Performance-Driven Interface Contract Enforcement for Scientific Components

10th International Symposium on Component-Based Software Engineering
Medford, MA USA
July 9-11, 2007

Tamara “Tammy” Dahlgren (dahlgren1@llnl.gov)
http://www.llnl.gov/comp/bio.php/dahlgren1

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

Scientific Discovery through Advanced Computing

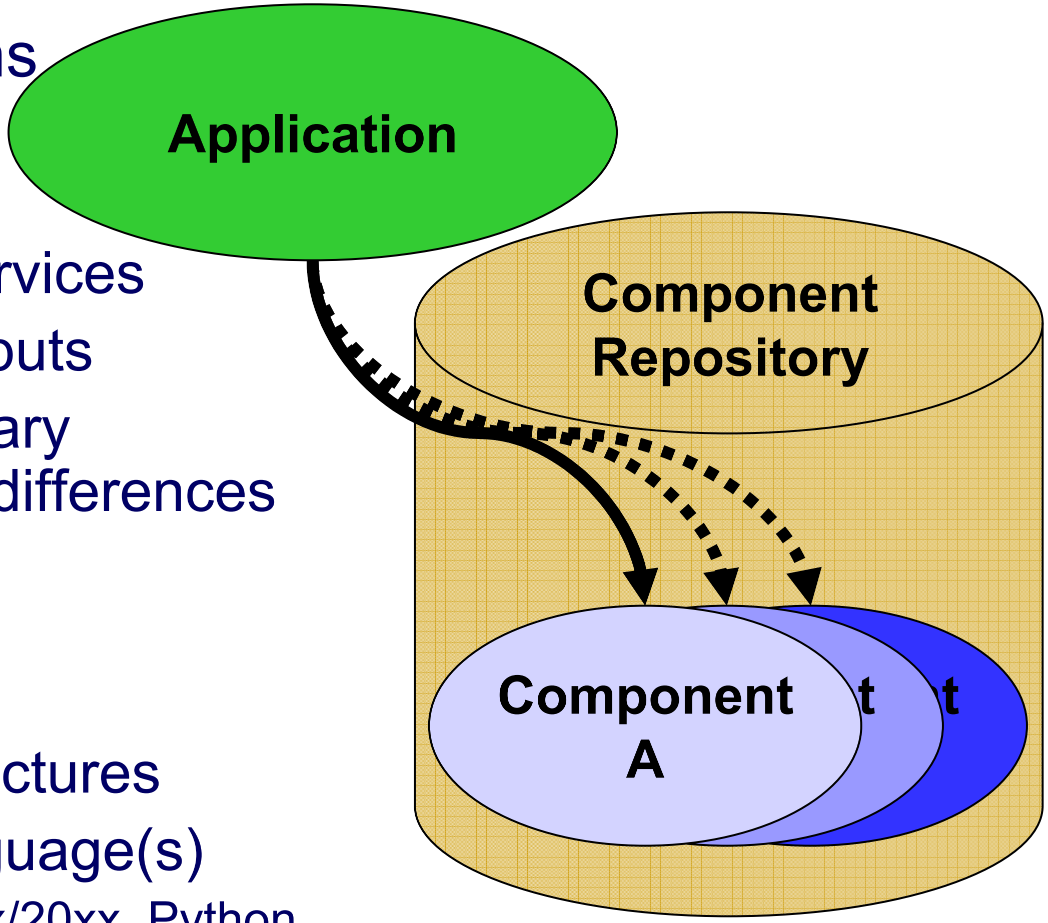
UCRL-PRES-232517

Performance-Driven Interface Contract Enforcement for Scientific Components

- Motivation
- Babel Toolkit
- Experiments
- Summary

Applications built using plug-and-play components depend on common interfaces.

- Multiple implementations conform to the same specification
  - Provide same basic services
  - Require same basic inputs
  - Implementations can vary significantly to include differences in…
    - Algorithms
      - Solution accuracies
    - Underlying data structures
    - Implementation language(s)
      - C, C++, Fortran 77/9x/20xx, Python, Java



Different organizations → different development processes (and rigor)

