Combining Mechanized Proofs and Model-Based Testing in the Formal Analysis of a Hypervisor

Hanno Becker Juan Manuel Crespo Jacek Galowicz Ulrich Hensel
Yoichi Hirai César Kunz Keiko Nakata Jorge Luis Sacchini
Hendrik Tews Thomas Tuerk

Formal Methods group of an unnamed company

November 11th, 2016

The company that must not be named

2012 decision to invest in microkernel based virtualization and formal methods for a disruptive technology change

2013 hiring started in the *microkernel valley* in Dresden, Germany

2014 office opening and press releases

2015 $\approx$ 25 employees in Dresden, including 8 PhD's on formal methods

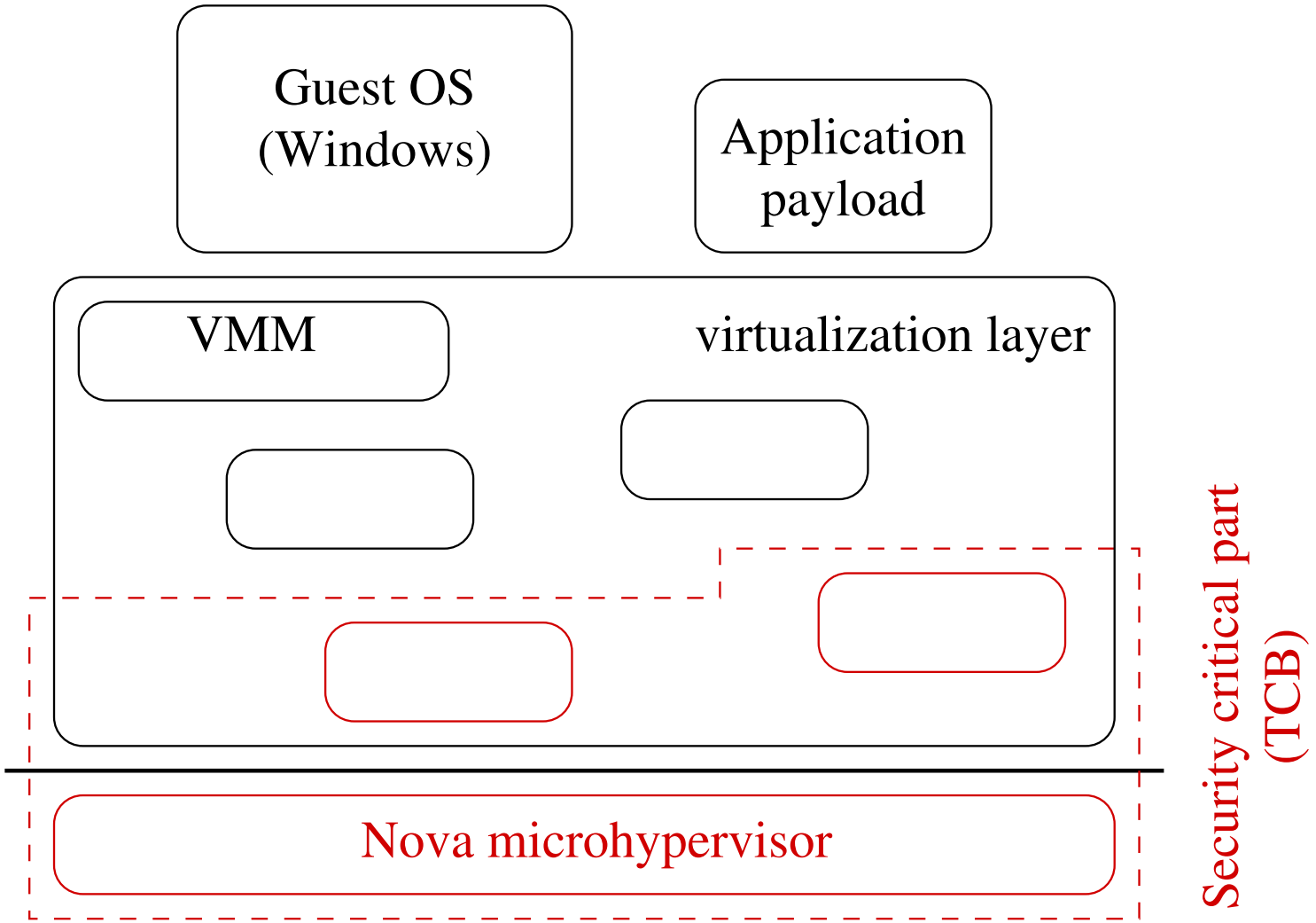
2016 confidence in disruptive technology diminished in sync with the fall of stocks

2016 office shutdown on August 15 without prior notice (no press release)

My role there

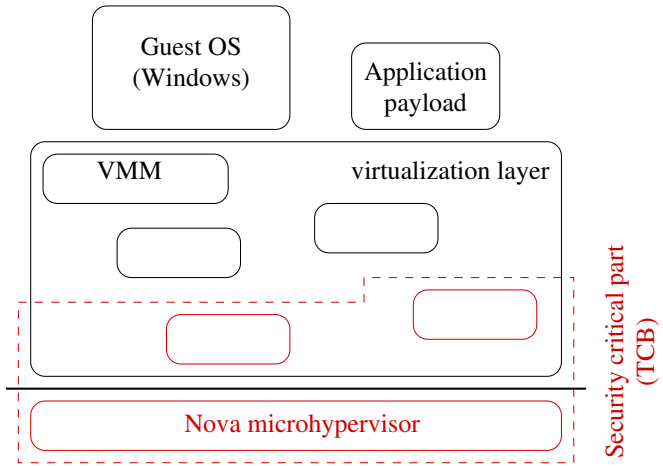
- office manager
- principal formal methods architect

System Architecture



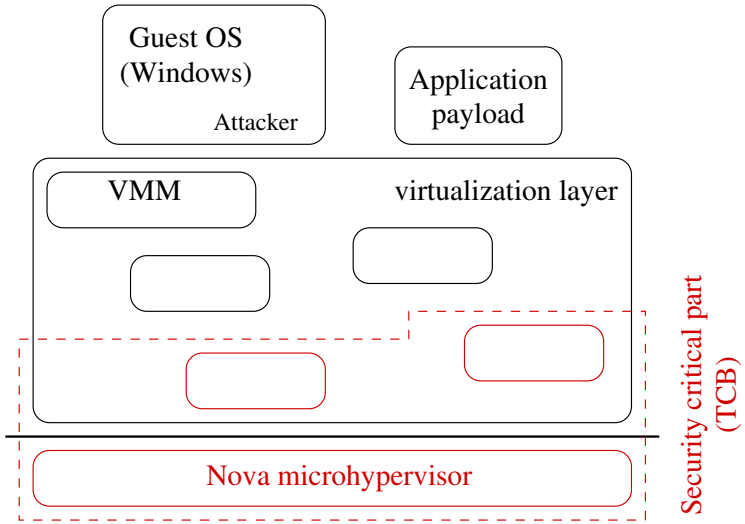
Project Goals

- microkernel based virtualization
- formally verified security guarantees for the TCB
- for security aware industry leaders
- they hopefully set a trend for everybody
- guarantees might be a competitive advantage



- about 20 Kloc C++ in TCB
- partial verification results only for first releases (e.g., incomplete refinement chain)
- formal verification focused on microkernel only

Our Vision was



Guest Attacker Security

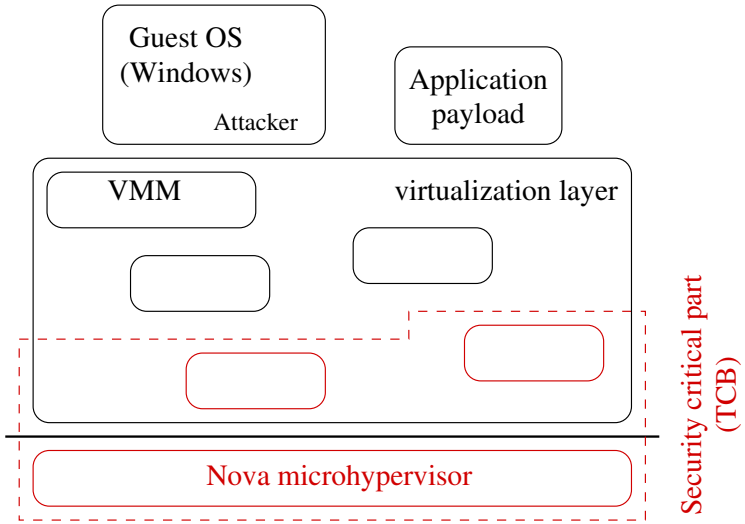
An attacker present inside the guest can neither

- directly modify the memory outside the guest,
- nor change the behaviour of any component outside the guest.

