CS5030

AUTOMATED PROGRAM VERIFICATION

WHO, WHERE AND WHEN

- Instructor : Kartik Nagar
- CS 24, Slot F
- All course material will be available on the course moodle page.

WHAT IS PROGRAM VERIFICATION?

Ensuring that a program does what it is supposed to do.

Testing is the most common strategy used to ‘verify’ programs, from novice to expert programmers.

DOES TESTING ACHIEVE PROGRAM VERIFICATION?

Ensuring that a program does what it is supposed to do

on some inputs. 

Testing is the most common strategy used to ‘verify’ programs, from novice to expert programmers.

DOES TESTING ACHIEVE PROGRAM VERIFICATION?

Ensuring that a program does **always** what it is supposed to do on all inputs.

Testing is the most common strategy used to ‘verify’ programs, from novice to expert programmers.

WHY IS VERIFICATION NEEDED

Bugs are rampant and can have catastrophic consequences

SOFTWARE CRASHES

THE INFAMOUS BLUE SCREEN

Paris highway

Metro station, Manhattan

Olympic Games, 2008

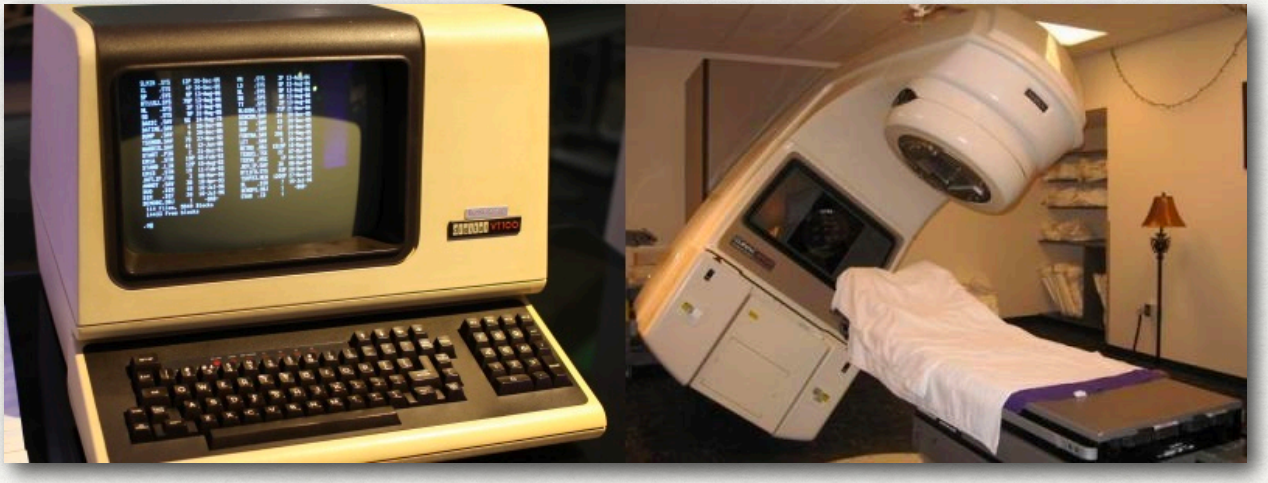
MOST EXPENSIVE BUG IN HISTORY

THE ARIANE 5 DISASTER [1996]

- On June 4, 1996, the Ariane 5 took a 90 degree flip and self-destructed in a gigantic explosion 37 seconds after its launch.
- The disaster cost $370 Million.
- Happened because of an integer overflow error!
- Widely acknowledged as the most expensive bug of all time.

SOFTWARE KILLS...

THERAC-25 KILLER BUG [1985-87]



- The Therac-25 was a radiation therapy machine used for treatment of cancer
- Between 1985 and 1987, the machine was the cause of six radiation-overdose accidents, resulting in 2 deaths.
- Happened due to buggy synchronization between the software and the radiation hardware resulting in a race condition.

MORE CATASTROPHIC BUGS IN HISTORY...

- Boeing 737 Max Bug [2019]. Estimated Loss of $9.2 Billion.
- The DAO Smart Contract Hack [2017]. Loss of 3.6 Million Ether ($50 Million).
- Mars Climate Orbiter Crash [1999]. Loss of $235 Million.
- ...