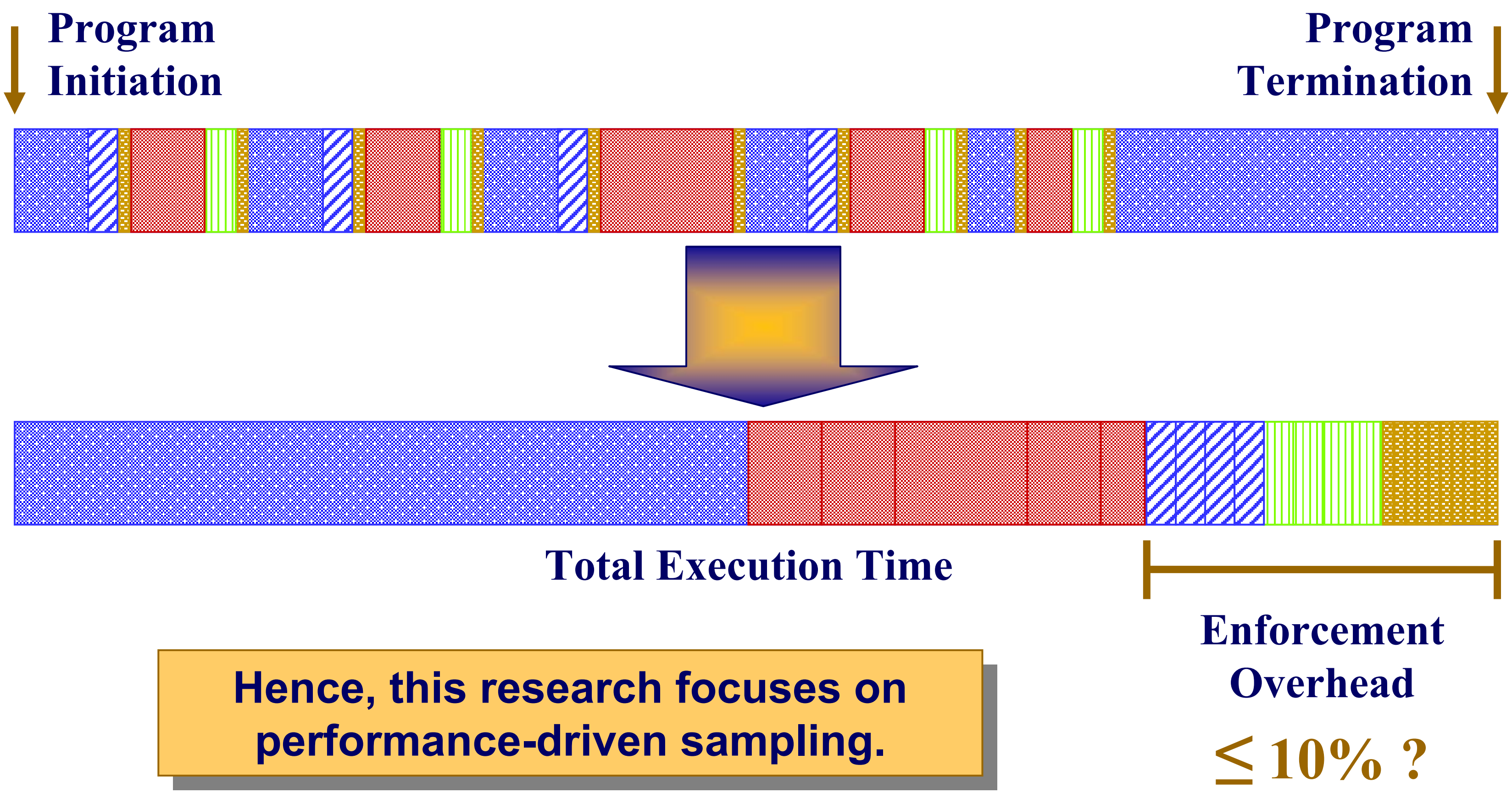
Contracts added to common interfaces can be used to improve software quality.

```
package vector version 1.0 {
  interface Utils { …

    double norm (in array<double> u, in double tol)
      throws                       /* Exceptions */
        sidl.PreViolation, NegativeValueException,
        sidl.PostViolation;

      require                      /* Preconditions */
        not_null : u != null;
        u_is_1d : dimen (u) == 1;
        non_negative_tolerance : tol >= 0.0;

      ensure                       /* Postconditions */
        no_side_effects : is pure;
        non_negative_result : result >= 0.0;
        nearEqual (result, 0.0, tol) iff isZero (u, tol);
  … }
}
```

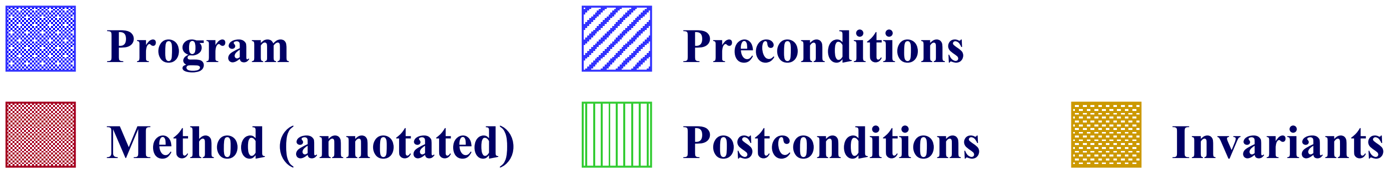
vector.Utils.isZero (u, tol), *which would typically be O(|u|)*

Example based on Babel's vector.sidl (class) specification

Computational Scientists are typically willing to incur no more than 10% overhead.



