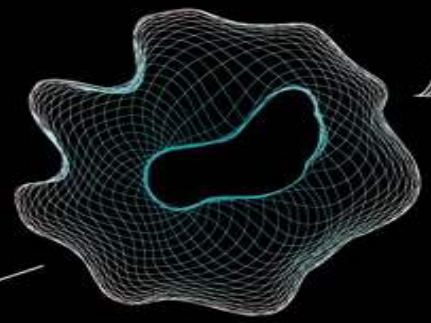
UNIVERSITY OF TWENTE.

VerCors

BUG-FREE SOFTWARE: A REALISABLE DREAM?

MARIEKE HUISMAN
UNIVERSITY OF TWENTE, NETHERLANDS



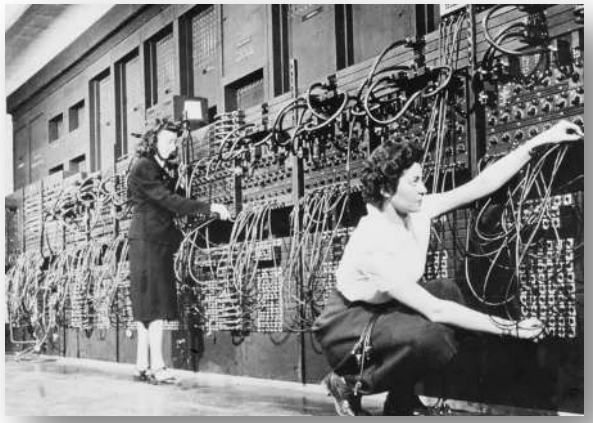
SAVES

CHEOPS
VERIFIED CONSTRUCTION OF CORRECT AND OPTIMISED PARALLEL SOFTWARE

Mercedes
MAXIMAL RELIABILITY OF CONCURRENT AND DISTRIBUTED SOFTWARE

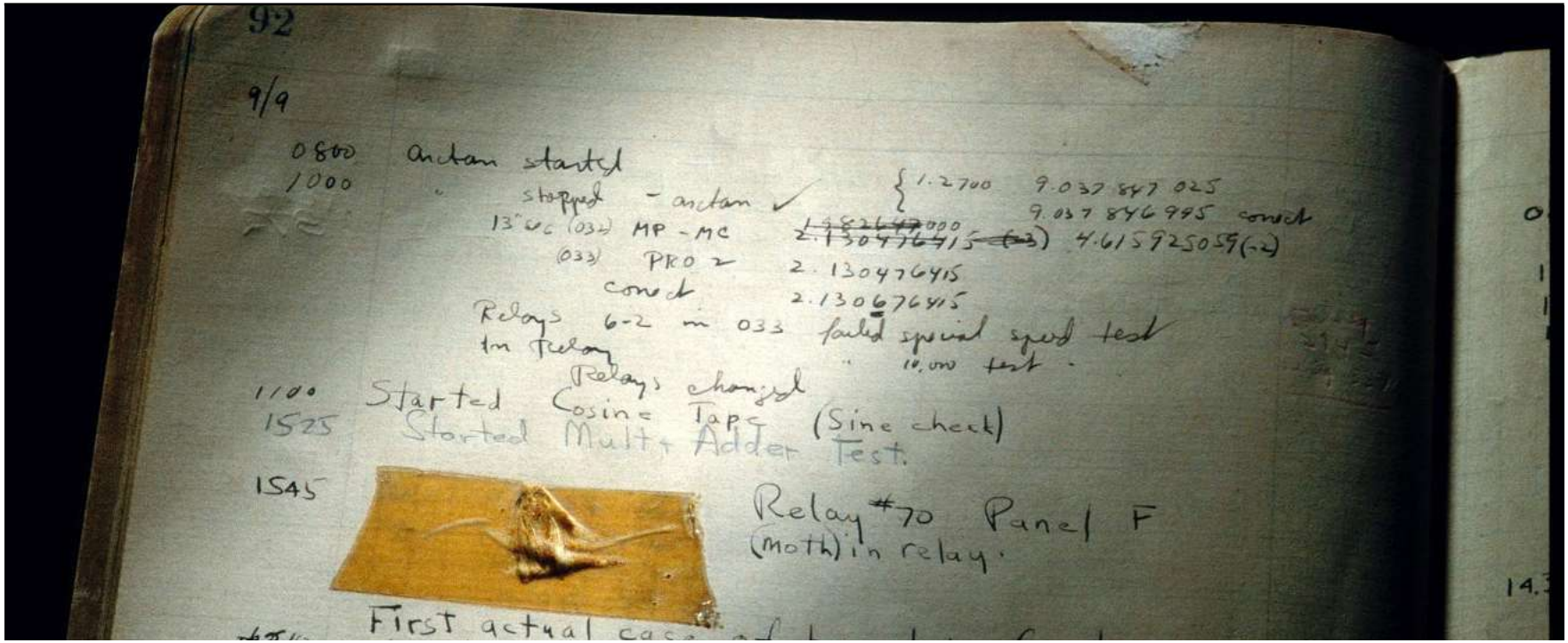
Pallas

SOFTWARE IS EVERYWHERE

