On Teaching Formal Methods:

Behavior Models and Code Analysis

Jan Kofroň
Ondřej Šerý
Pavel Parízek

DISTRIBUTED SYSTEMS RESEARCH GROUP
CHARLES UNIVERSITY IN PRAGUE
CZECH REPUBLIC

Dependable systems study plan

- **Goal:** Teach students to efficiently construct dependable and predictable systems
  - W.r.t. functional but also non-functional properties
  - Using state-of-the-art methods and tools

- **This talk:** About courses focused on techniques to construct functionally correct systems

- **Target audience:** Graduate students
  - Prerequisites: common programming languages, logics, automata theory
  - Not the easiest path for students
  - Jobs: system SW, critical embedded systems, R&D

Courses overview

- Recommended programming practices
- Crash dump analysis
- Embedded and real time system
- Operating systems
- Object and component systems
- Middleware
- Software development and monitoring tools
- Performance evaluation of computer system
- **Behavior models and verification**
- **Program analysis and code verification**

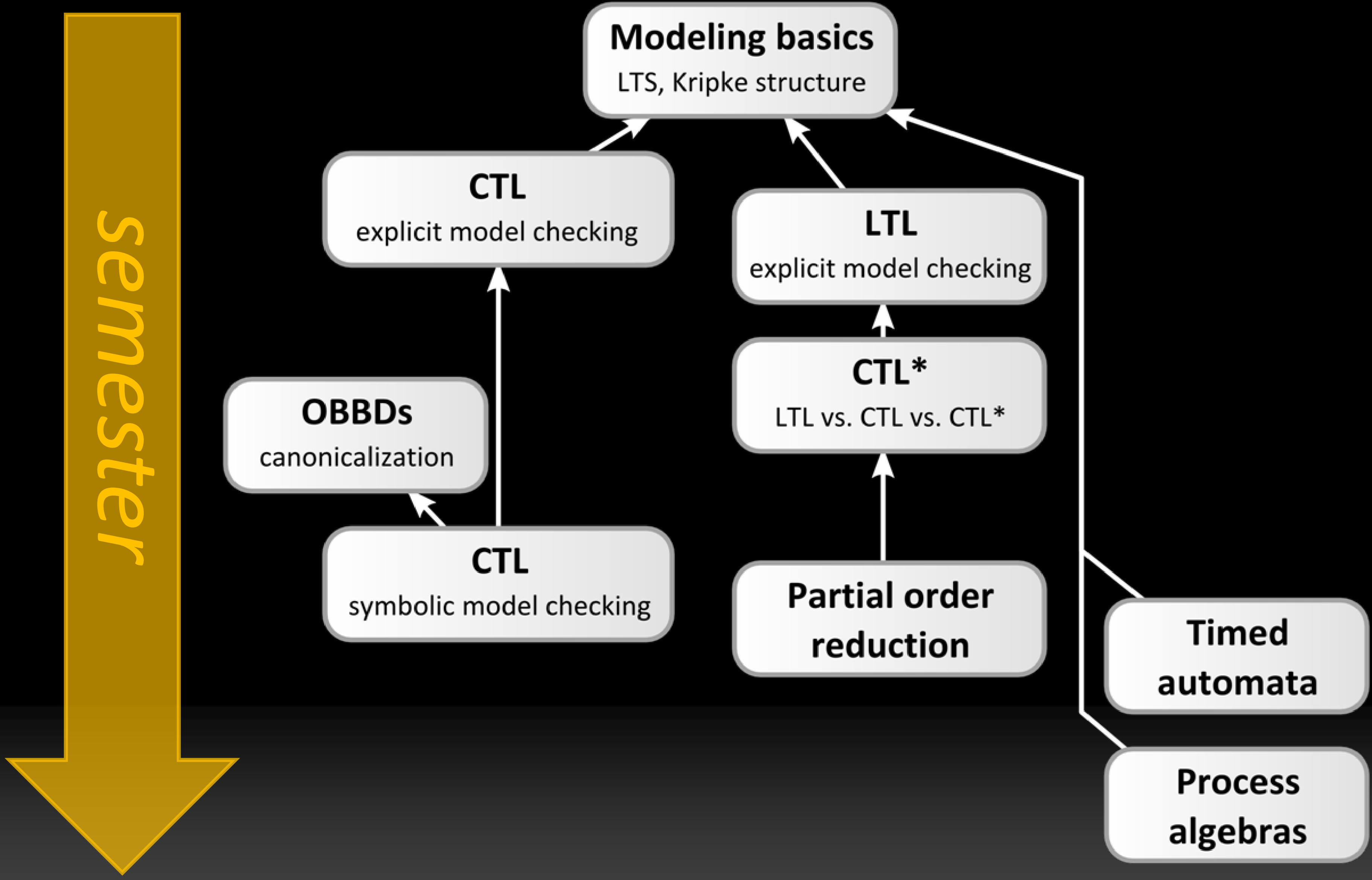
Behavior models and verification

- Basic course on formal methods
  - Model checking, behavior specification, temporal logics, basic principles and algorithms

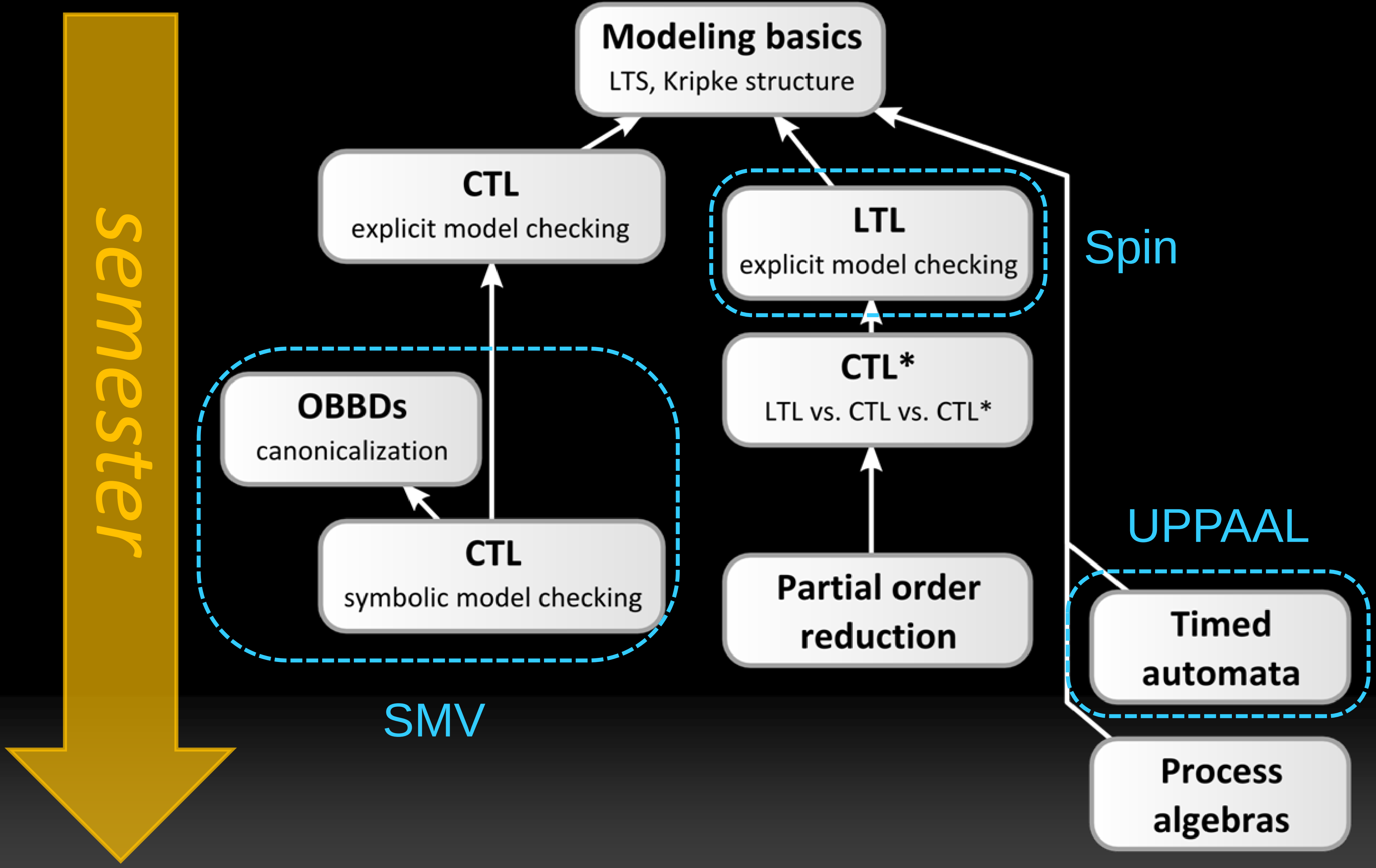
- **Lectures**
  - Modeling and verification of behavior
    - Both software and hardware