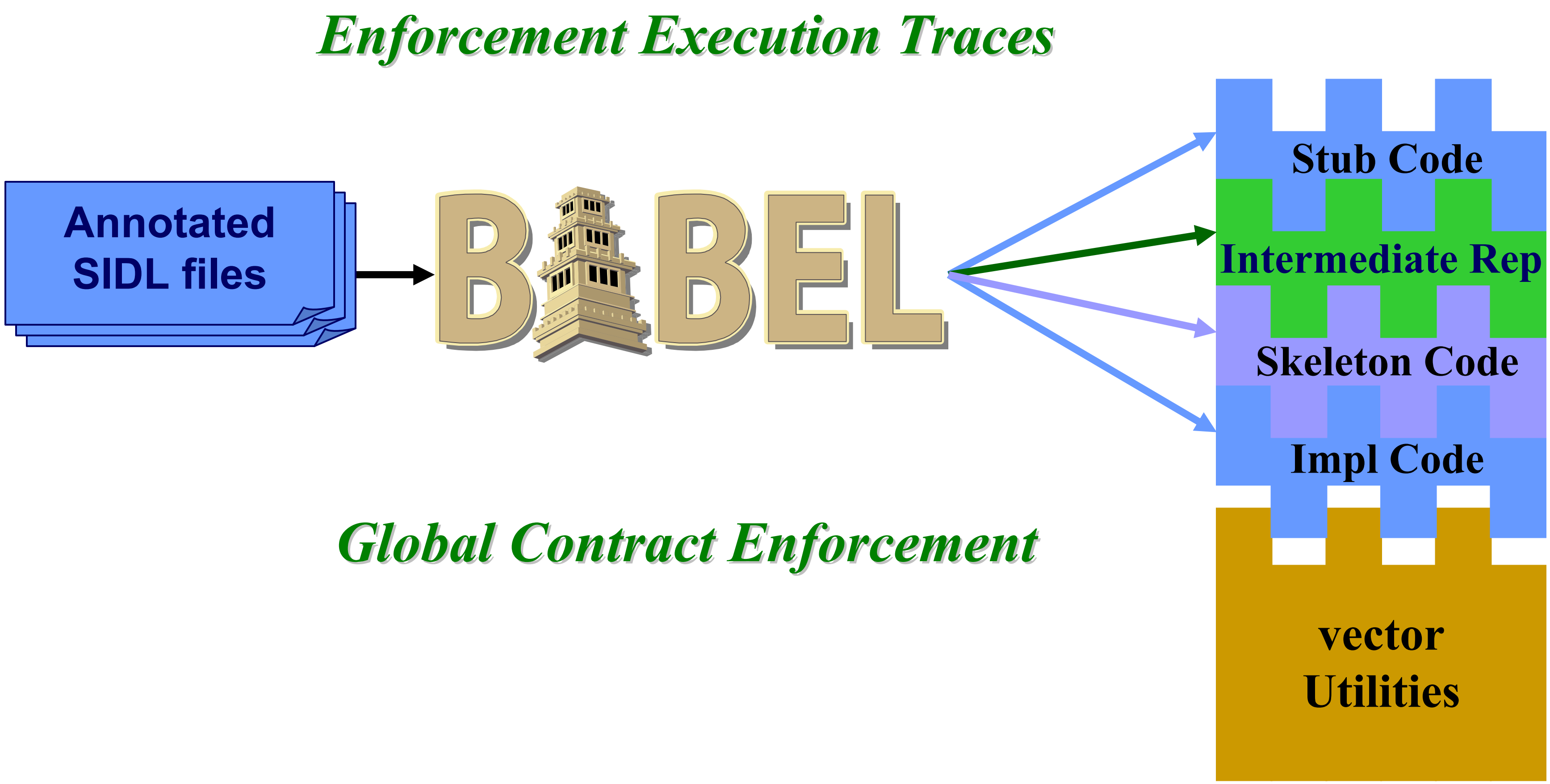
Hence, this research focuses on performance-driven sampling.