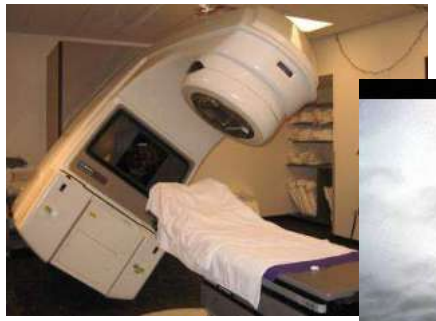


1946 now

BUGS ARE JUST AS OLD AS SOFTWARE



A HISTORY OF SOFTWARE ERRORS



Therac 25, 1982

Ariane 5, 1996

Ramspoelbridge, 2016

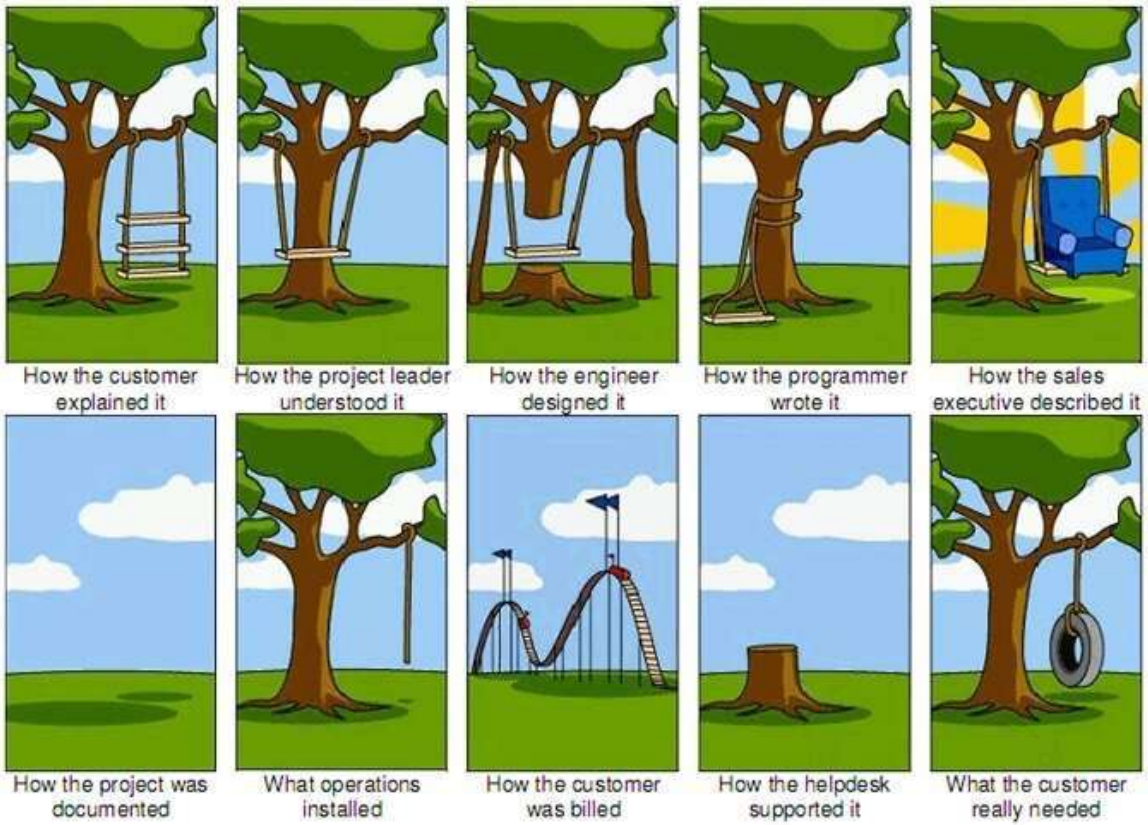
June 2019

Cloudstrike, July 2024

PLAN FOR THIS TALK

- Why do we have software bugs?
- How could formal methods help?
- My favourite formal method: VerCors
- How to realise the dream?

SOURCE OF THE PROBLEM?