Formally proved for the source code of the TCB.

Started verification 2014 with working on Nova.

Our Vision was



Guest Attacker Security

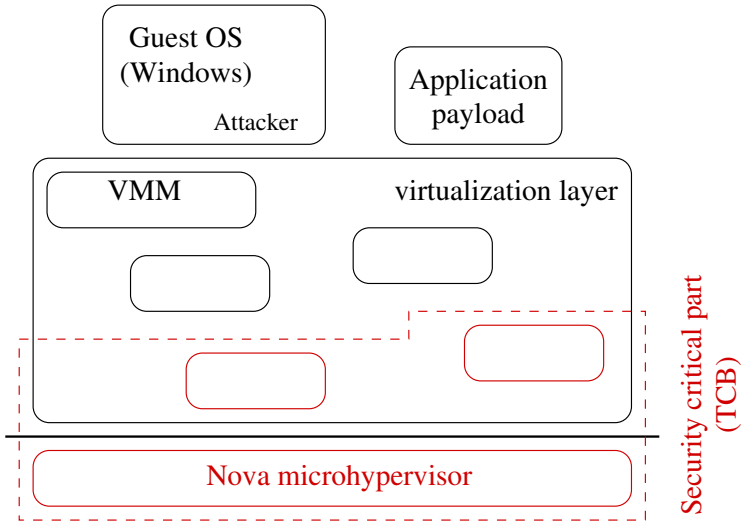
An attacker present inside the guest can neither

- directly modify the memory outside the guest,
- nor change the behaviour of any component outside the guest.

Formally proved for the source code of the TCB.

Started verification 2014 with working on Nova.

Our Vision was



Guest Attacker Security

An attacker present inside the guest can neither

- directly modify the memory outside the guest,
- nor change the behaviour of any component outside the guest.

Formally proved for the source code of the TCB.

Started verification 2014 with working on Nova.

Process Challenges

Industrial Software Development with Formal Verification

- development and formal verification in parallel
- development driven by feature requests and performance concerns (i.e. *not* by ease of formal verification)
- reprioritization, plan changes
- verification of a moving target
- release planning independent of formal verification results
- C++ 11 expert level sources

Verification Process Requirements

- provide partial results early on
- partial results must have significance for potential customers
- 6-9 month milestones, again with meaningful results

Process Challenges

Industrial Software Development with Formal Verification

- development and formal verification in parallel
- development driven by feature requests and performance concerns (i.e. *not* by ease of formal verification)
- reprioritization, plan changes
- verification of a moving target
- release planning independent of formal verification results
- C++ 11 expert level sources

Verification Process Requirements

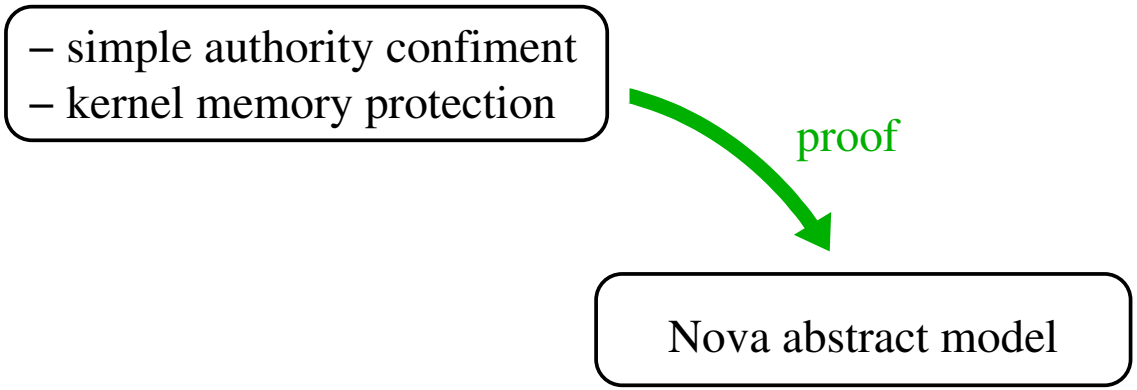
- provide partial results early on
- partial results must have significance for potential customers
- 6-9 month milestones, again with meaningful results

Our Approach

Nova abstract model

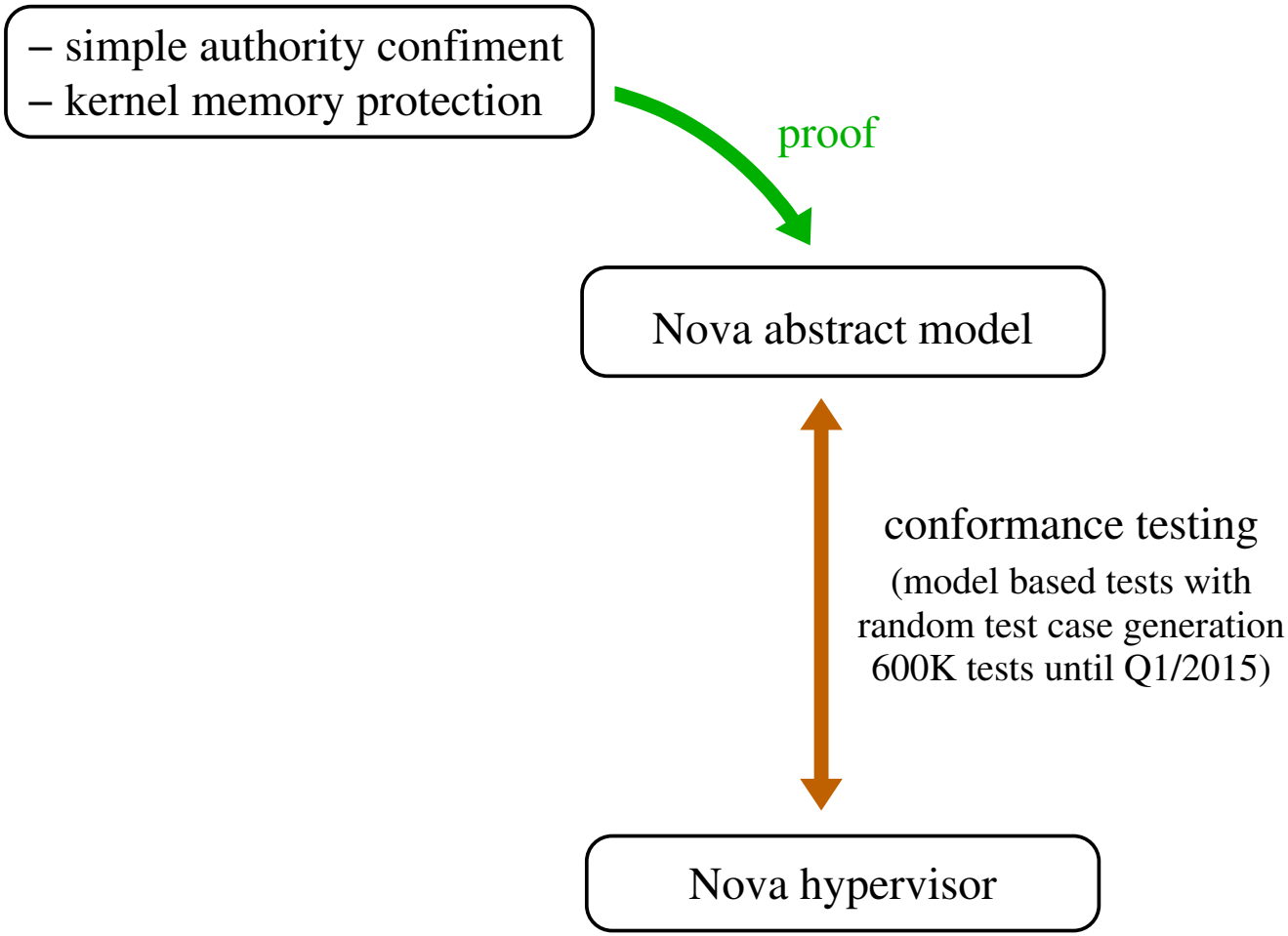
Nova hypervisor

Our Approach



Nova hypervisor

Our Approach



Our Approach (cont.)