Performance-Driven Interface Contract Enforcement for Scientific Components

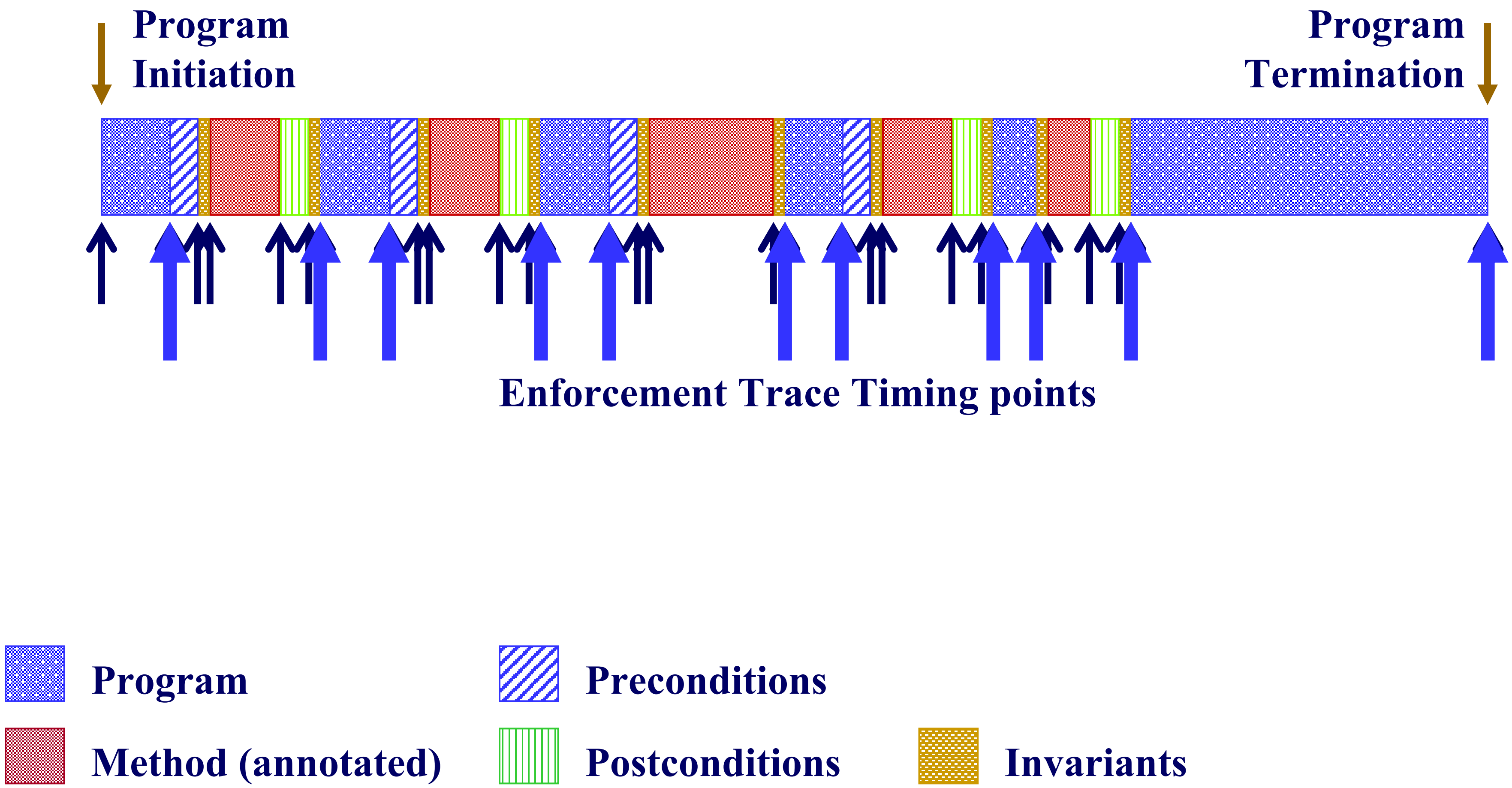
- Motivation
- Babel Toolkit
- Experiments
- Summary

Enforcement automatically generated by Babel* language interoperability toolkit.



*Experiments were conducted using an experimental version of Babel based on release 0.10.8.

Enforcement tracing currently provides simple timing dumps on exercised contracts.



Global enforcement options are based on two parameters: frequency and type.

Enforcement Frequency	Contract [Clause] Type
Never	All
Always	Constant-time
Periodic	Linear-time
Random	Preconditions*
Adaptive Fit (AF)	Postconditions*
Adaptive Timing (AT)	Invariants*
Simulated Annealing (SA)	Simple Expressions
	Method Calls
	Results

*All combinations of the three Eiffel method clause types are actually available.