SOLVING THE PROBLEM…

- Education: well-trained programs
- Follow clear procedures
- Spend time on requirements, design
- Document decisions
- Correctness should not be an afterthought: detect problems during development

WHY FORMAL METHODS CAN HELP

Formal specification of intended behaviour

- Make implicit assumptions explicit and unambiguous
- Make it possible to use tools to check if the implementation indeed has the intended behaviour
- Make it possible to guarantee that behaviour is preserved after update (or that patch fixes the behaviour)

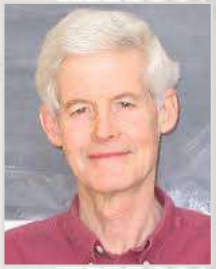
PILLARS OF FORMAL VERIFICATION

LinkedIn post
Amirfarhad Nilizadeh
February 2025



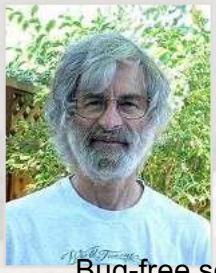










SUCCESS STORIES FORMAL METHODS

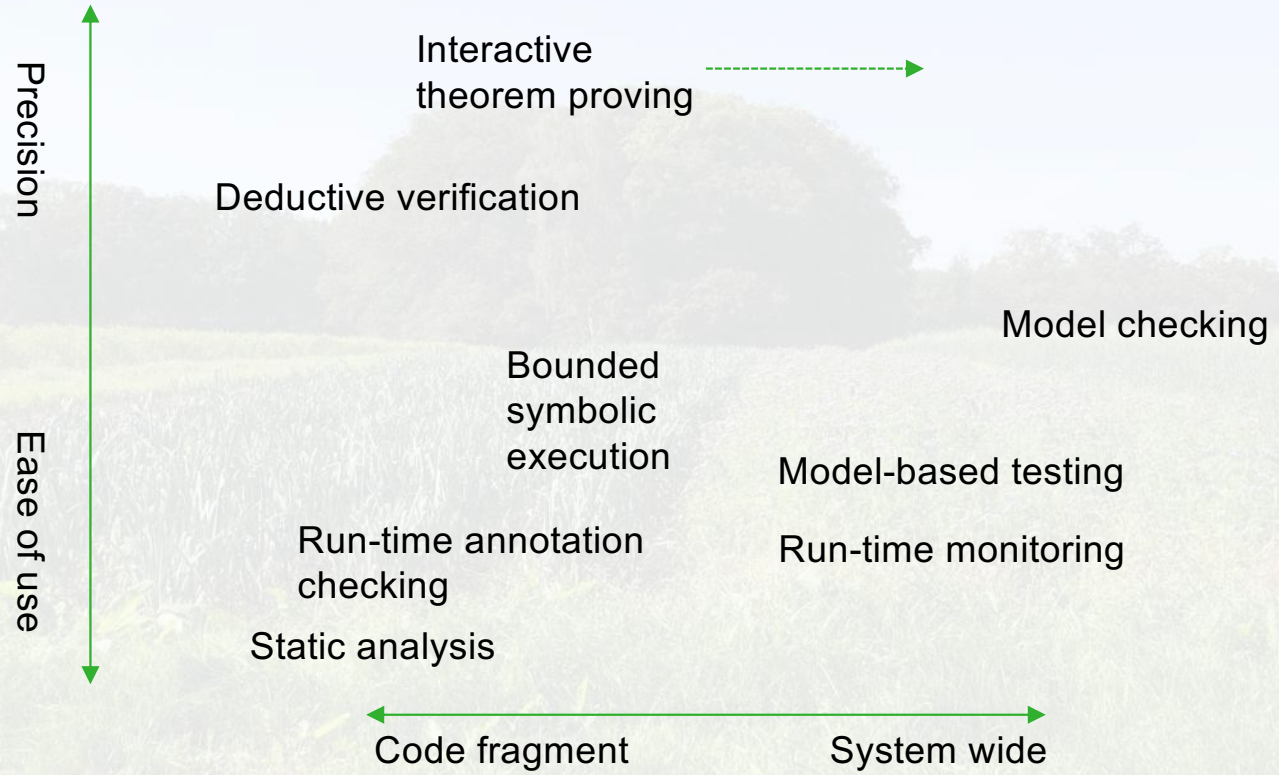
- Railway systems
- Driverless metro systems
- Smart cards
- Software tools as Microsoft's Code Contracts
- Bug in TimSort
- Swarm Robotics
- Infer@Facebook in Continuous Integration

HOW DO WE APPLY FORMAL METHODS?

