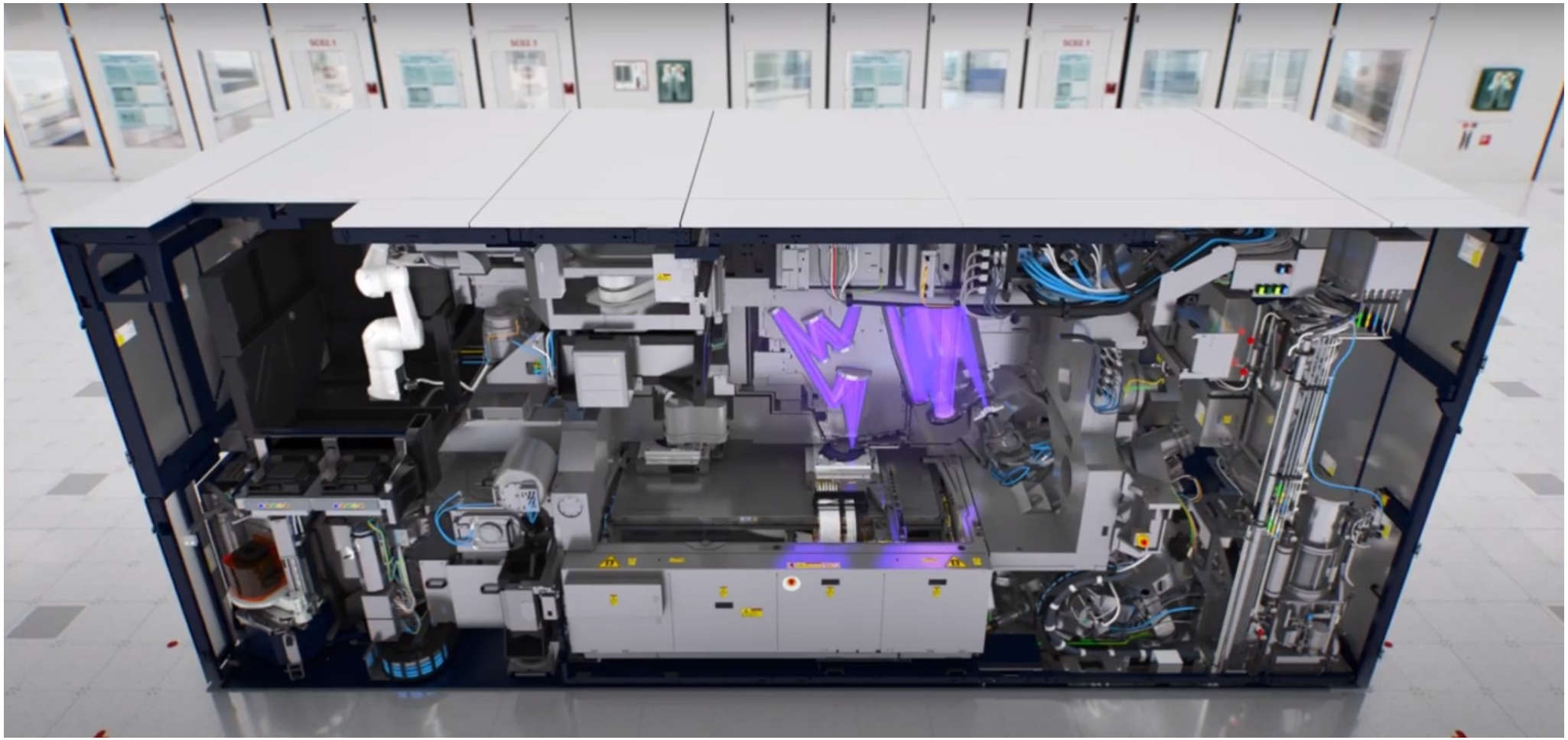