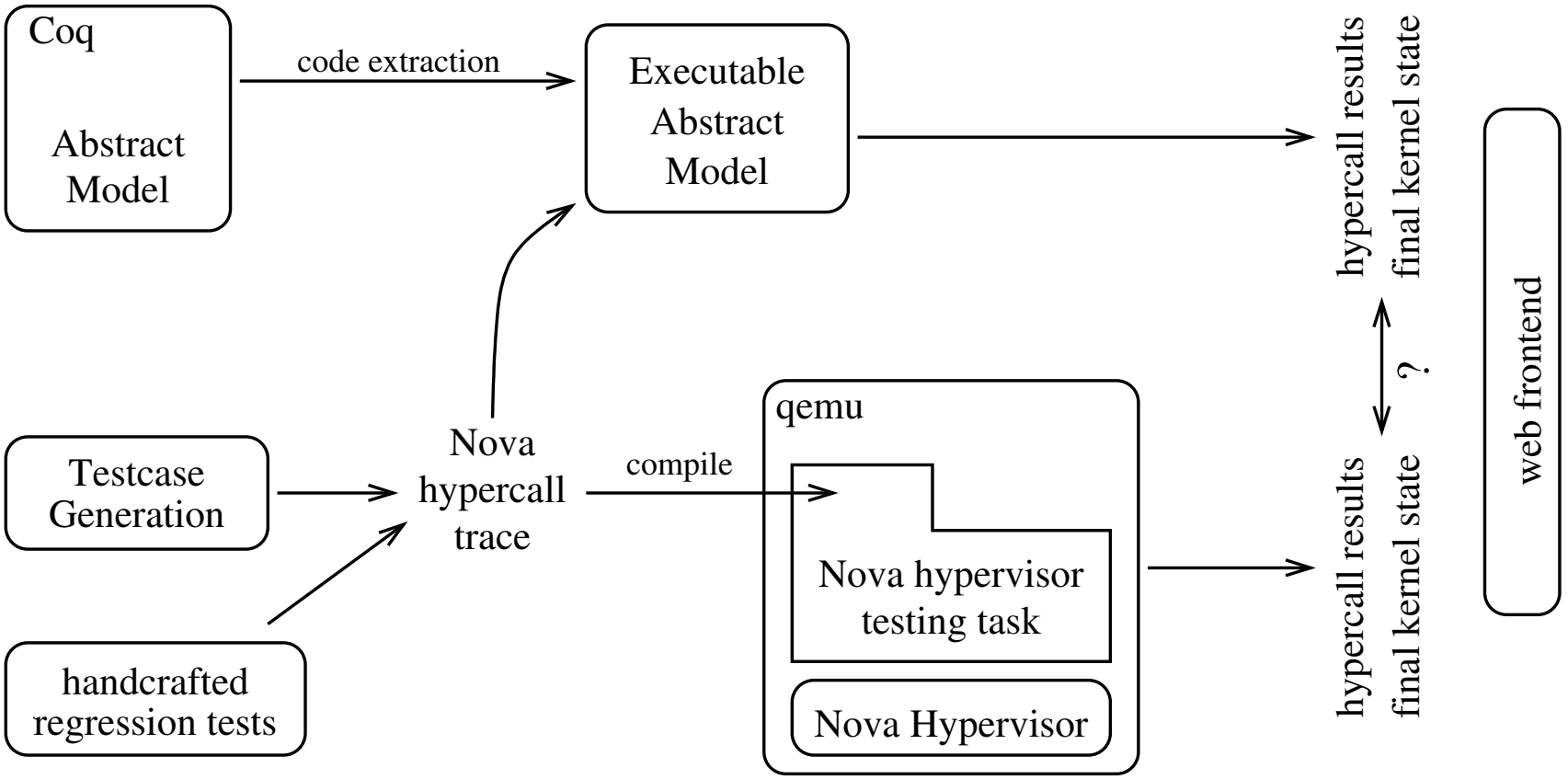
Restrictions

- sequential model, sequential conformance testing
- conformance testing on modified hypervisor

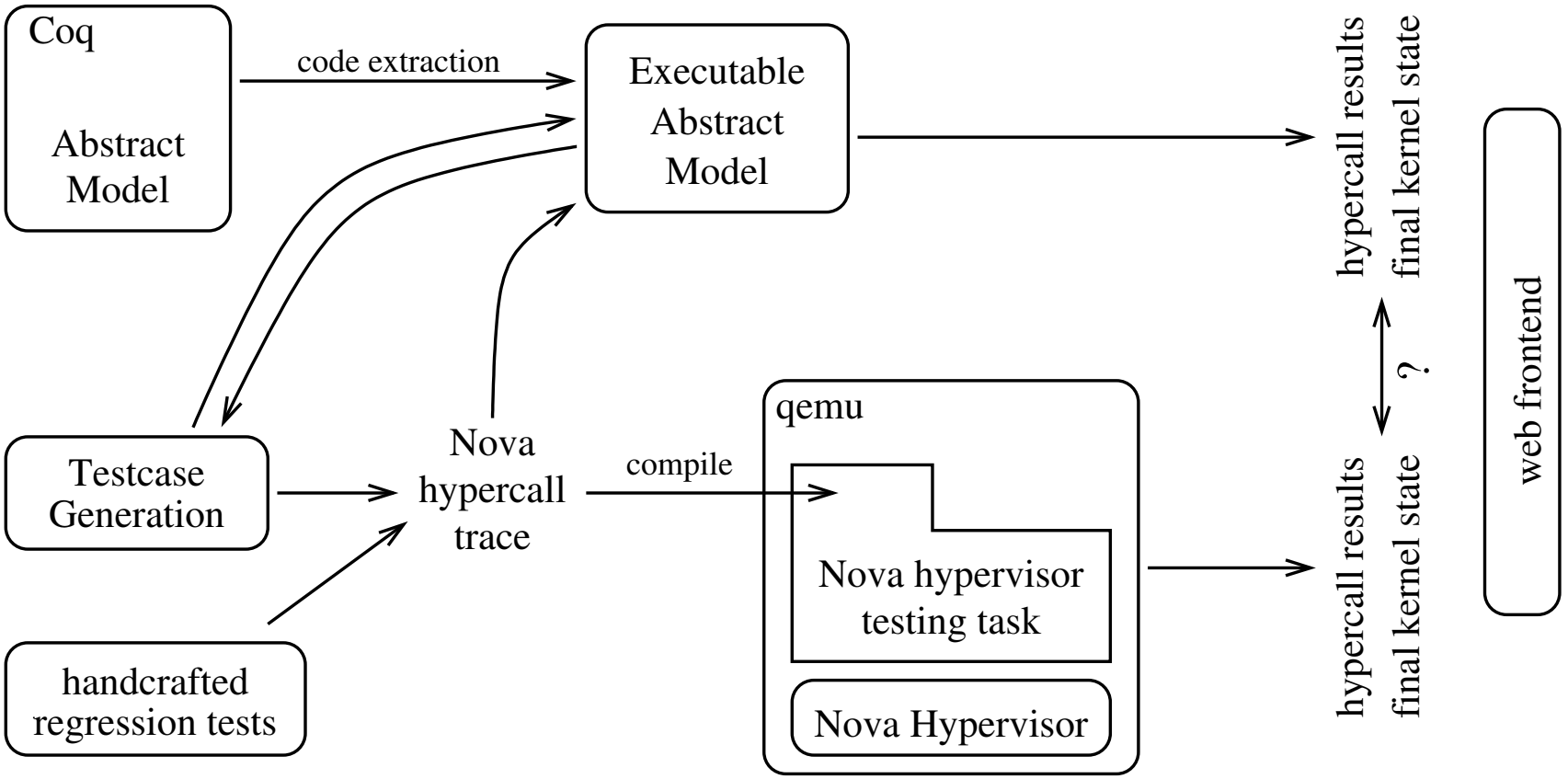
Benefits

- testing catches errors outside the scope of the model
- Independent from C++ sources
- first verification results after 9 month (27 person month)
- some flexibility on feature changes (proofs can be postponed)