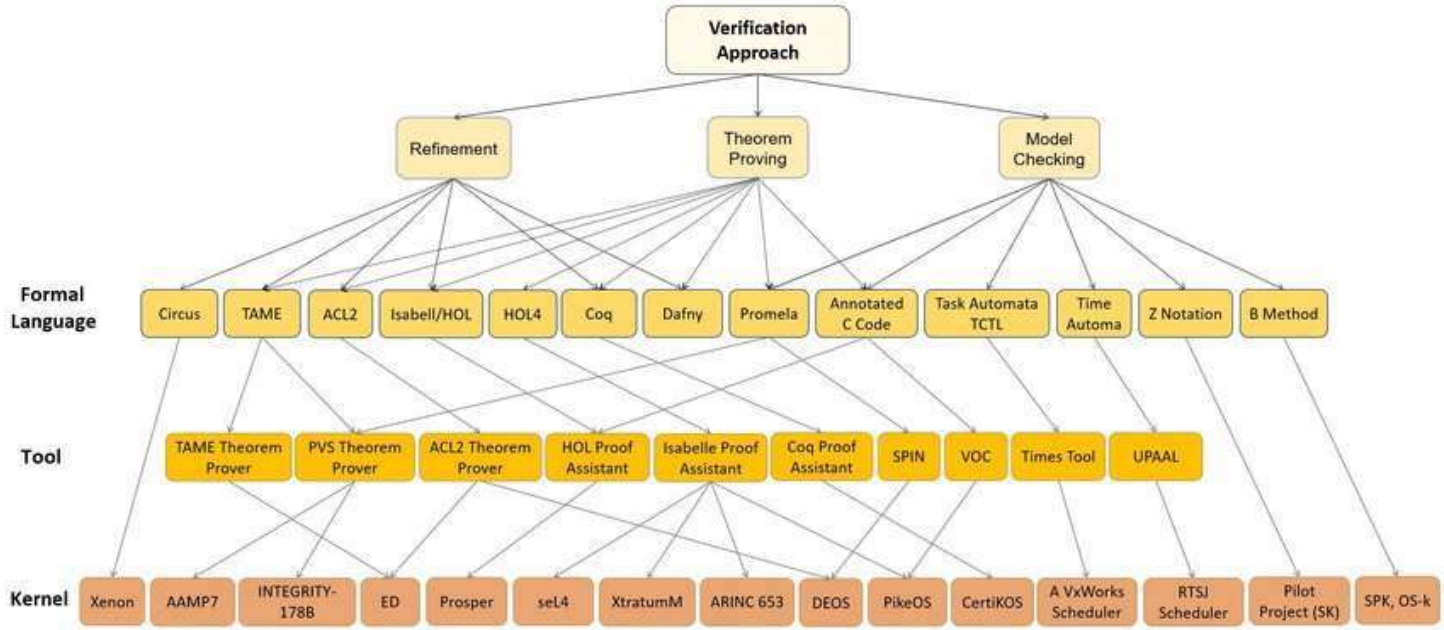
- Write formal specification
  - At different levels:
    - For code fragment
    - System-wide
  - Different kinds of property
    - About one particular program state
    - About how a system evolves over time (behavioural properties)

Validate implementation w.r.t. specification

THE FORMAL METHODS LANDSCAPE



WIDE RANGE OF FORMAL METHODS AND TOOLS



Source: Is formal verification of seL4 adequate to address the key security challenges of kernel design?
Mina Soltani Siapoush, Jim Alves-Foss, IEEE Access, 2023

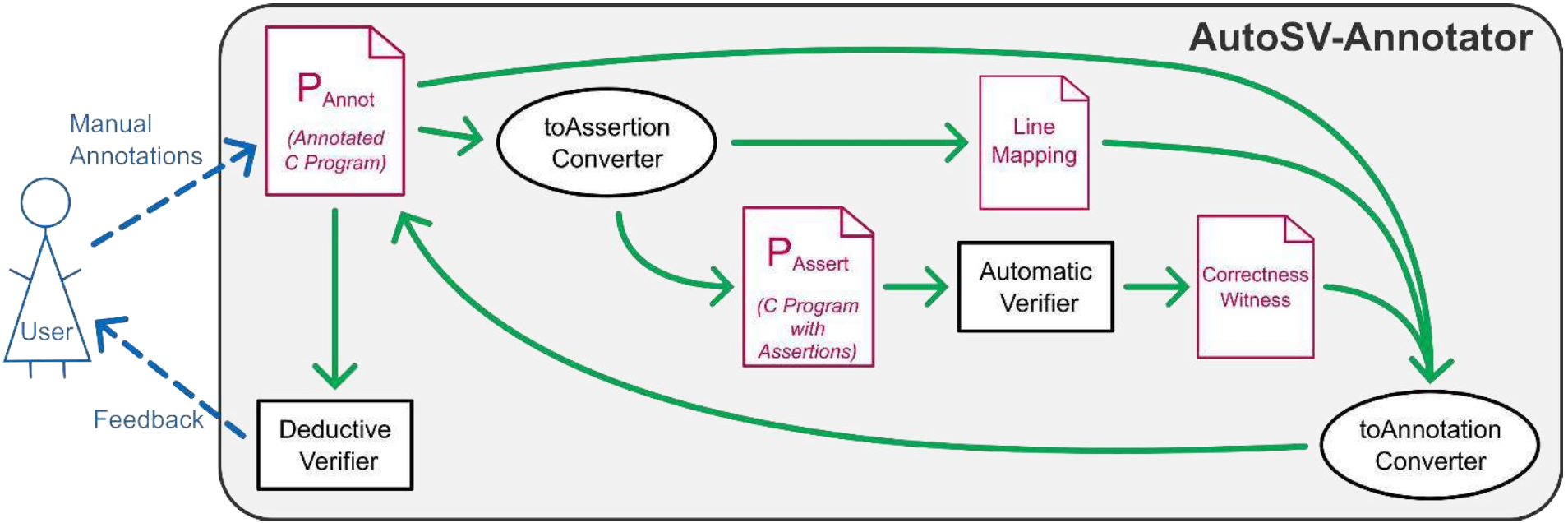
CHALLENGES OF FORMAL METHODS

Applying formal methods does not come for free

- Requires expertise
- Requires specifications (is extra work)
- Current tools have limitations