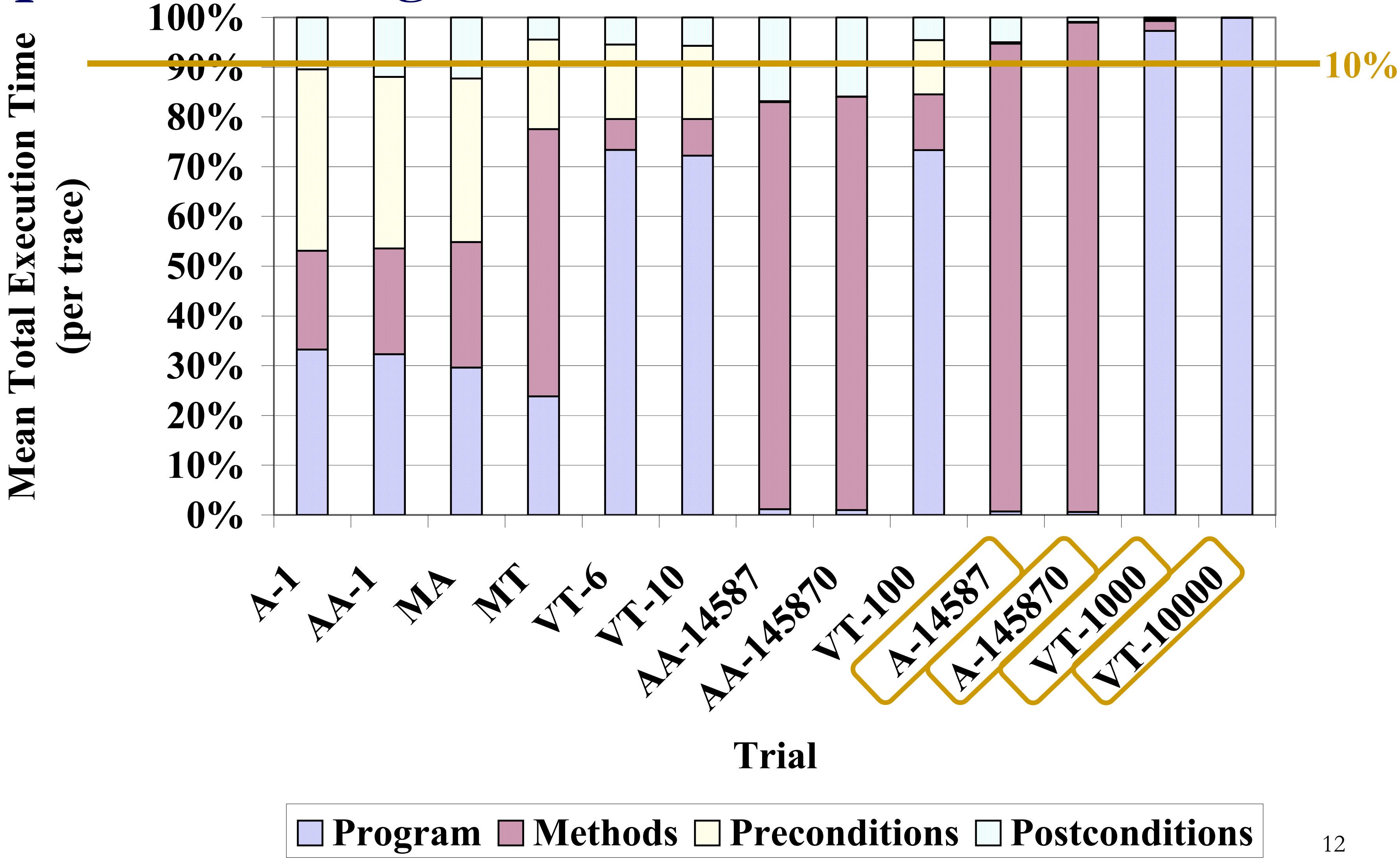
Performance-Driven Interface Contract Enforcement for Scientific Components

