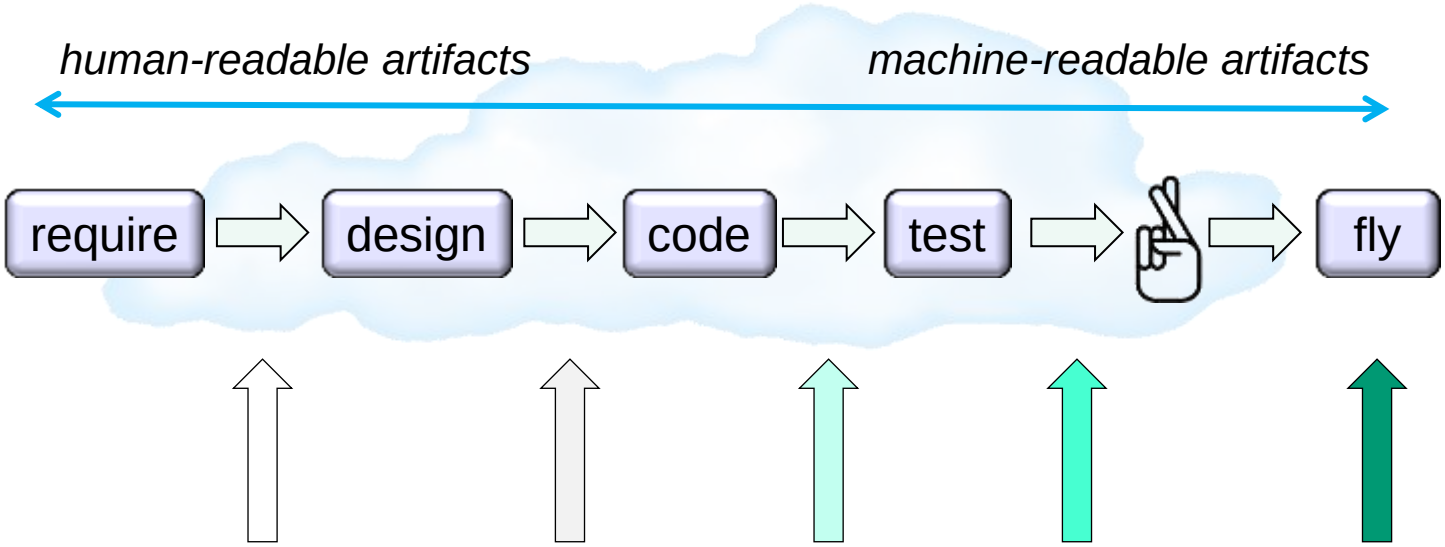
Agile Verification

Gerard J. Holzmann
gh@jpl.nasa.gov

how we design & test software today

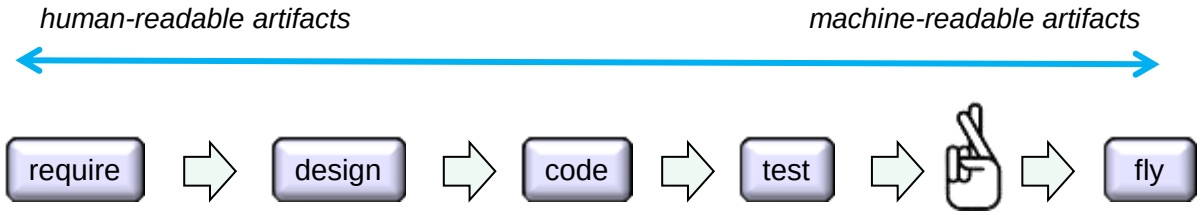
(a simplification)



the main quality “gates”

gaining in strength and precision as we move to the left

the standard *formal methods* pitch



- *model-based* design techniques can introduce machine readable artifacts earlier in the life-cycle
  - which enable more thorough *tool-based analysis* techniques for requirements & design
    - for instance, *logic model checkers* can then be used to verify requirements against high-level design models