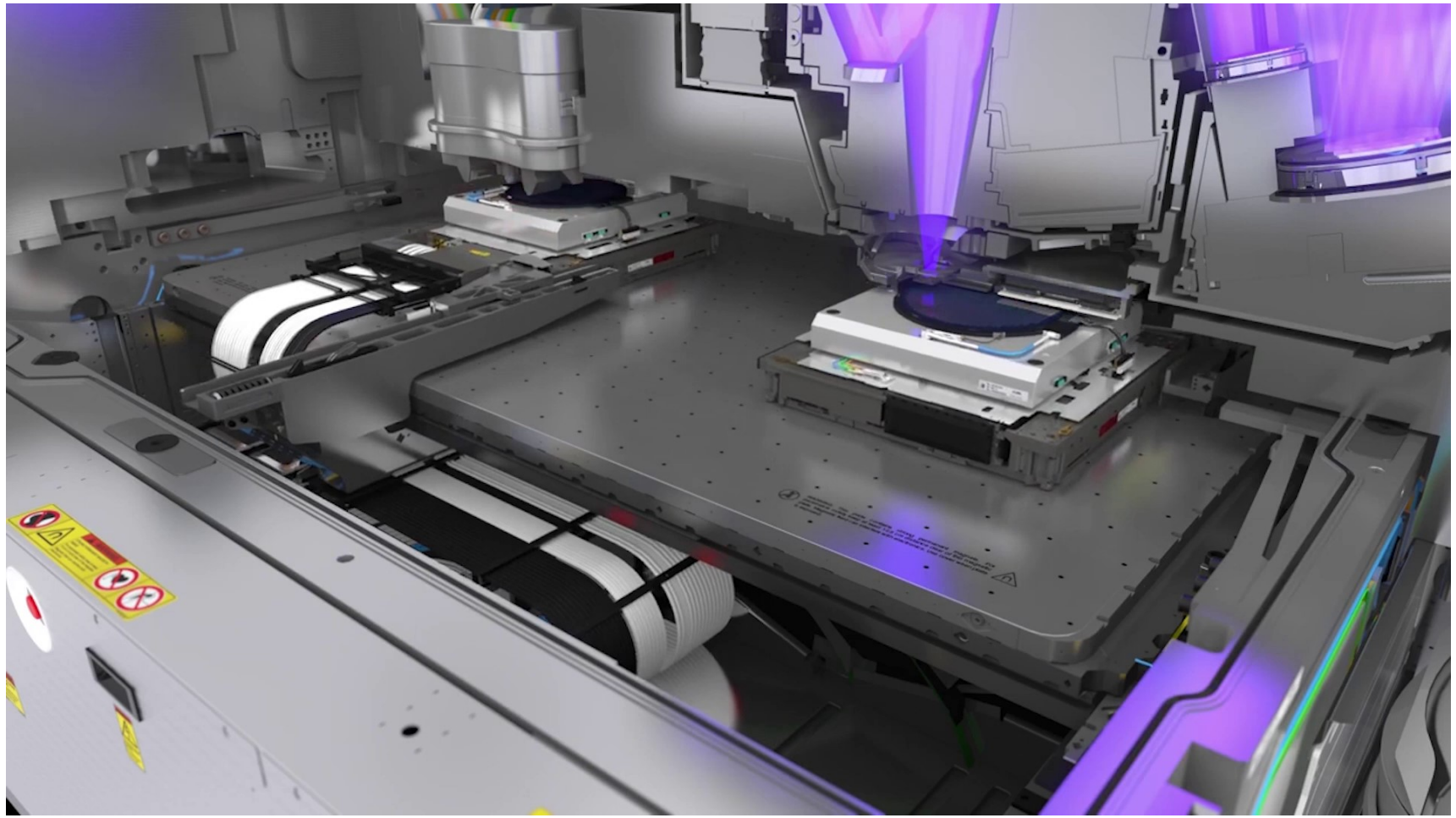
Want to know more about this?