WHY IS VERIFICATION NEEDED

- Compilers can catch syntactic bugs, but what about semantic bugs?
- Bugs occur due to many reasons
  - Failure to understand the nuances of the programming language (e.g. integer overflow)
  - Failure to take into account the behaviour of the underlying system (especially relevant for concurrent, distributed, cyber-physical systems)
  - Changes in the requirements/updates in the system
  - Higher volume of software with multiple developers
  - The “Human” factor (carelessness, lack of attention, etc.)
  - …

WHERE IS VERIFICATION NEEDED

- There is a growing need for verified software in many areas
  - Aerospace, Avionics, Automobiles
  - Medical devices
  - Financial software
  - Operating Systems, Compilers, Software Libraries
  - Network Protocol Implementations
  - Any code ever written???

AUTOMATED VERIFICATION

"Push-button Verification"

- Correctness is generally specified in the form of a formal mathematical specification
- The specification should hold for all executions of the program
- The verification is always with respect to the specification

AUTOMATED VERIFICATION

AUTOMATED VERIFICATION

- Output array is sorted.
- Output array is permutation of input array.

QUESTIONS

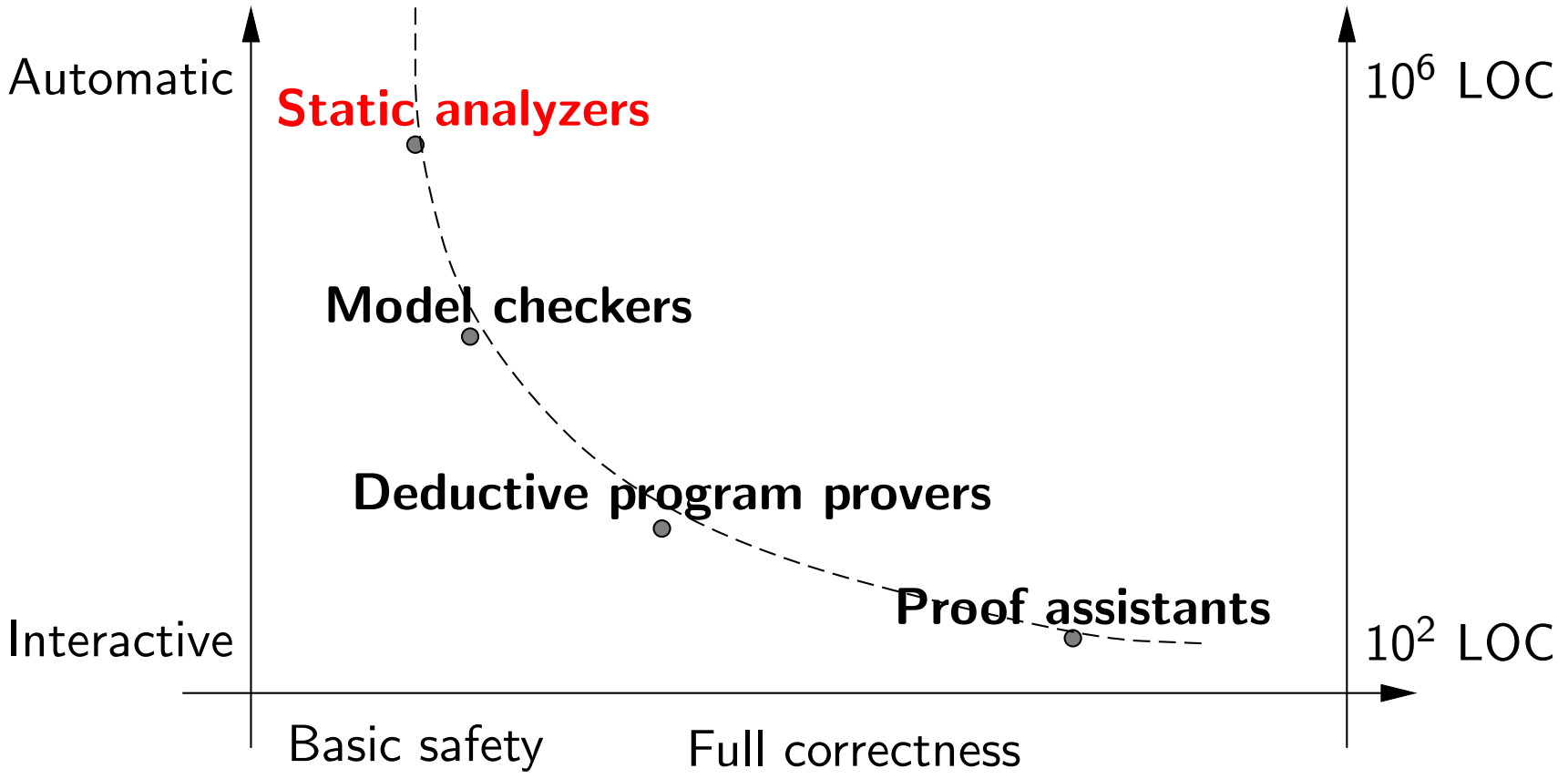
- Is it possible that a program is verified but it still has bugs?
- Is it possible that the verifier says the program is not correct but the program actually has no bugs?