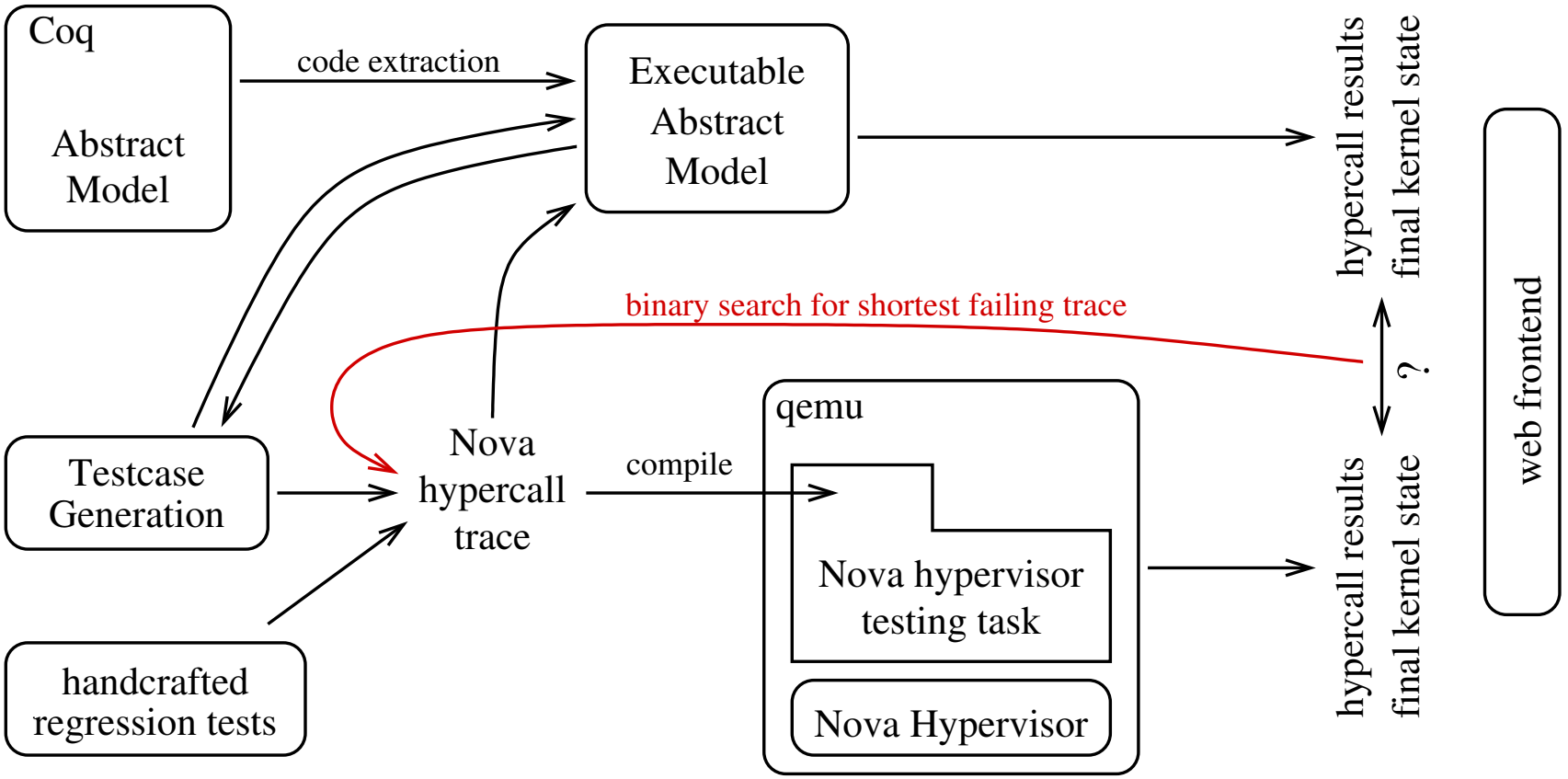
Conformance Testing Framework



Conformance Testing Framework



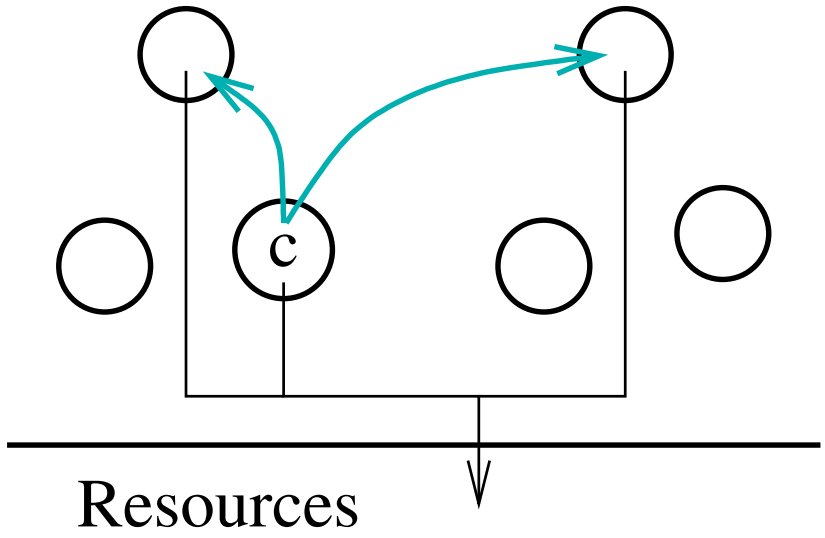
Conformance Testing Framework



Security Property: Authority Confinement

Consider

- a capability c, providing access rights to some ressource (memory, device, ...)
- a partitioning of the processes into two sets: *trusted* and *untrusted*
- an arbitrary execution $s \rightsquigarrow q$



If

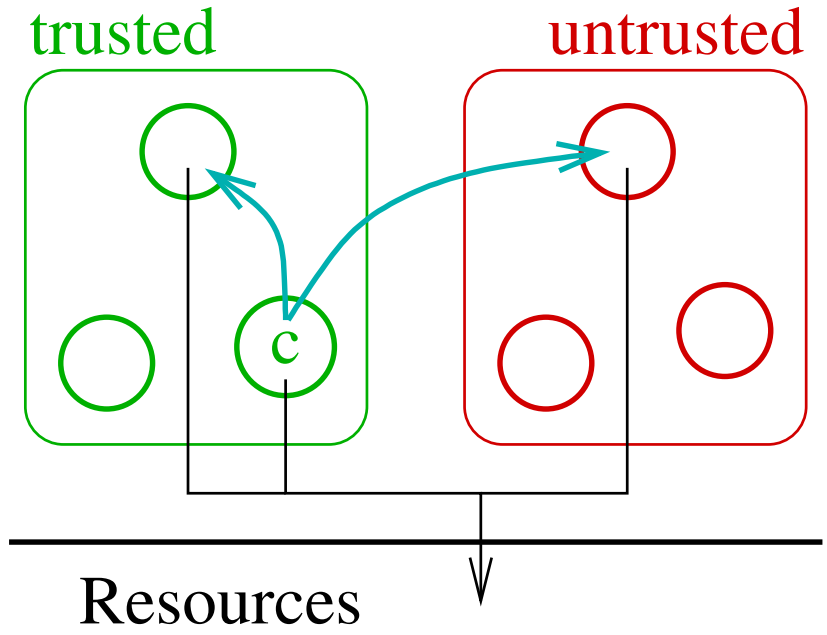
- the untrusted processes cannot access c in state s
- the untrusted processes do not create c
- no trusted process delegates c to an untrusted one

then the untrusted processes cannot access c in state q.