Walkinshaw and Bogdanov, "Automated comparison of state-based software models in terms of their language and structure", TOSEM, volume 22, number 2, 2013

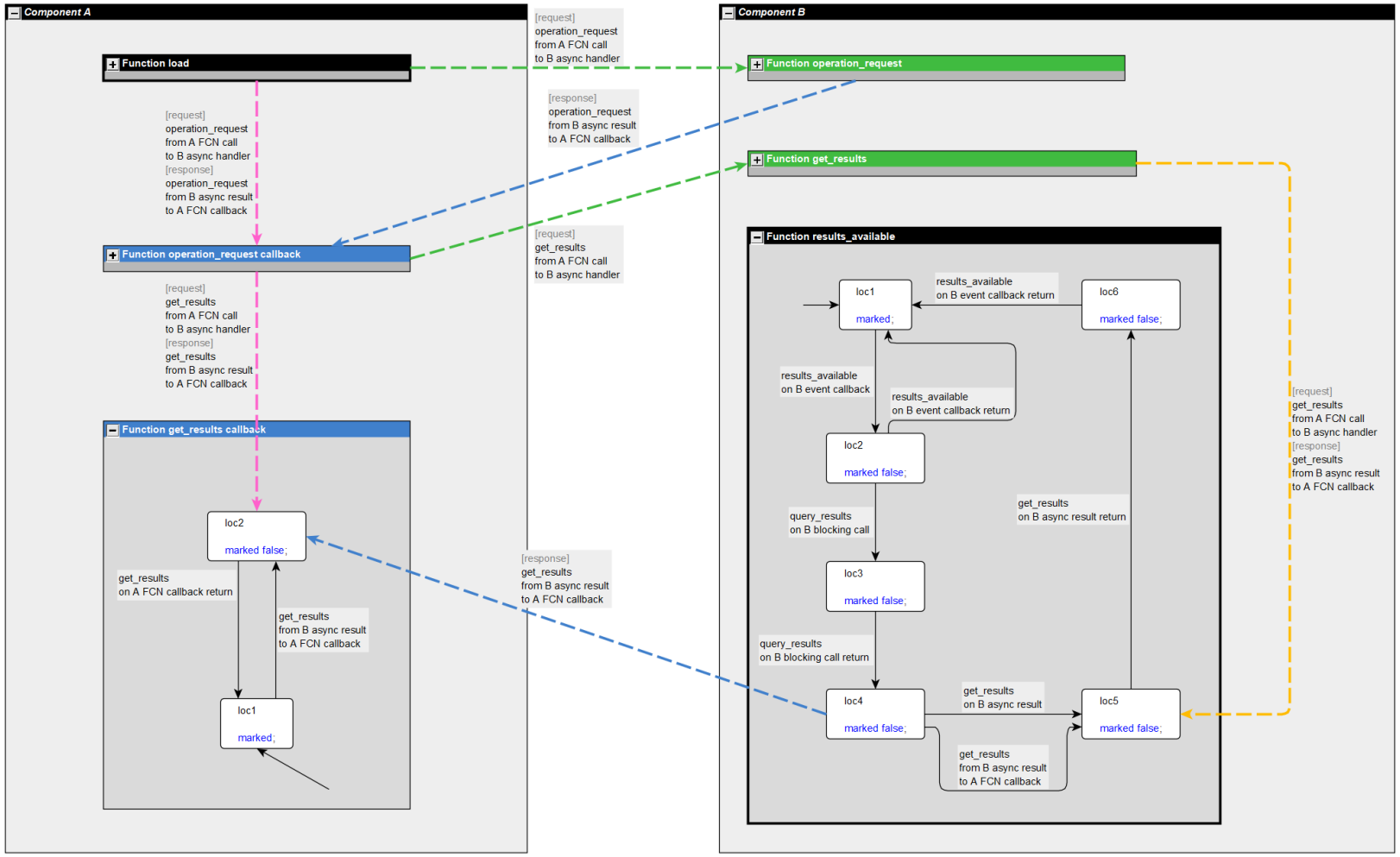
ASML's TWINSCAN system



ASML's TWINSCAN system



1) Inferred models at multiple levels of detail



"The ability to analyze behavior of our software at a very high level of abstraction, as well as zooming in on the details is extremely cool."

"You can really see how it behaves, for this configuration."