Verification is always relative to the specification. There could be bugs in the specification itself!

UNDECIDABILITY OF VERIFICATION

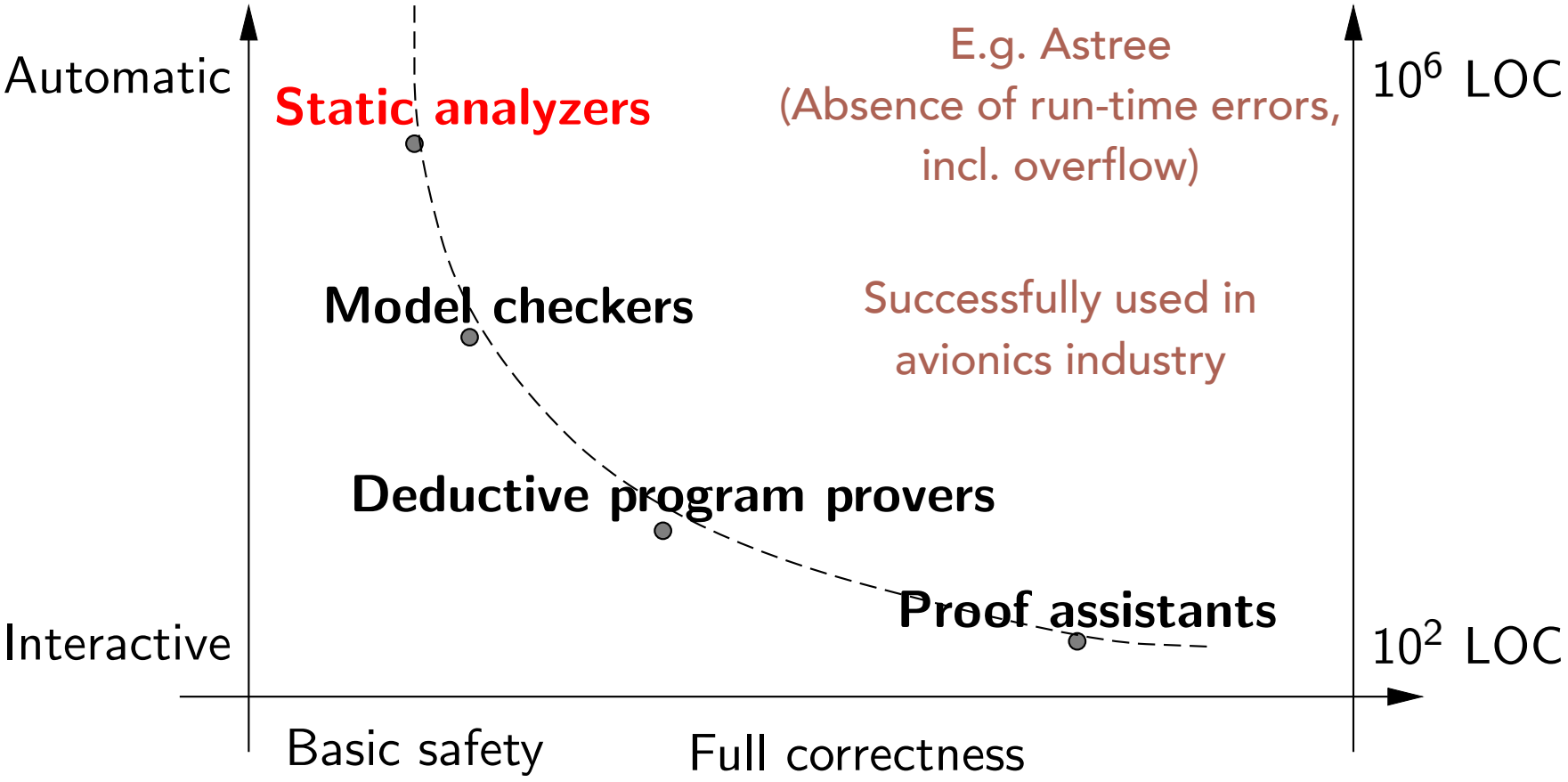
- Automated verification of programs written in a Turing-complete language is undecidable.
  - Rice's Theorem: There is no Turing machine that can decide whether the language accepted by a given Turing machine has a non-trivial property.
- In practice, automated verifiers either restrict the space of programs, or the space of specifications.
  - Even then, verification is computationally expensive.