Security Property: Authority Confinement

Consider

- a capability c, providing access rights to some ressource (memory, device, ...)
- a partitioning of the processes into two sets: *trusted* and *untrusted*
- an arbitrary execution $s \rightsquigarrow q$



If

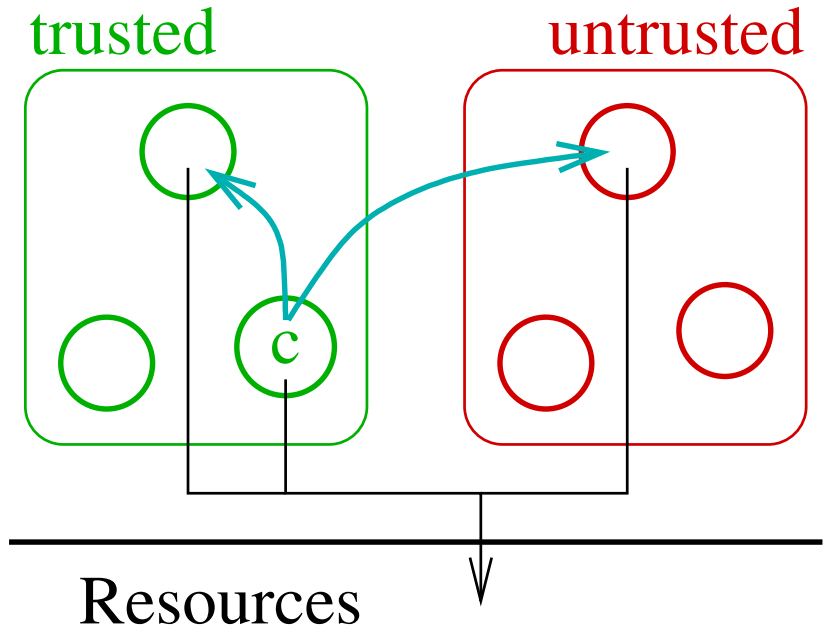
- the untrusted processes cannot access c in state s
- the untrusted processes do not create c
- no trusted process delegates c to an untrusted one

then the untrusted processes cannot access c in state q.

Security Property: Authority Confinement

Consider

- a capability c, providing access rights to some ressource (memory, device, ...)
- a partitioning of the processes into two sets: *trusted* and *untrusted*
- an arbitrary execution $s \rightsquigarrow q$



If

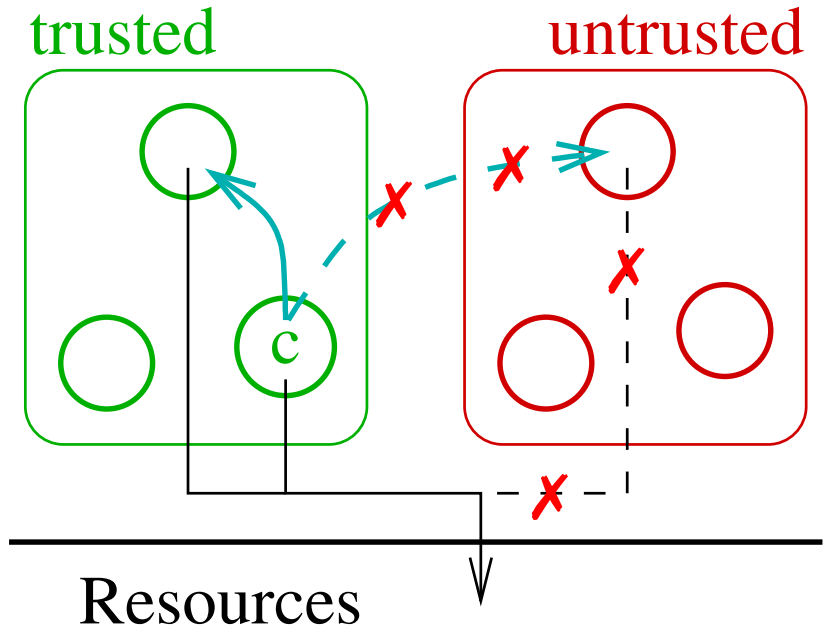
- the untrusted processes cannot access c in state s
- the untrusted processes do not create c
- no trusted process delegates c to an untrusted one

then the untrusted processes cannot access c in state q.

Security Property: Authority Confinement

Consider

- a capability c, providing access rights to some ressource (memory, device, ...)
- a partitioning of the processes into two sets: *trusted* and *untrusted*
- an arbitrary execution $s \rightsquigarrow q$



If

- the untrusted processes cannot access c in state s
- the untrusted processes do not create c
- no trusted process delegates c to an untrusted one

then the untrusted processes cannot access c in state q.

Security Property: Kernel Memory Safety

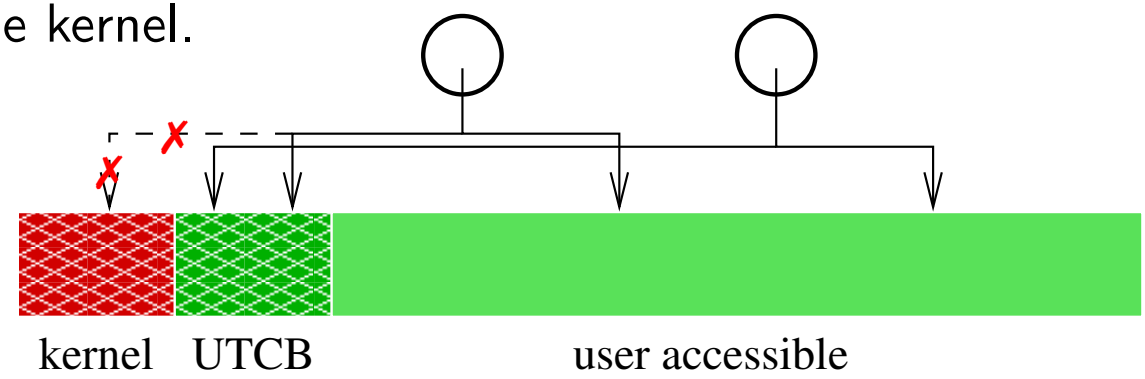
Note

User thread control blocks (UTCB) are allocated in the kernel but accessible in deprivileged processes.