2a) Change impact analysis: no differences for a redesign

- High level chuck swap controller
- End-of-life technology in legacy implementation replaced by new implementation

	Before	After
record_exception	A	A
adjust_configuration	A	A
configuration_changed	A	A
request_idle	A	A
get_initial_results	A	A
get_initial_results_sync	A	A
obtain_status	A	A
obtain_status_sync	A	A
register_for_queued_actions	A	A
deregister_for_queued_actions	A	A
flush_trigger	A	A
fixate_config	A	A
determine_remaining_work	A	A
find_current_work	A	A
report_current_status	A	A
report_current_timing	A	A
perform_operation	A	A
flush_operation_cache	A	A
prepare_measurement	A	A
perform_measurement	A	A
report_on_measurements	A	A
process_identifiers	A	A
find_component_identifier	A	A
specify_component_identifier	A	A
specify_process_identifier	A	A
perform_progress_step	A	A
persist_progress_info	A	A
persist_component_info	A	A
operation_request	A	A
reset_operation_status	A	A
handle_operation_reset	A	A
handle_data_persist	A	A
obtain_storage_info	A	A
identifier_storage	A	A
ogress_status	A	A
ensor	A	A
_synchronization	A	A
ck_ids__blk_sync	A	A
initiate_sensor_data_store	A	A
signal_component_action	A	A
obtain_signal_status	A	A
prepare_sensor_storage	A	A
finish_operation	A	A
prepare_print_queue_content	A	A
handle_print_queue_content	A	A
determine_input_parameters	A	A
fixate_input_parameters	A	A
continue_synchronizing	A	A
synchronize_operations	A	A
proceed_operation	A	A
hande_operation_shift	A	A
print_shifting_state	A	A
export_measurements	A	A
request_measurement_export	A	A
handle_measurement_export	A	A
signal_operation_start	A	A
abort_operation	A	A

“Manual analyses that normally take days can now be completed in hours automatically.”

“This gives sufficient confidence […]”

Legend

A 1st behavior for function

2b) Change impact analysis: quickly identify potential regressions

- High level expose controller
- One line code change caused a regression

	Before	After
obtain_budget_identifier *(function 1 of 113)*	A	A
check_memory_state *(function 2 of 113)*	A	A
renew_memory_state *(function 3 of 113)*	A	A
get_printer_characteristics *(function 4 of 113)*	A	A
get_component_name *(function 5 of 113)*	A	A
get_current_active_task *(function 6 of 113)*	A	A
set_task_started *(function 7 of 113)*	A	A
load_configuration *(function 8 of 113)*	A	A
store_configuration *(function 9 of 113)*	A	A
perform_plate_assignment *(function 10 of 113)*	A	A
prepare_lens_config *(function 11 of 113)*	A	A
calibrate_level *(function 12 of 113)*	A	A
fixate_measure *(function 13 of 113)*	A	A
signal_printer *(function 14 of 113)*	A	A
prepare_queue *(function 15 of 113)*	A	A
validate_config *(function 16 of 113)*	A	A
obtain_operation_state *(function 17 of 113)*	A	A
operation_queued *(function 18 of 113)*	A	B
report_phase *(function 19 of 113)*	A	A
persist_state *(function 20 of 113)*	A	A
retrieve_error_code *(function 21 of 113)*	A	A
collect_corrections *(function 22 of 113)*	A	A
start_operation *(function 23 of 113)*	A	B
detect_settings_change *(function 24 of 113)*	A	A
sync_measurements *(function 25 of 113)*	A	A
clear_buffers *(function 26 of 113)*	A	A
terminate_measurements *(function 27 of 113)*	A	A
evaluate_measurements *(function 28 of 113)*	A	A
validate_measurements *(function 29 of 113)*	A	A
reset_mark *(function 30 of 113)*	A	A
perform_sensor_corrections *(function 31 of 113)*	A	A
preload_setup_tasks *(function 32 of 113)*	A	A
determine_position *(function 33 of 113)*	A	A
start_synchronization *(function 34 of 113)*	A	A
finalize_configuration *(function 35 of 113)*	A	A