- Motivation
- Babel Toolkit
- Experiments
- Summary

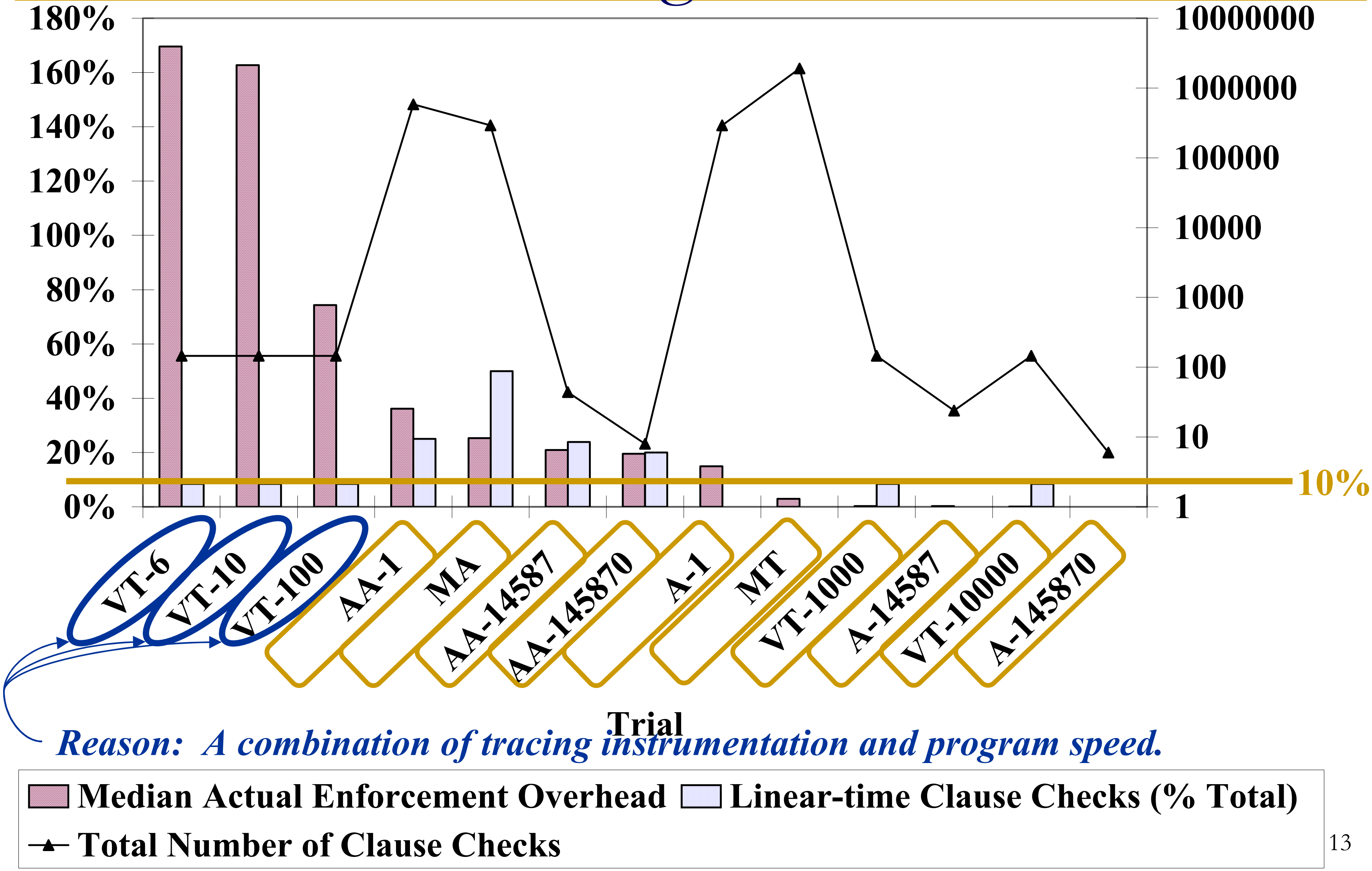
Input sets were varied for three of five programs, forming a total of ***thirteen*** trials.

Component	Program	Description
Simplicial Mesh	MA	Retrieve all faces from the mesh then, for each face, retrieve the adjacent vertices.
	A	Retrieve all faces from the mesh in sets based on size of input array. Sizes **1**, **14587** (10%), and **145870** (100%).
	AA	Retrieve faces as in A plus, for each set of faces, retrieve their corresponding adjacent vertices. The ***same input sizes*** were used.
GRUMMP 0.2.2b's Volume Mesh	MT	Exercise and check consistency of five mesh interfaces: core, single entity query and traversal, entity array query and traversal, single entity mesh modification, and entity array mesh modification.
Vector Utilities	VT	Exercise all supported functions to include successful execution; one or more precondition violations; and one or more postcondition violations. Sizes **6** (original), **10**, **100**, **1000**, and **10000**.

Baseline experiments resulted in a variety of profiles using contract enforcement traces.