A panorama of verification tools



Ref: Xavier Leroy, "In search of software perfection"

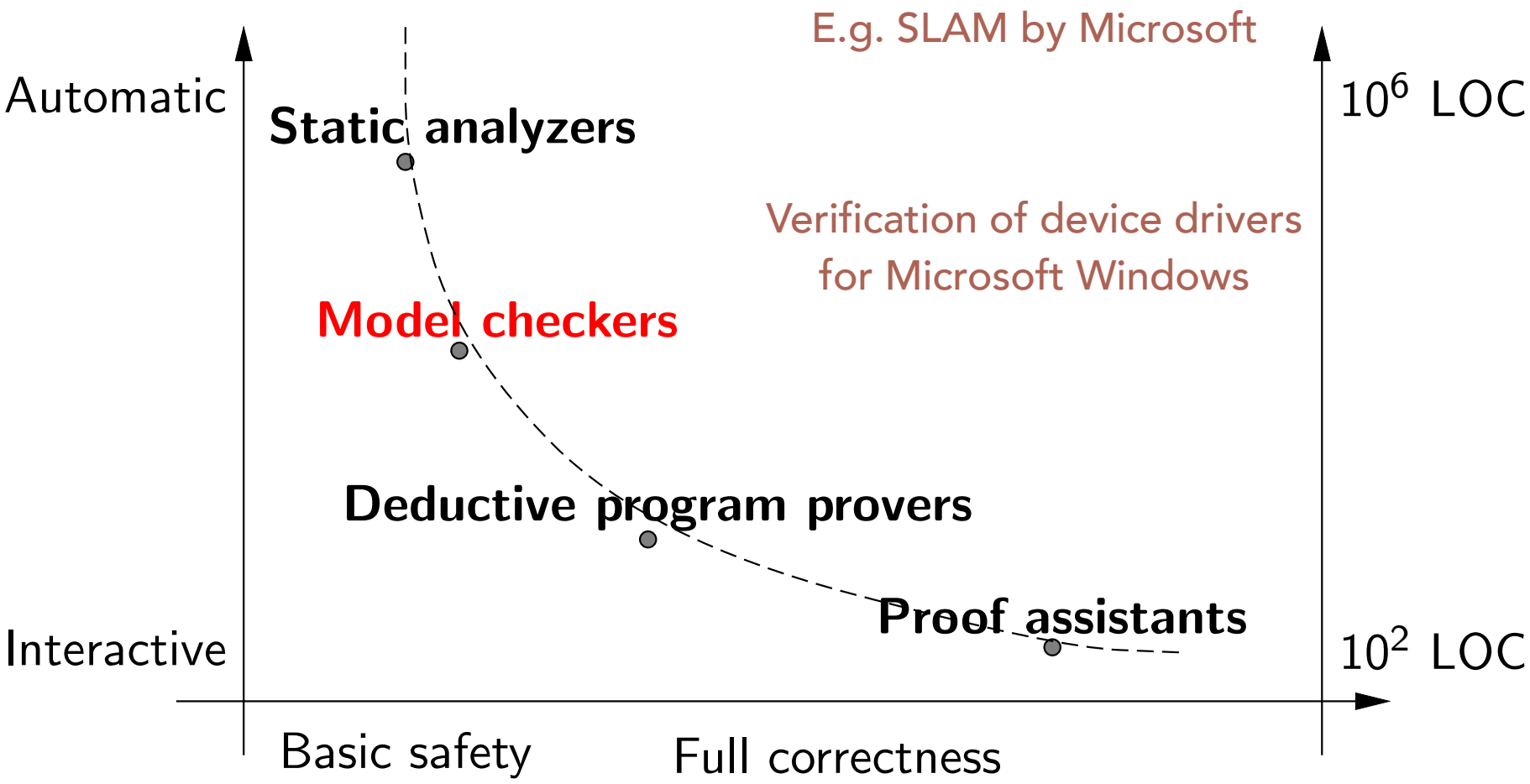
A panorama of verification tools



Static analysis: automatically infer simple properties of one variable ($x \in [N_1, N_2]$, x mod $N = 0$, etc) or several ($x + y \leq z$).

Ref: Xavier Leroy, "In search of software perfection"