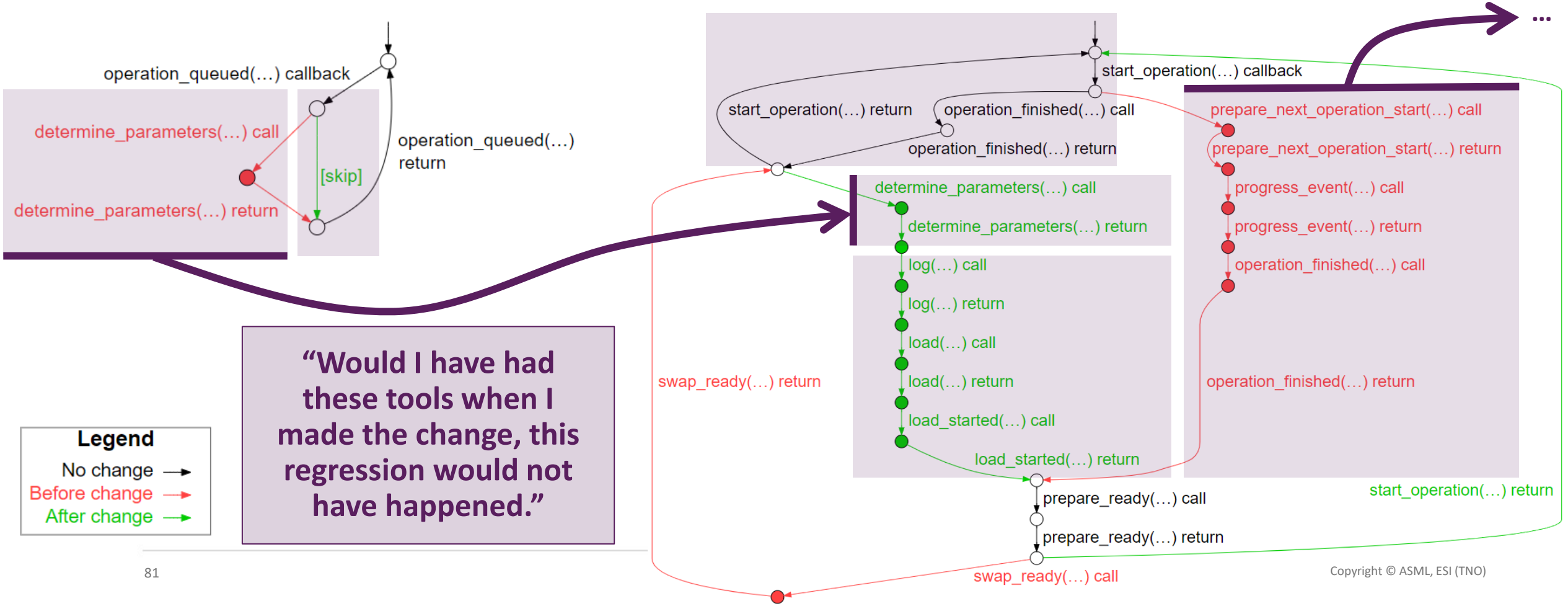
Legend

- A: 1st behavior for function
- B: 2nd behavior for function

2b) Change impact analysis: inspect the regression in detail

operation_queued callback

start_operation callback

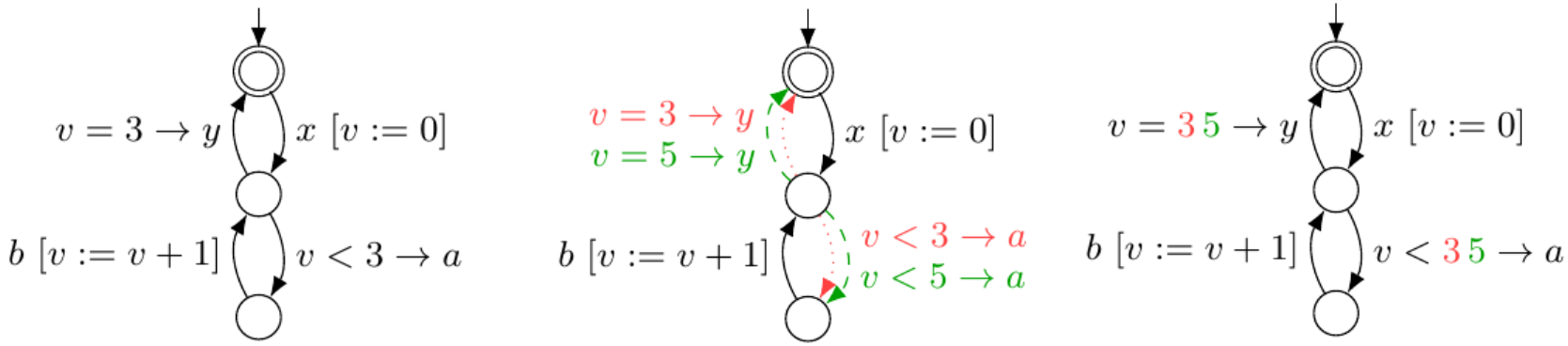


"Would I have had these tools when I made the change, this regression would not have happened."

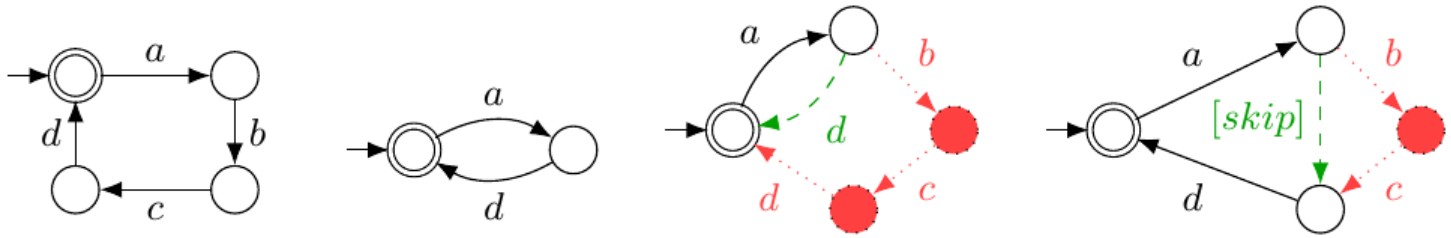
What's next for comparison of software behavior?

gLTSdiff: generalized version of LTSDiff

- Compare not only LTSs, but any state machine representation (NFAs, EFAs, FFSMs)



- More intuitive differences (e.g., less duplication)



- …

MSc graduation project at TNO-ESI available

https://sws.cs.ru.nl/Teaching/ChangeImpactAnalysis

Theme: Change Impact Analysis

Software-intensive systems constantly evolve. To prevent software changes from unintentionally introducing costly system defects, it is important to both understand how the current software behaves, and what impact changes have on that behavior. This helps to reduce risks.

For more information, you can:

- Watch this 8 minute video.
- Read the published paper.

The following MSc graduation projects currently have an open vacancy:

- [2022-11-15] Intuitive insights into the impact of software changes on system behavior at TNO-ESI.

If you're interested, contact Dennis Hendriks or apply directly.

6. Conclusions

Conclusions

Software behavior

- Software behavior models serve many purposes — e.g. insight, preventing regressions, MBT
- They should contain all relevant aspects — i.e. order, data, time, etc

Model learning

- Active automata learning is very promising — to learn the complete behavior
- For the short term, we use passive learning — easier to apply, but typically incomplete
- For the long term, best to combine multiple methods — use their complementary strengths

Theory vs practice

- Fundamental research is different from applied research — e.g. algorithms to choose, practicalities
- A non-perfect solution can be extremely valuable in practice — e.g. change impact analysis for ASML

Radboud University Nijmegen

TU/e

ESI

ASML

This research is a collaboration of TNO-ESI, ASML, Radboud University, Eindhoven University of Technology (TU/e) and University of Stuttgart.
This research is partly carried out as part of the Transposition project under the responsibility of TNO-ESI in co-operation with ASML. The research activities are supported by the Netherlands Ministry of Economic Affairs and TKI-HTSM.