- **Labs**
  - Practical experience with Promela and Spin, (Nu)SMV, UPPAAL

Behavior models and verification



Behavior models and verification



Feedback

- Homework:

  “**model a railway station in Promela**”

  - Two groups of solutions
    - Simple models that can be verified, usually too abstract to reveal real problems
    - Too complex model exceeding the computational resources (time, memory)

- Hands-on experience with tools showed to be essential for comprehension of theory
  - All possible thread interleavings, LTL semantics, …

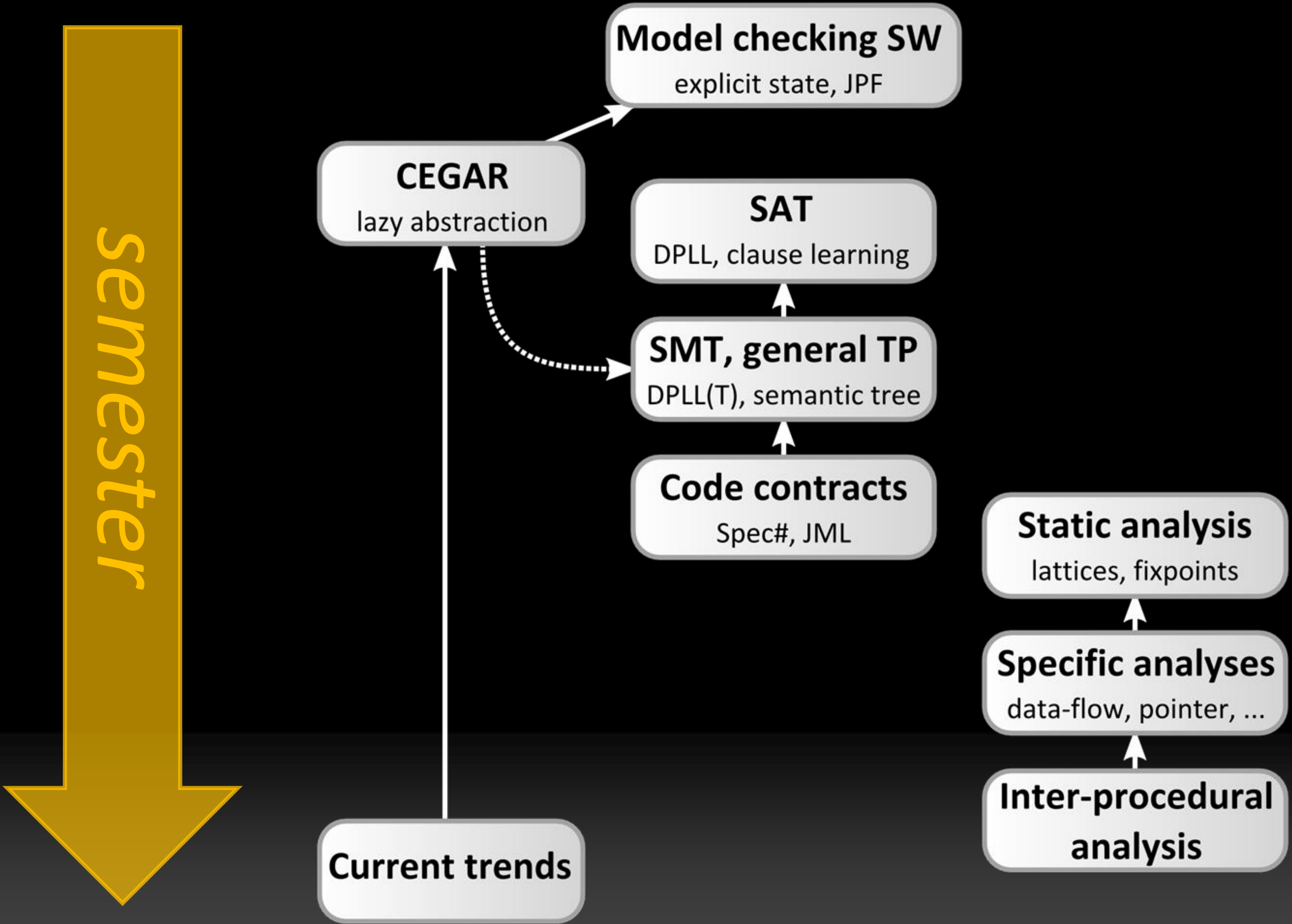
Program analysis and code verification

- Basic course on program verification
  - Program model checking
  - Deductive methods
  - Static analysis of code

- **Lectures**
  - Basic principles and algorithms

- **Labs**
  - Practicing the algorithms “by hand”
  - Experience with tools (JPF, PicoSAT, Soot)