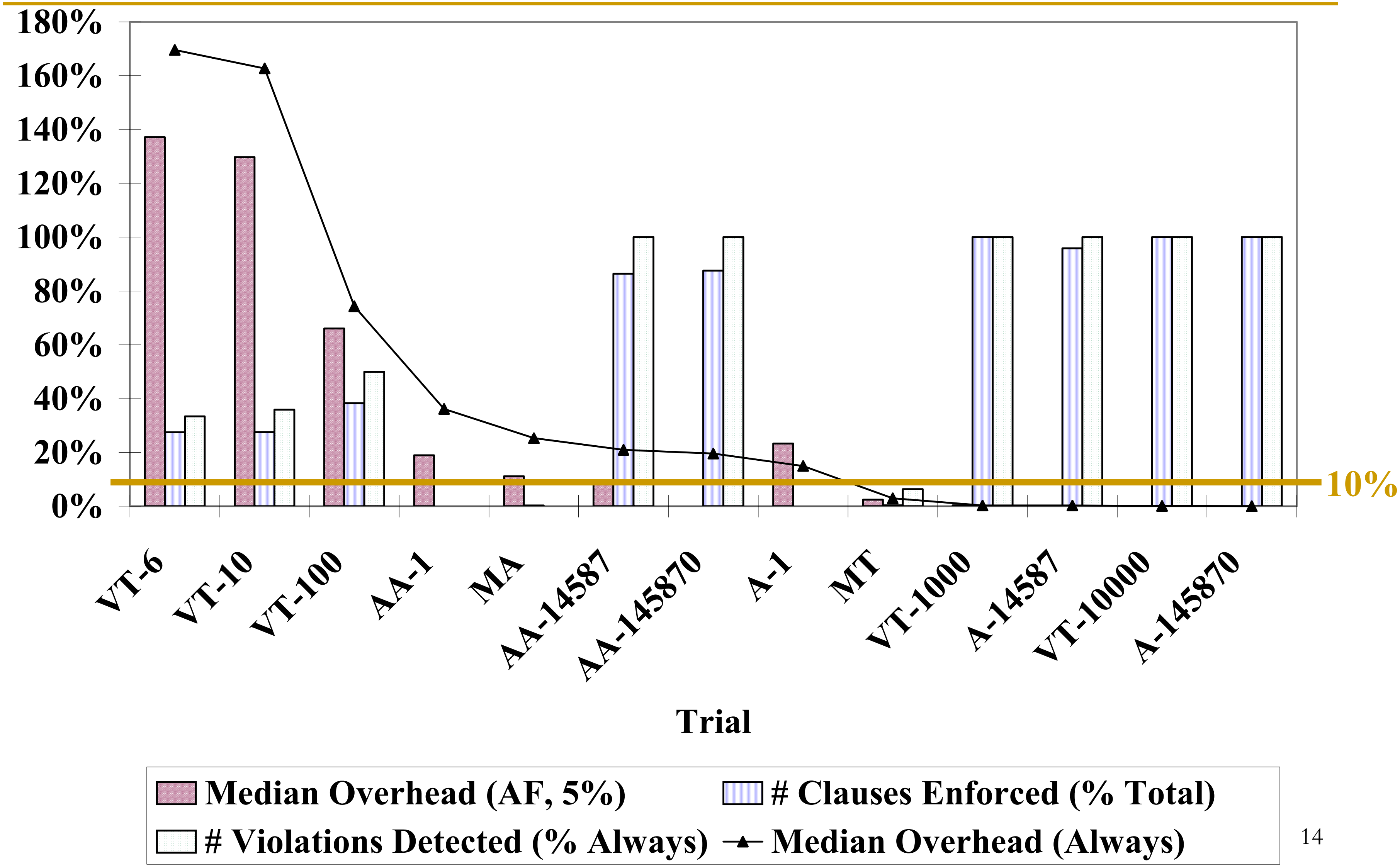


Enforcement performance was *generally* better without tracing instrumentation.