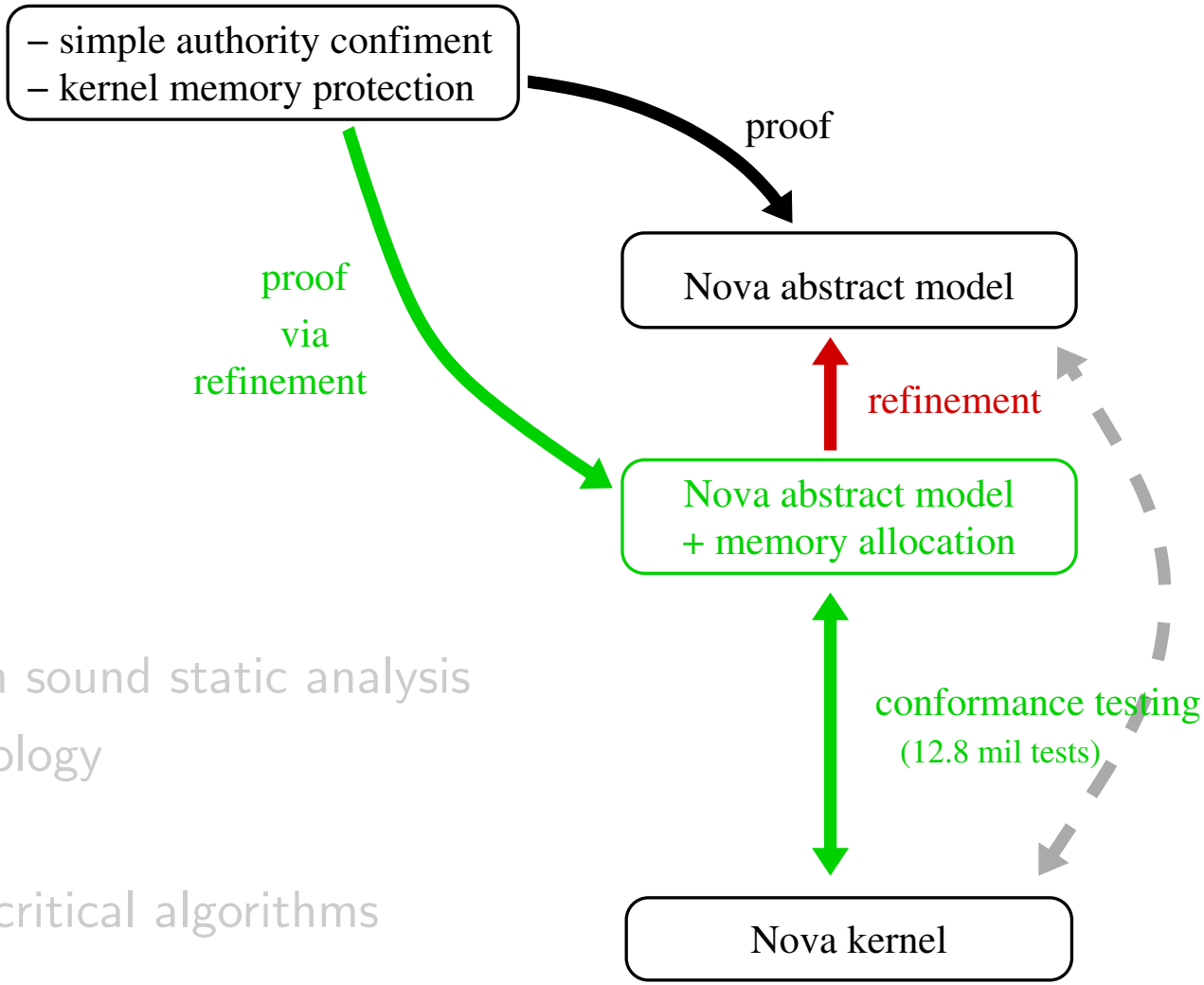
Kernel Memory Safety

In all reachable system states, all memory capabilities of all (deprivileged) processes point either to

- a UTCB, or
- to memory outside the kernel.



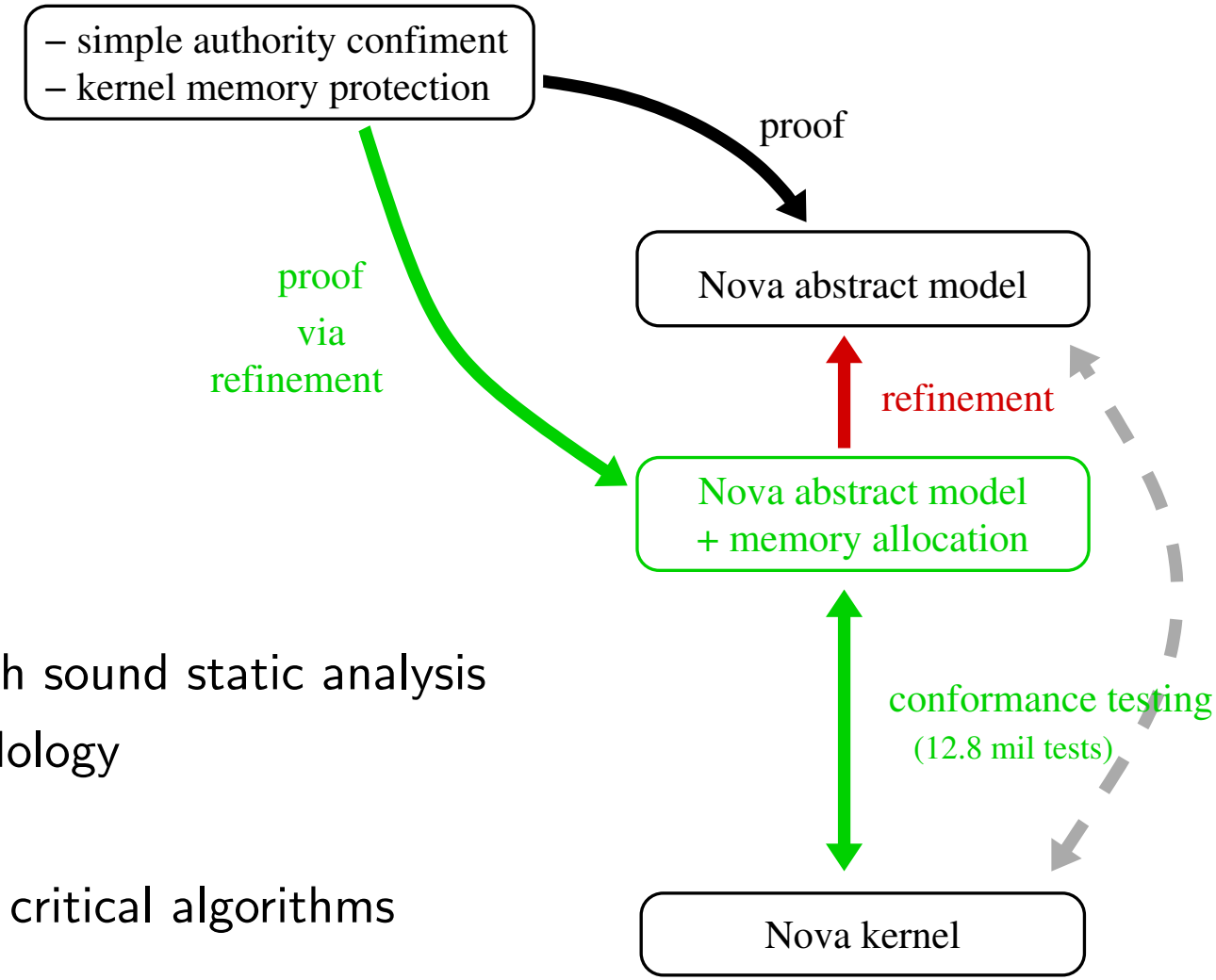
Work after Q1/2015



Additionally

- experiments with sound static analysis
- coverity methodology
- fuzzing
- model checking critical algorithms