Program analysis and code verification



semester

Model checking SW
explicit state, JPF

CEGAR
lazy abstraction

SAT
DPLL, clause learning

SMT, general TP
DPLL(T), semantic tree

Code contracts
Spec#, JML

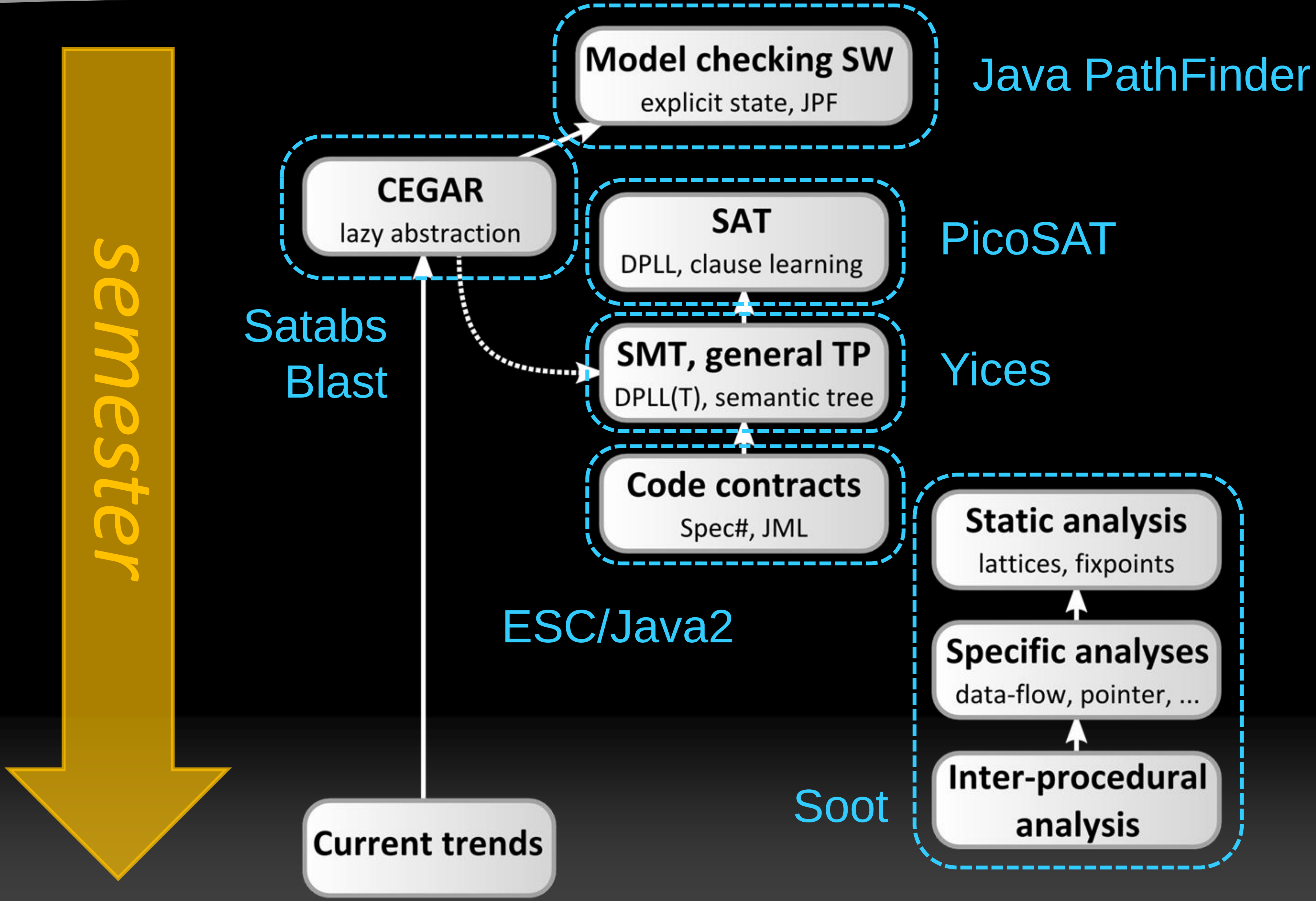
Static analysis
lattices, fixpoints

Specific analyses
data-flow, pointer, ...

Inter-procedural analysis

Current trends

Program analysis and code verification



semester

Model checking SW
explicit state, JPF

Java PathFinder

CEGAR
lazy abstraction

Satabs
Blast

SAT
DPLL, clause learning

PicoSAT

SMT, general TP
DPLL(T), semantic tree

Yices

Code contracts
Spec#, JML

ESC/Java2

Static analysis
lattices, fixpoints

Specific analyses
data-flow, pointer, …

Inter-procedural
analysis

Soot

Current trends

Feedback

- Practicing the algorithms “by hand”
  - Essential for comprehension
- Hands-on experience with tools
  - Students can see that the tools work (discover errors)

- Overview of many tools vs. deep insight into few tools
  - Each tool works good in some cases and not so good in others
  - Different means, goals, and application domains

- Challenges for students
  - Creating JML-like specifications (contracts)
    - Choosing the right level of abstraction and precision
  - No problems with properties expressed in the code (JPF)

Feedback

- **Missing textbooks** on program code analysis
  - Lectures based mostly on research papers
- Most **tools** are **not mature**
  - Cryptic user interface
  - Integration into IDE would be useful

Conclusion

- Low attendance – students may consider these courses
  - Difficult (compared to SE, DB, …)
  - Not practical for them

- Prospective students participate in our research projects
  - They get necessary background in the field

- Outlook
  - Keeping the courses up with state of the art
  - Making courses more appealing and accessible