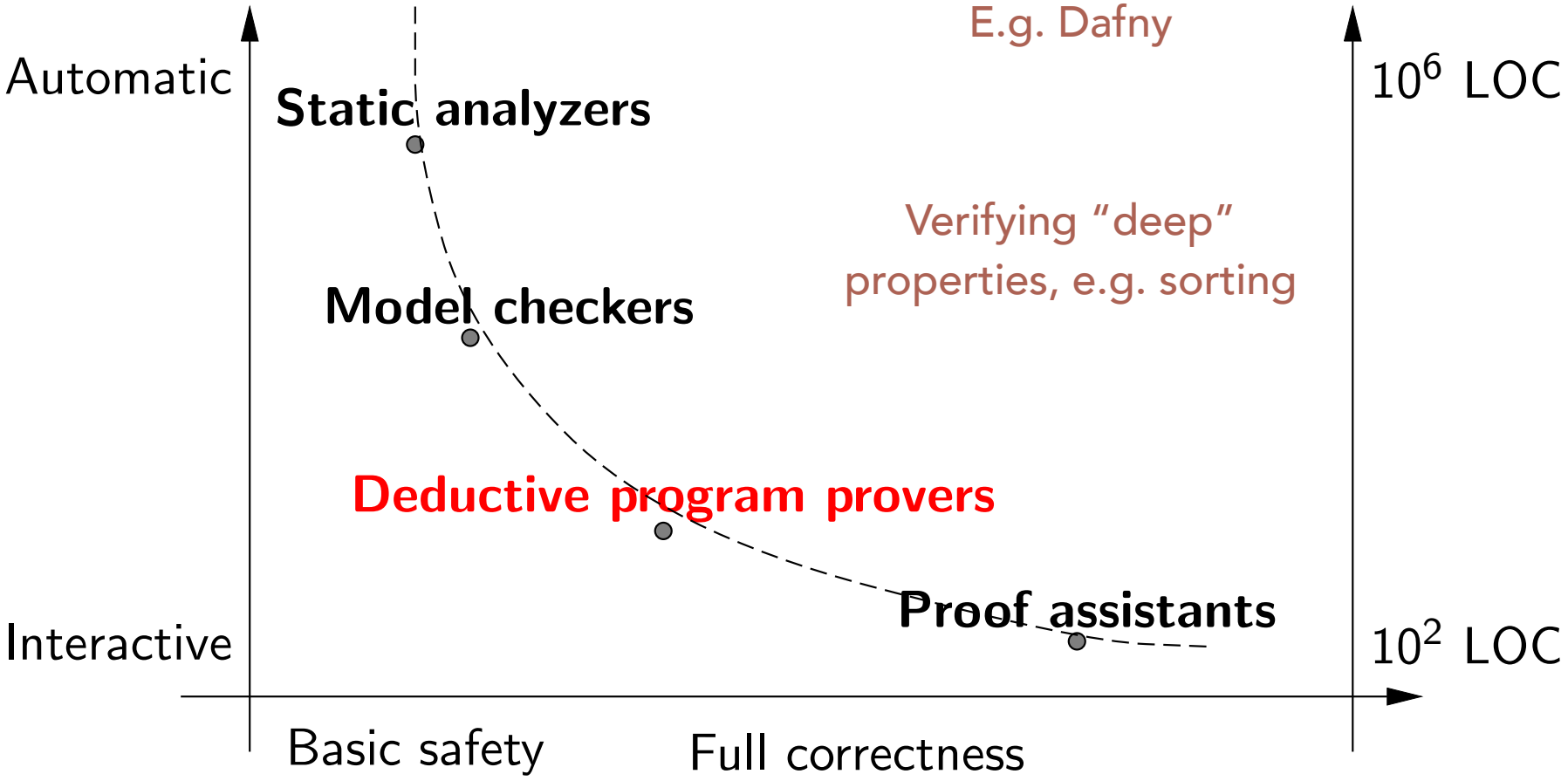
A panorama of verification tools



Model checking: automatically check that some "bad" program points are not reachable.

Ref: Xavier Leroy, "In search of software perfection"