Armborst et al. FMICS 2025

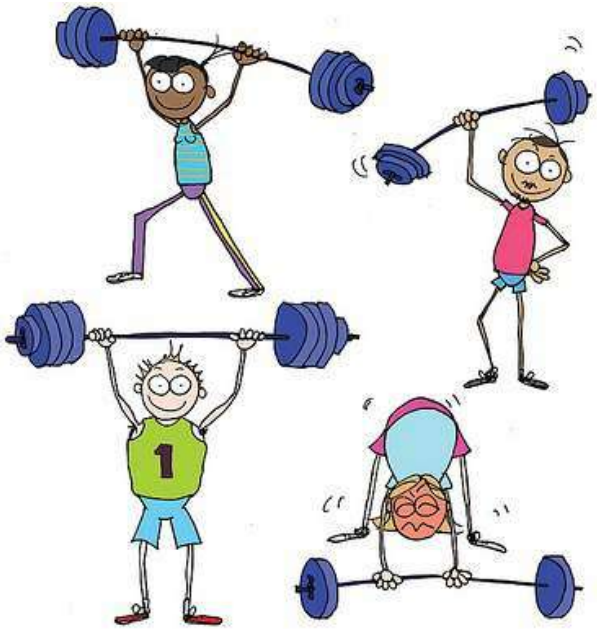
COOPERATIVE VERIFICATION



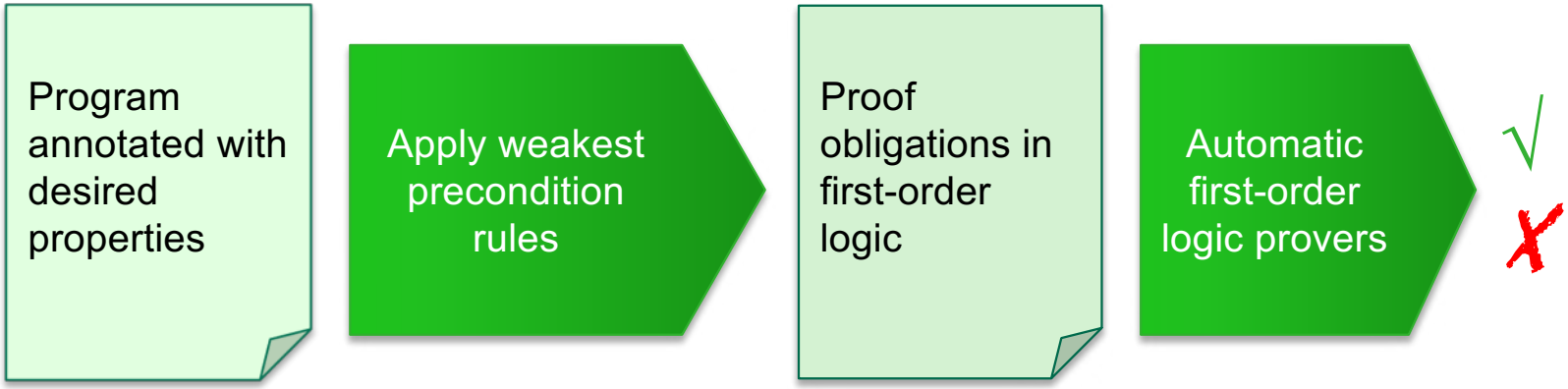

VerCors

REASONING ABOUT CODE: POWER

```
{ a ≥ 0 ∧ n ≥ 0 }
k := 0;
z := 1;
{Inv: z = a^k ∧ k ≤ n ∧ a ≥ 0 ∧ k ≥ 0}
while (k < n){
    z := z * a;
    k := k + 1;
}
{ z = a^n }
```



DEDUCTIVE PROGRAM VERIFICATION



Preferably also counterexample:

why does program not have desired behaviour

First order logic: atomic propositions, and, or, implies, quantifiers

VERIFICATION OF CONCURRENT SOFTWARE

- Permission-based Separation Logic
- Separation logic for sequential Java
- Concurrent Separation Logic (with variations/extensions)
- Permissions
- JML specifications
- Dynamic frames
- ...