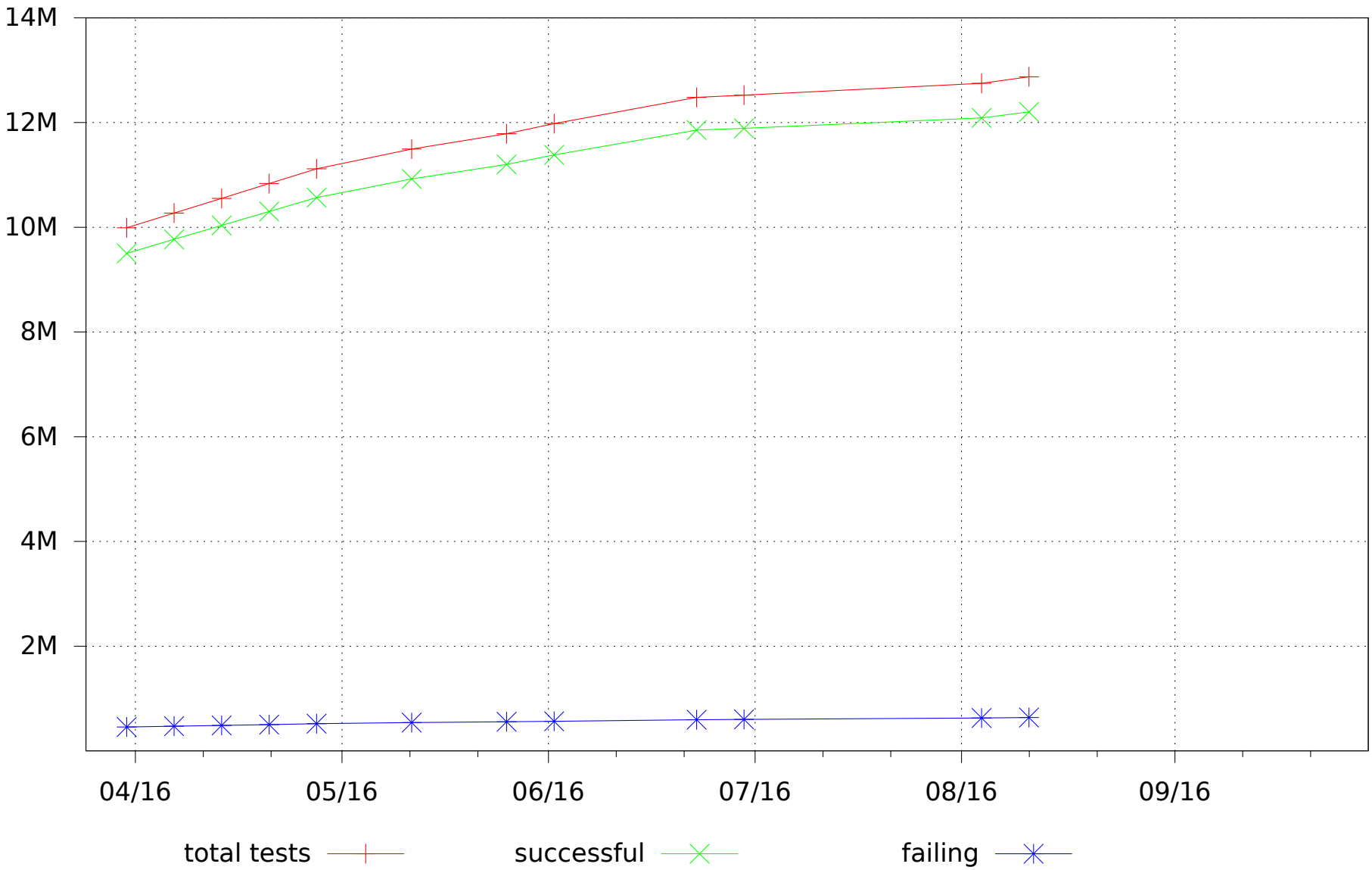
Work after Q1/2015



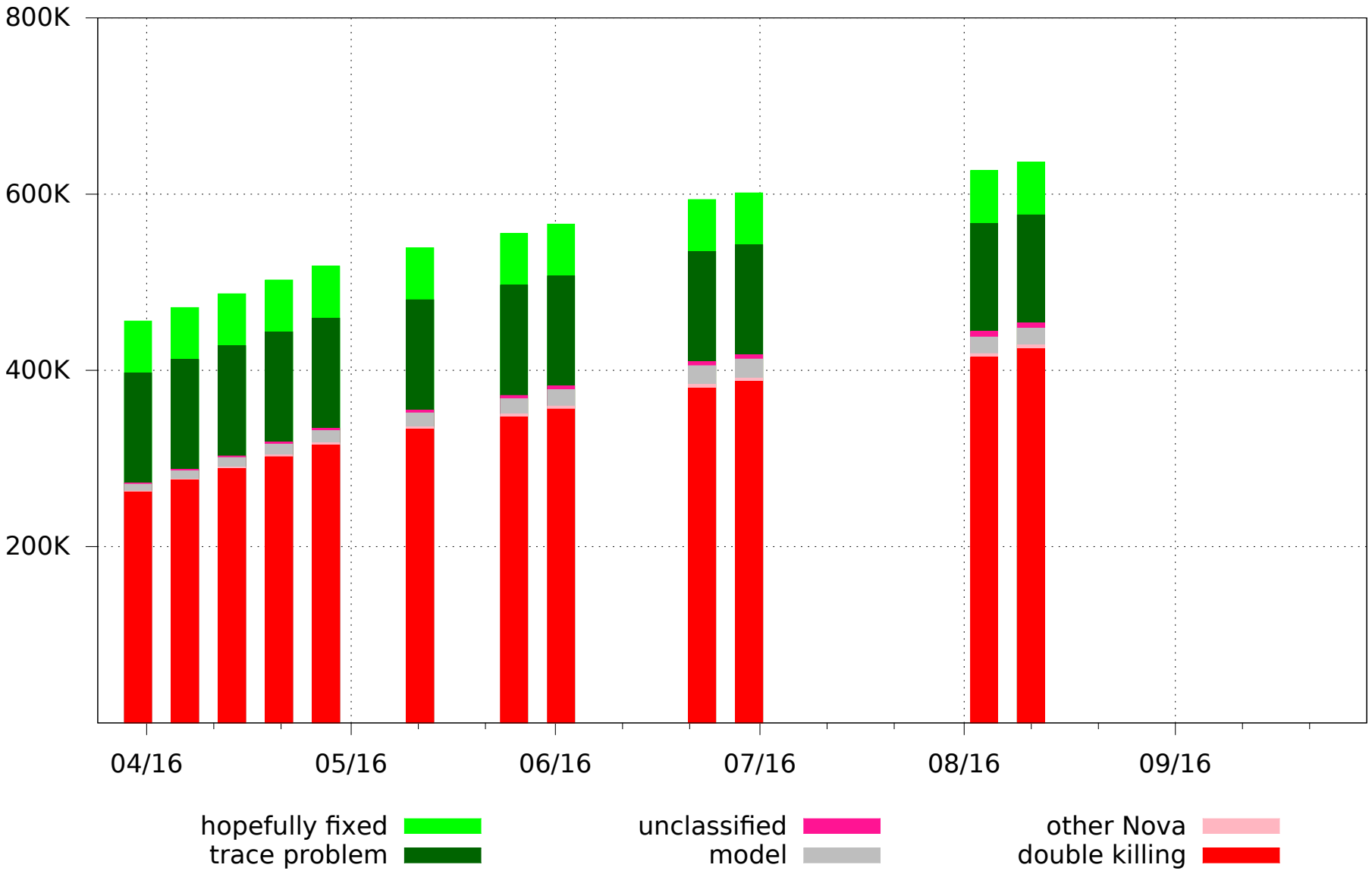
Additionally

- experiments with sound static analysis
- coverity methodology
- fuzzing
- model checking critical algorithms