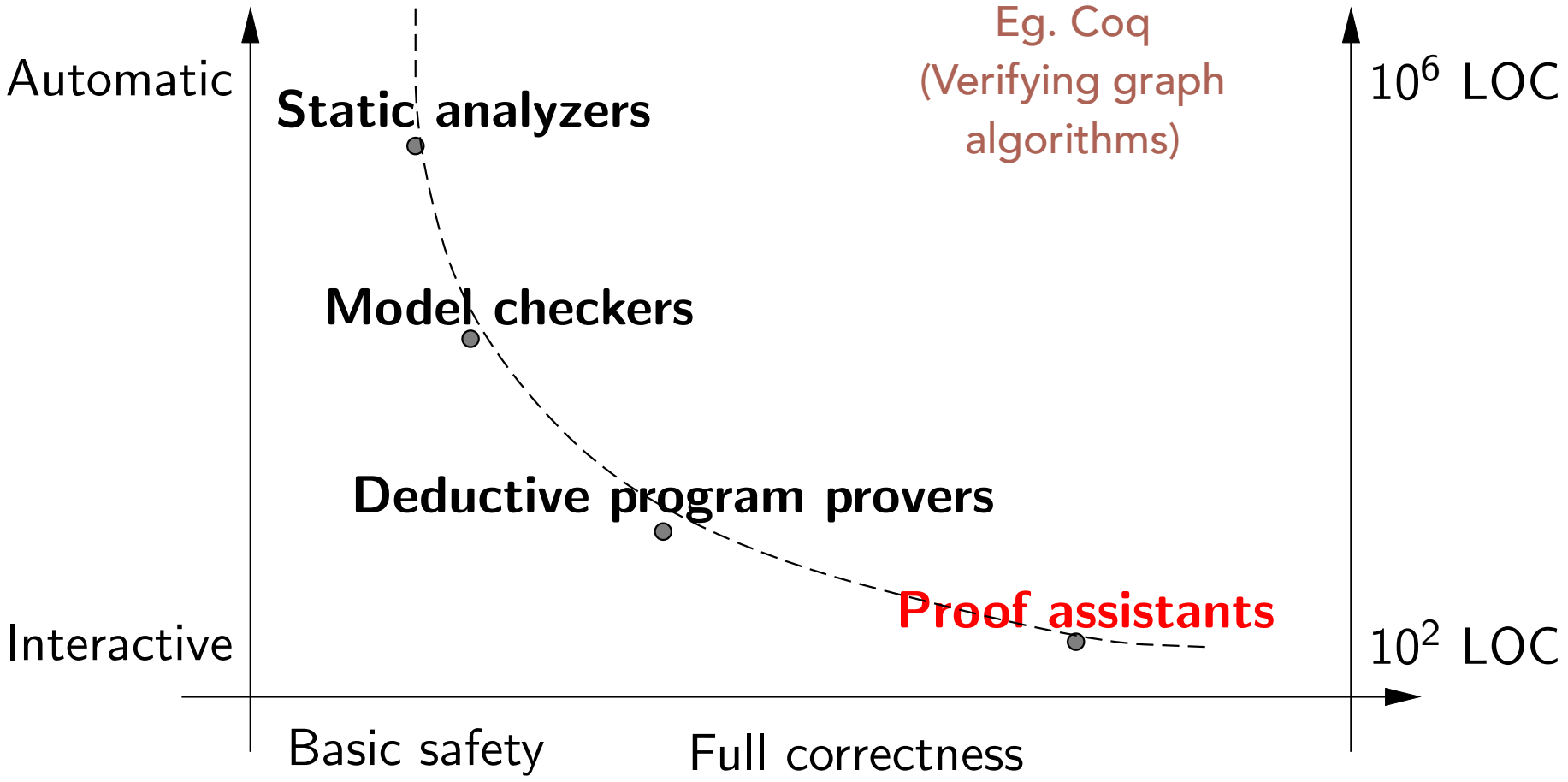
A panorama of verification tools



Program proof: show that
preconditions $\Rightarrow$ *invariants* $\Rightarrow$ *postconditions*
using automated theorem provers.