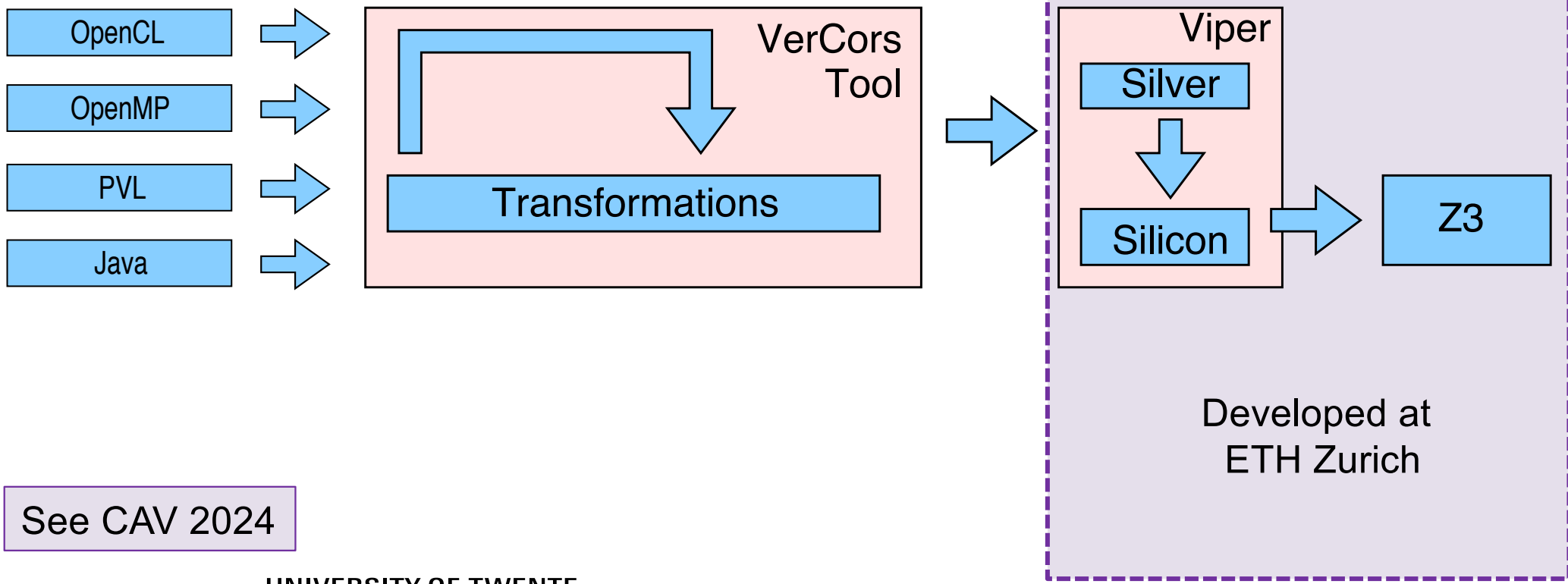
Separation logic developed to reason about programs with pointers

VerCors

REASONING WITH PERMISSIONS

- Permissions: fractional value between 0 and 1
  - Write permission: exclusive access (encoded by 1)
  - Read permission: shared access (encoded by fractional value between 0 and 1)
- Global invariant: for each heap location, the sum of all the permissions in the system is never more than 1
- Permission specifications frame functional properties
- Read and write permissions can be exchanged whenever threads synchronise

VERCORS TOOL ARCHITECTURE



See CAV 2024

VERCORS HIGHLIGHTS

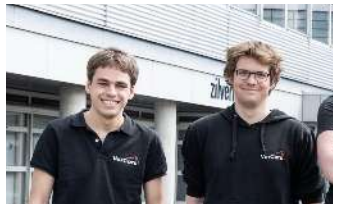
Automated verification of concurrent software