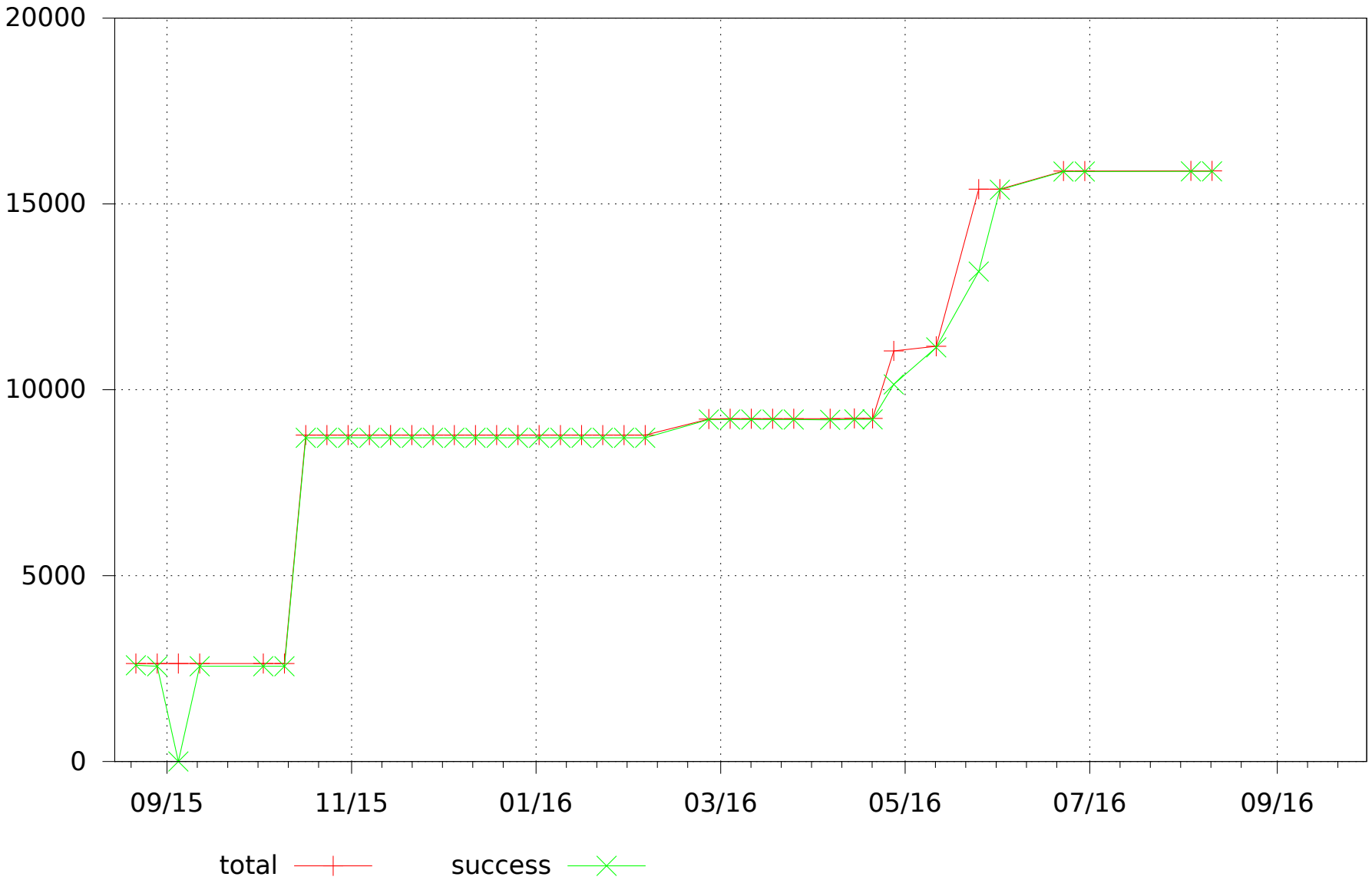
Total number of randomly generated test cases



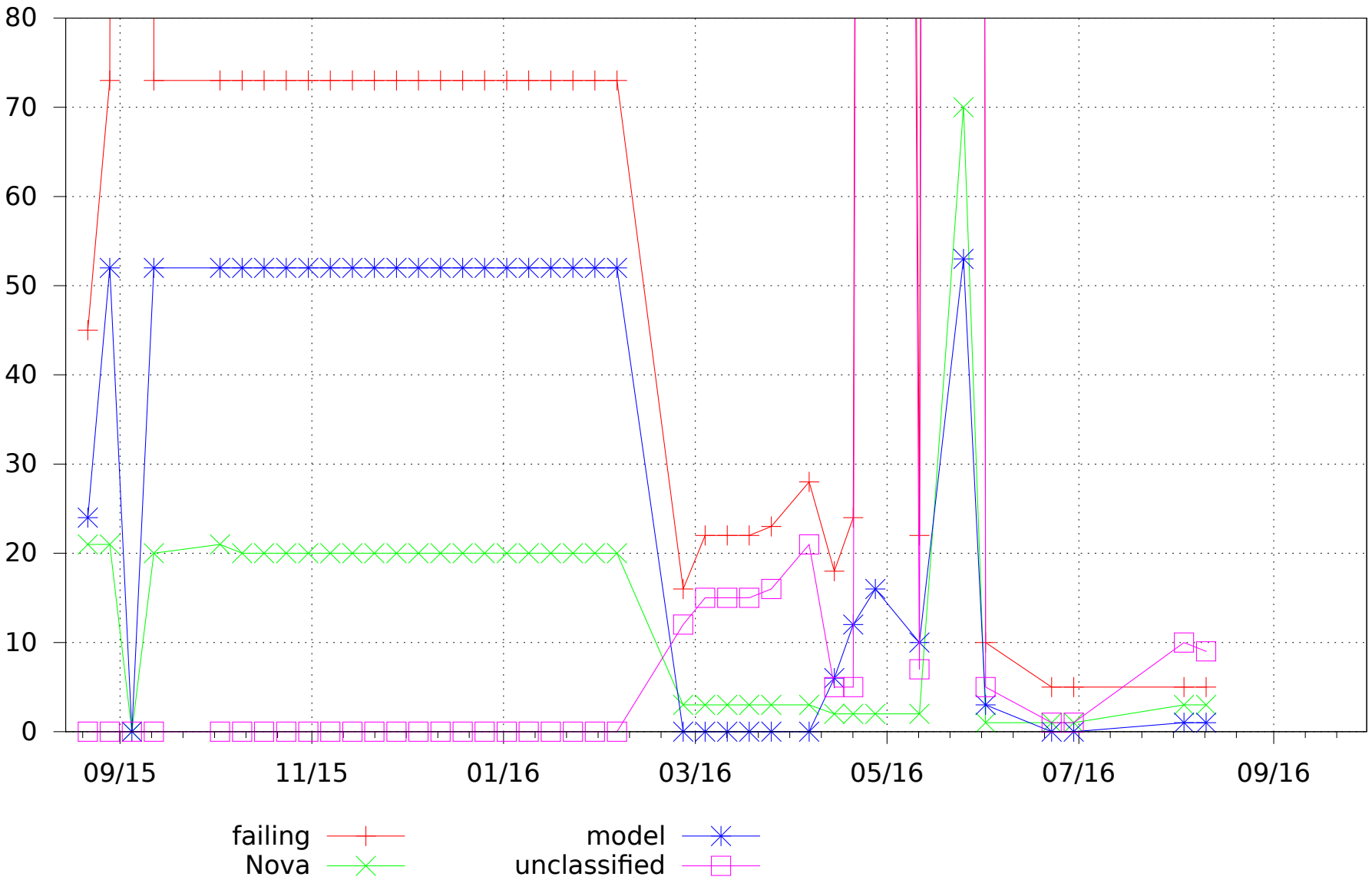
Break down of failing test cases



Regression tests



Failing regression tests



Bugs found

Nova kernel

code review	10
conformance testing	18
total	28

- many boring corner cases
- crashes
- arbitrary kernel memory access
- races

Virtualization layer

coverity	42
code review	41
fuzzing	11
sound static analyzer	6
other	6
total	106

$\approx$ 1 Bug per FM person month on average

Lessons learned

Model based testing

- conformance testing was effective, but a lot of work (kernel testing is hard)
- Heisenberg effects on the borderline of invalidating test results
- need to plan in advance:
  - time frame for fixing *uninteresting* bugs
  - rerunning outdated tests for bug-fix validation
- need to support testing without the model

Other

- top-down approach fulfilled expectations
  first verification results long before product demonstrator
- Maybe other properties are much more relevant?
  - VMM does not introduce bugs in the virtualized OS
  - correctness of parts outside the TCB are necessary to avoid guest OS crashes
    e.g., virtual APIC
- need more quick tools/methods for improving code quality

Lessons learned

Model based testing

- conformance testing was effective, but a lot of work (kernel testing is hard)
- Heisenberg effects on the borderline of invalidating test results
- need to plan in advance:
  - time frame for fixing *uninteresting* bugs
  - rerunning outdated tests for bug-fix validation
- need to support testing without the model

Other

- top-down approach fulfilled expectations
  first verification results long before product demonstrator
- Maybe other properties are much more relevant?
  - VMM does not introduce bugs in the virtualized OS
  - correctness of parts outside the TCB are necessary to avoid guest OS crashes
    e.g., virtual APIC
- need more quick tools/methods for improving code quality

System Architecture (slide copied)