Ref: Xavier Leroy, "In search of software perfection"

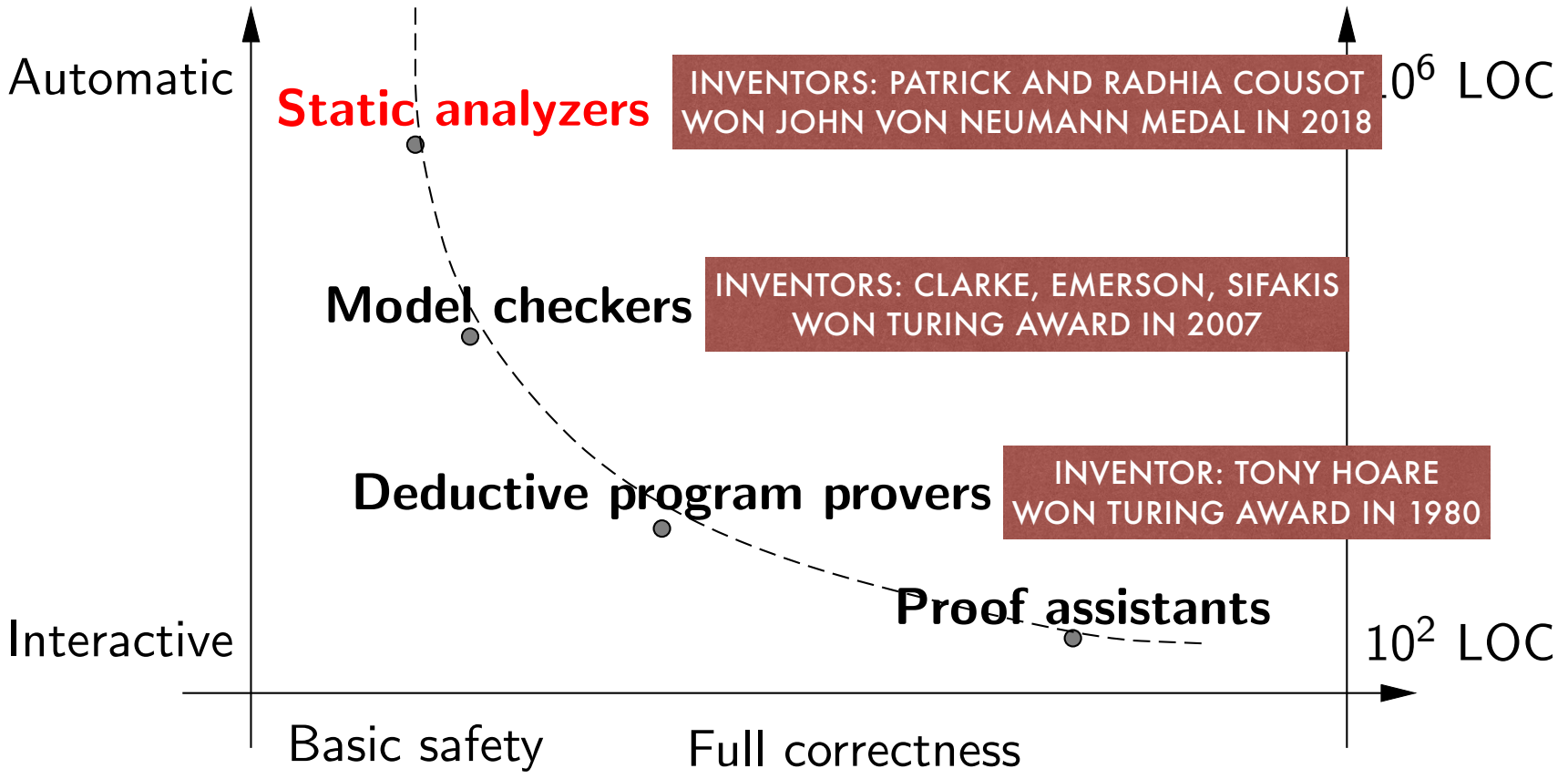
A panorama of verification tools



Proof assistants: conduct mathematical proofs in interaction with the user; re-check the proofs for correctness.