Different concurrency programming languages and paradigms

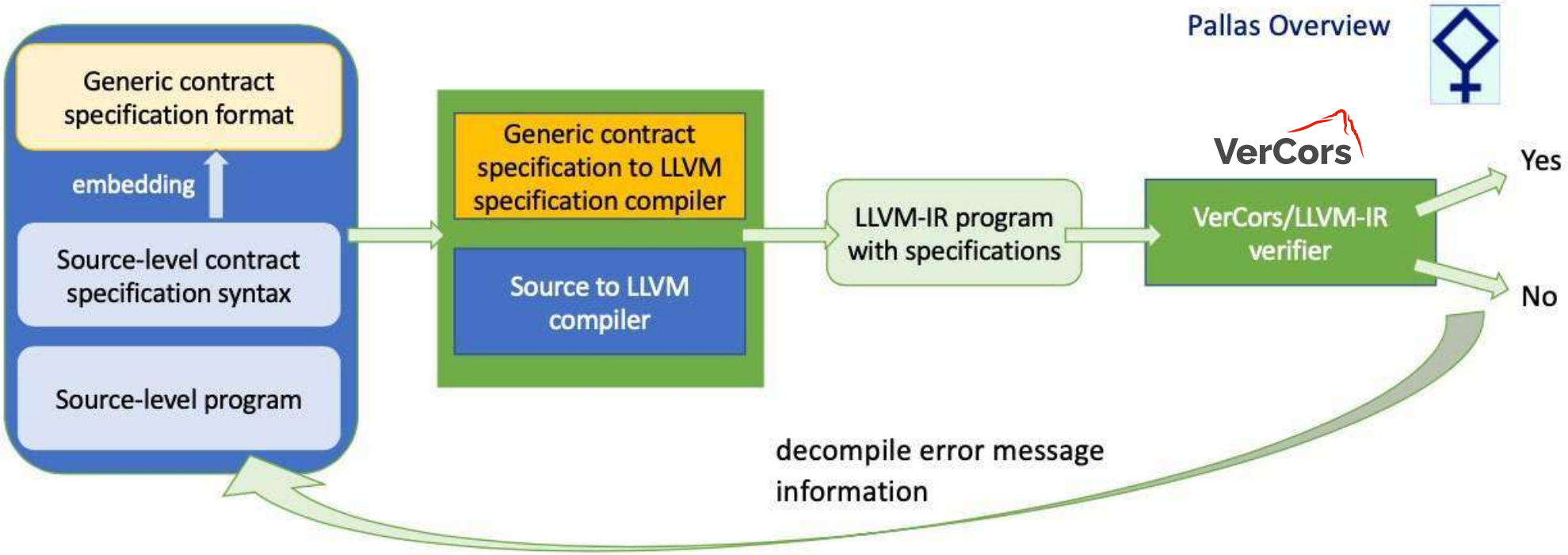
VERIFICATION TECHNOLOGIES

- Correctness preservation of program transformations (optimiser, compiler) [TACAS 2022, 2024]
- Generation of correct executable code from high-level abstractions [iFM 2024]
- Annotation generation and specification translation [iFM 2023, JSS 2024, HCVS 2024]
- Verification of parametrised embedded systems applications [SEFM 2024]

PALLAS: OVERALL IDEA



Alexander Stekelenburg & Robert Mensing



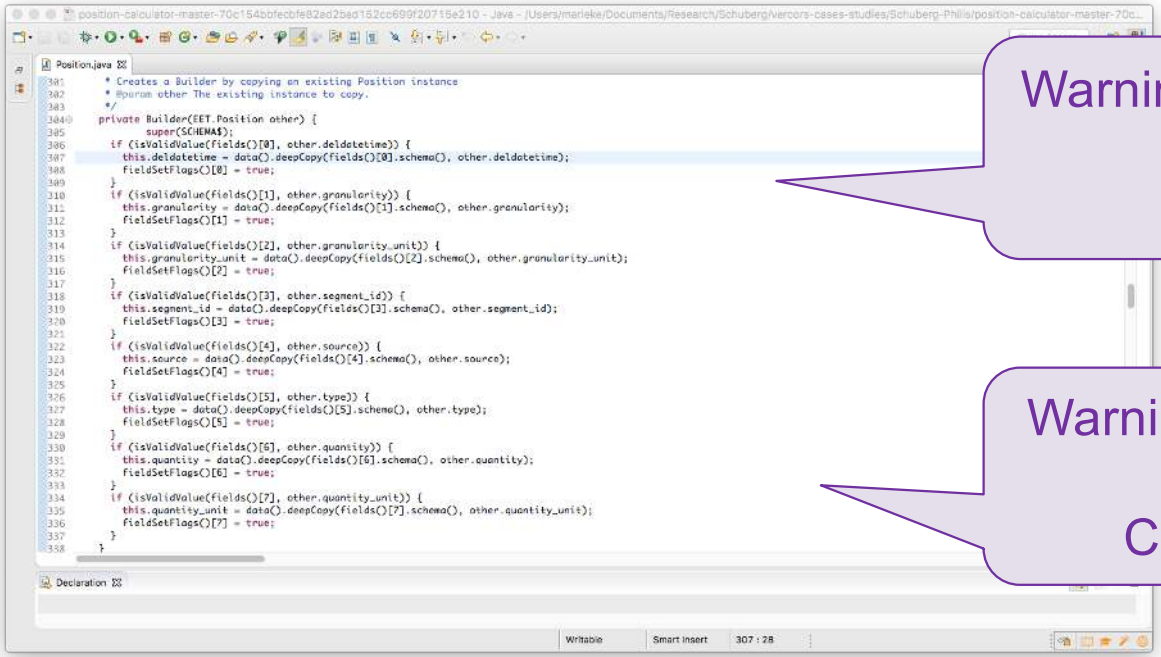
VERCORS CASE STUDIES

- GPU examples
  - Prefix sum
  - Summed area table
  - Parallel Bellman-Ford Algorithm
- Parallel model checking algorithms
  - Parallel nested depth-first search
- Industrially-inspired case studies
  - Tunnel control software
  - Red-black tree and parallel merge
  - Distributed locks
  - ArrayList

Blankenburgverbinding


TO THE FUTURE
TM

SEAMLESS INTEGRATION OF VERIFICATION



Warning: this method does not have intended behaviour.
Click here for a counterexample

Warning: at this point a nullpointer exception can occur.
Click here for an example execution