another look at testing

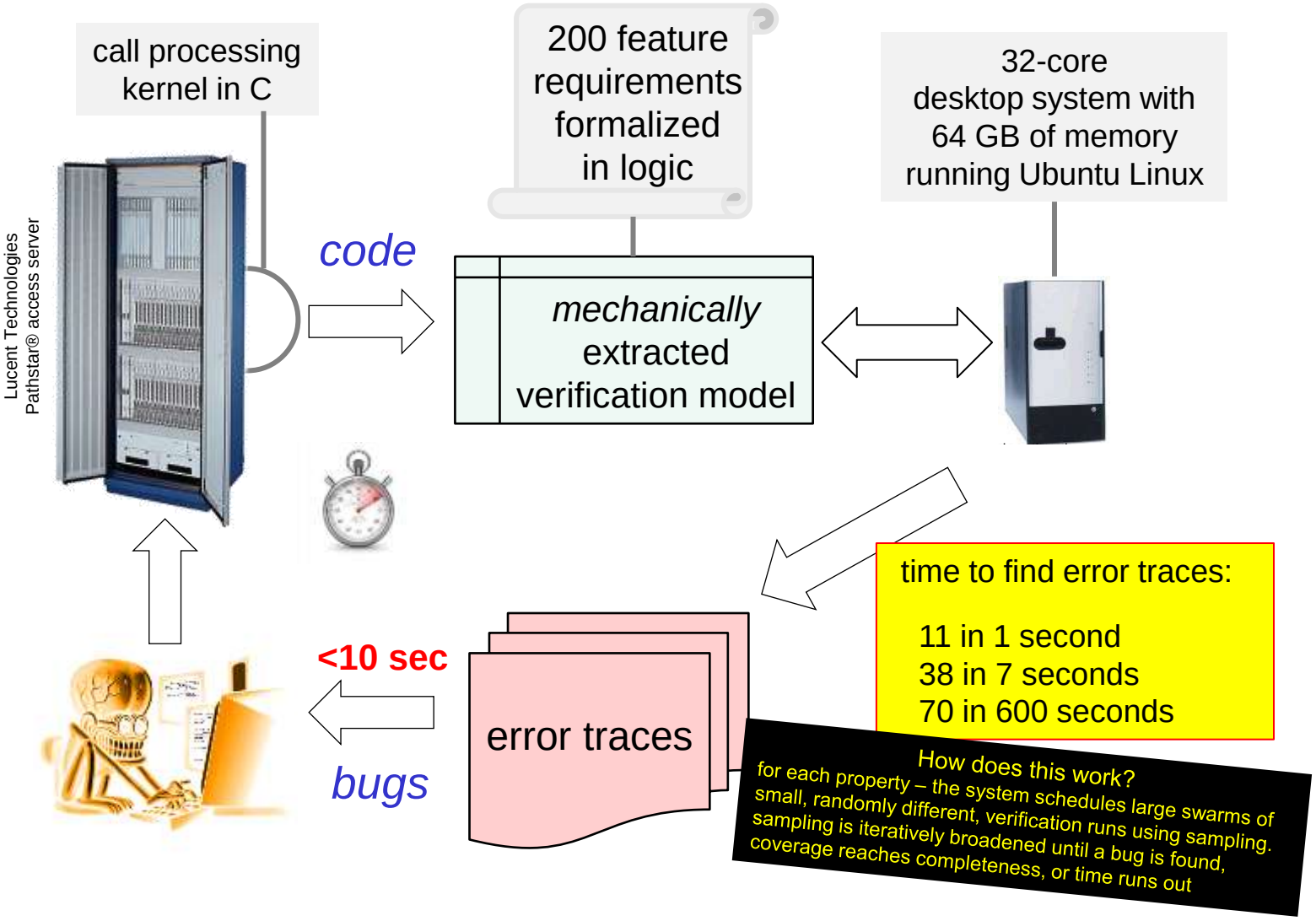
- in test-based verification, we tend to treat all code alike
  - but there's a difference between:
    - *deterministic* (e.g., math) routines and
    - *non-deterministic* (e.g., reactive) code

	Current Method	Better	Best
math routines (deterministic)	 sampling-based testing	randomized (fuzz) testing + static analysis	Pre- & Post-conditions, loop invariants, theorem proving
reactive code (concurrent)		?	logic model checking

these methods are very useful but none are "logically complete" and some are not "logically sound"

needs to be fast, automatic, and scalable

example of an agile verification process



synopsis

	Current Method	Better	Best
math routines (deterministic)	sampling-based testing	randomized (fuzz) testing + static analysis	Pre- & Post-conditions, loop invariants, theorem proving
reactive code (concurrent)		?	logic model checking

Agile Verification techniques can fill a gap we have in developing reliable mission-critical software

With the large multi-core systems that are now generally available, this approach has become technically feasible