Ref: Xavier Leroy, "In search of software perfection"

A panorama of ~~verification tools~~ awards



Ref: Xavier Leroy, "In search of software perfection"

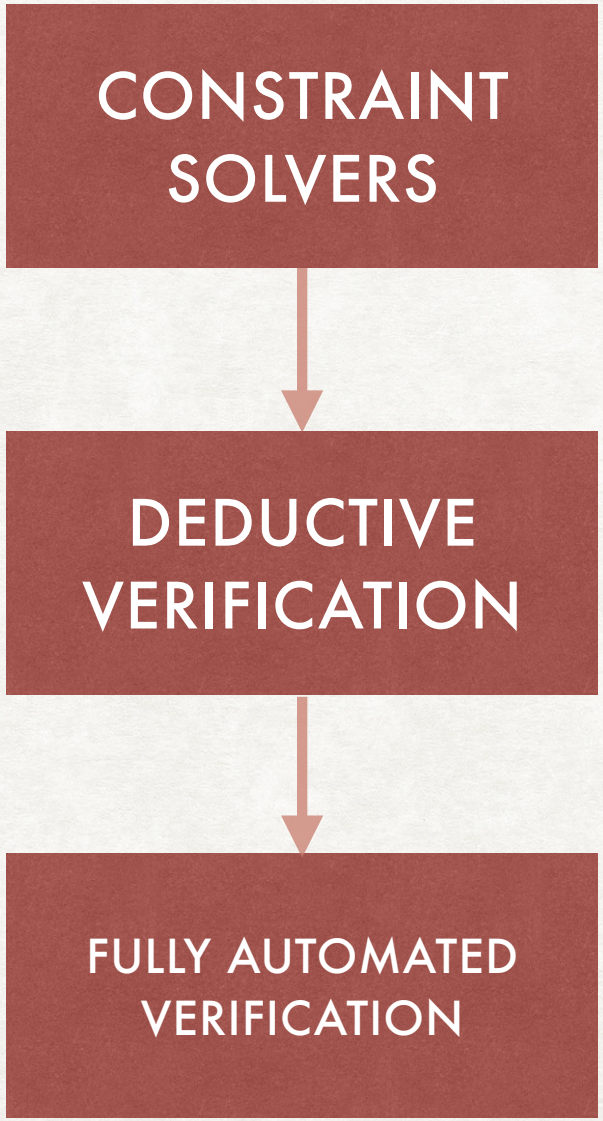
IN THIS COURSE...

- In the first part of the course, we will focus on **Deductive Verification using Constraint Solvers**.
  - Semi-automated approach, requires loop invariants from the programmer.
- In the second part, we will look at more advanced fully automated approaches to verification.
- Note that we will just scratch the surface in the area of verification
  - Topics **not** covered: Proof assistants, Verification for concurrent and distributed systems (a vast area, suitable for another course), Verification for Heap-manipulating programs, Program Synthesis, Verification of Hybrid/Cyber-physical systems,...

COURSE LOGISTICS

- Grading Policy (tentative)
  - Paper reading - 25%
  - Class participation - 5%
  - Assignments (3 Theory + 2 Tool) - 40%
  - End sem - 30%
- Textbook
  - The Calculus of Computation: Decision Procedures with Applications to Verification. Aaron R. Bradley and Zohar Manna.
  - Chapters will be uploaded to Moodle.

COURSE STRUCTURE



- Propositional Logic, SAT solving, DPLL
- First-Order Logic, SMT
- First-Order Theories

- Operational Semantics
- Strongest Post-condition, Weakest Pre-condition
- Hoare Logic

- Abstract Interpretation/Static Analysis
- Model Checking, Predicate Abstraction, CEGAR
- Property-directed Reachability