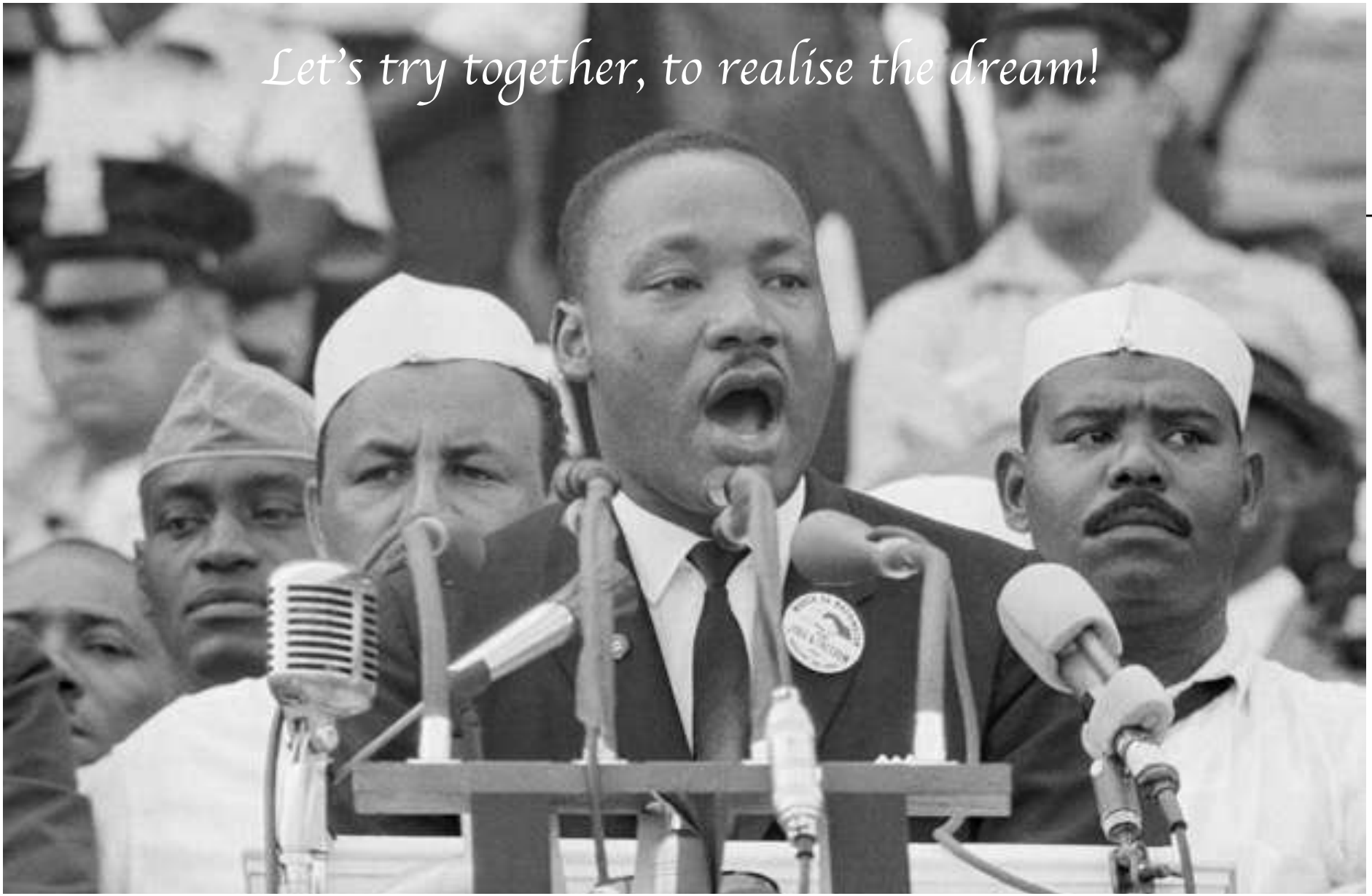
WHAT CAN I DO?

- Make verification work for more than toy examples
- Improve automation (and maybe AI can help)
- Improve feedback and usability

WHAT CAN YOU DO?

- Spend time on training and obtaining the necessary expertise
- Provide case studies
- Be ready to experiment
- Don’t expect me to write a tool that can be used off-the-shelf

This might feel like an investment, but you will benefit from it!


Let's try together, to realise the dream!

Afshin Amighi, Lukas Armborst, Stefan Blom, Petra van den Bos,
Pieter Bos, Saeed Darabi, Lars van den Haak, Paula Herber, Sebastiaan Joosten,
Sophie Lathouwers, Robert Mensing, Raúl Monti, Wojciech Mostowski,
Henk Mulder, Wytse Oortwijn, Bob Rubbens, Ömer Şakar, Alexander Stekelenburg,
Philip Tasche, Naum Tomov, Anton Wijs, Ellen Wittingen, Marina Zaharieva, and many BSc and MSc students

ACKNOWLEDGEMENTS

THE END...

Automated verification of concurrent software

More information and try the tool:
http://www.utwente.nl/vercors
m.huisman@utwente.nl