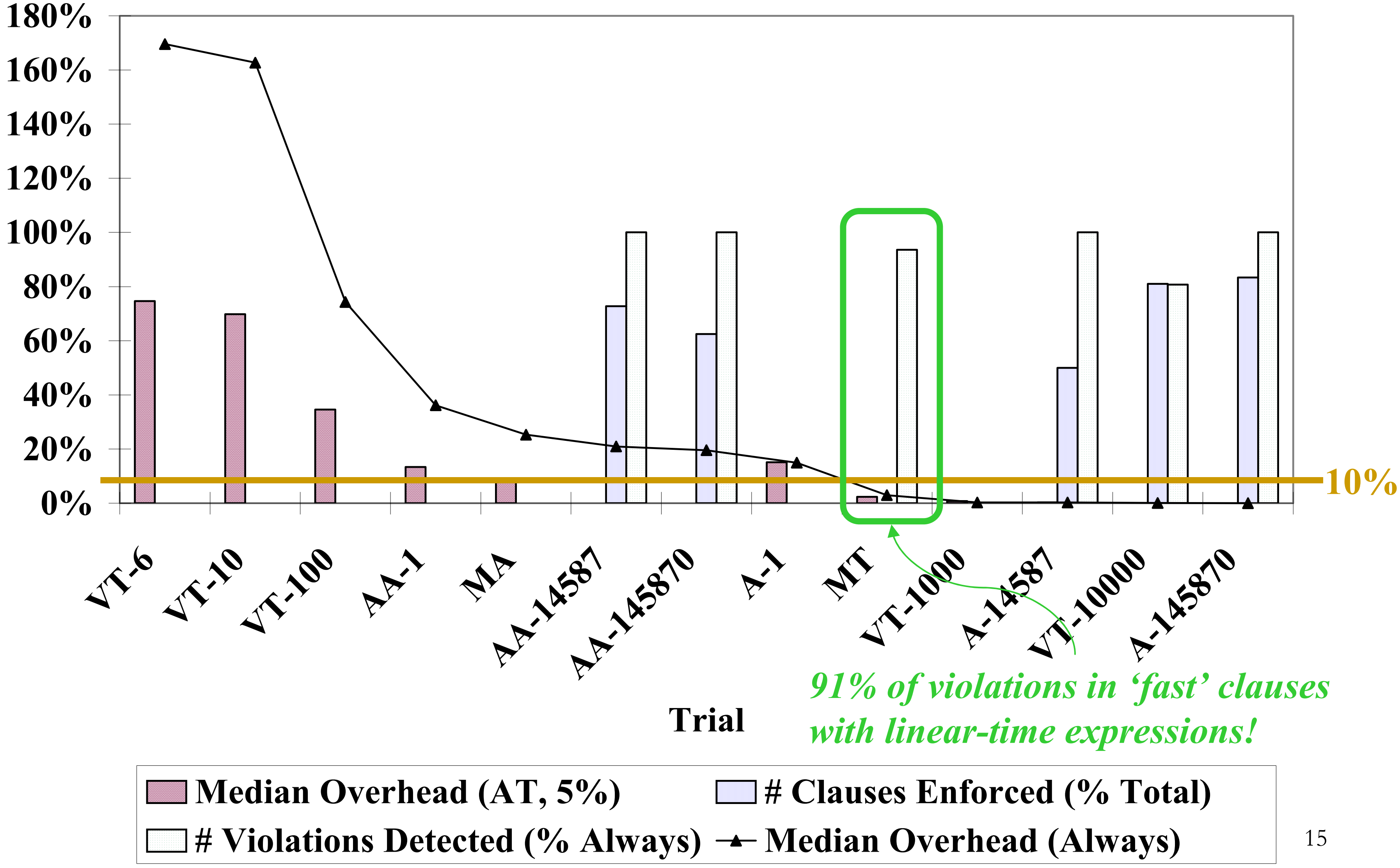


Reason: A combination of tracing instrumentation and program speed.

Adaptive Fit (AF) sampling tuned clause enforcement based on estimated overheads.



Adaptive Timing (AT) is biased toward ‘fast’ clauses relative to the cost of their methods.



Performance-Driven Interface Contract Enforcement for Scientific Components

- Motivation
- Babel Toolkit
- Experiments
- Summary

Performance-Driven Interface Contract Enforcement for Scientific Components

- **Goal**: Improve quality of applications built of automatically swapped, plug-and-play, third-party components
- **Approach**: Use performance criteria to tune contract [clause] enforcement to the program
  - Reduce overhead compared to full enforcement (i.e., *Always)*
  - Increase coverage over other sampling techniques (when appropriate)
    - ➔ Increase probability of detecting more violations
- **Findings**: Performance-driven enforcement appears to be most suited to contract clauses that are at most moderately expensive to check
  - Based on user-specified overhead limit
  - Relative to program/methods

Thank you for your attention.

Any Questions?

For more information related to this work, refer to the following web sites.

- Components Project
  - http://www.llnl.gov/casc/components
  - Note: Experiments conducted using experimental prototype of the Babel toolkit
- Common Component Architecture (CCA) Forum
  - http://cca-forum.org
- Center for Technology for Advanced Scientific Component Software (TASCS)
  - SciDAC's Plug and Play Supercomputing
  - http://www.scidac.gov/compsci/TASCS.html

Supplemental Material

What can be done to ensure plug-and-play components used and implemented correctly?

1950 1960 1970 1980 1990 2000

Theory/Proofs of Correctness

Assertions (Routines) June 1950 (Turing)

Assertions (Programs) 1967/68 (Floyd/Hoare)

Applied Research/Demonstrations of Correctness

Executable Assertions 1977 (Saib)

Design by Contract 1985 [Eiffel]

High-level Component Specs 1994+ [ADL]

This research is based on contracts on the [common] interfaces.

Performance overhead concerns lead to no or partial enforcement during deployment.

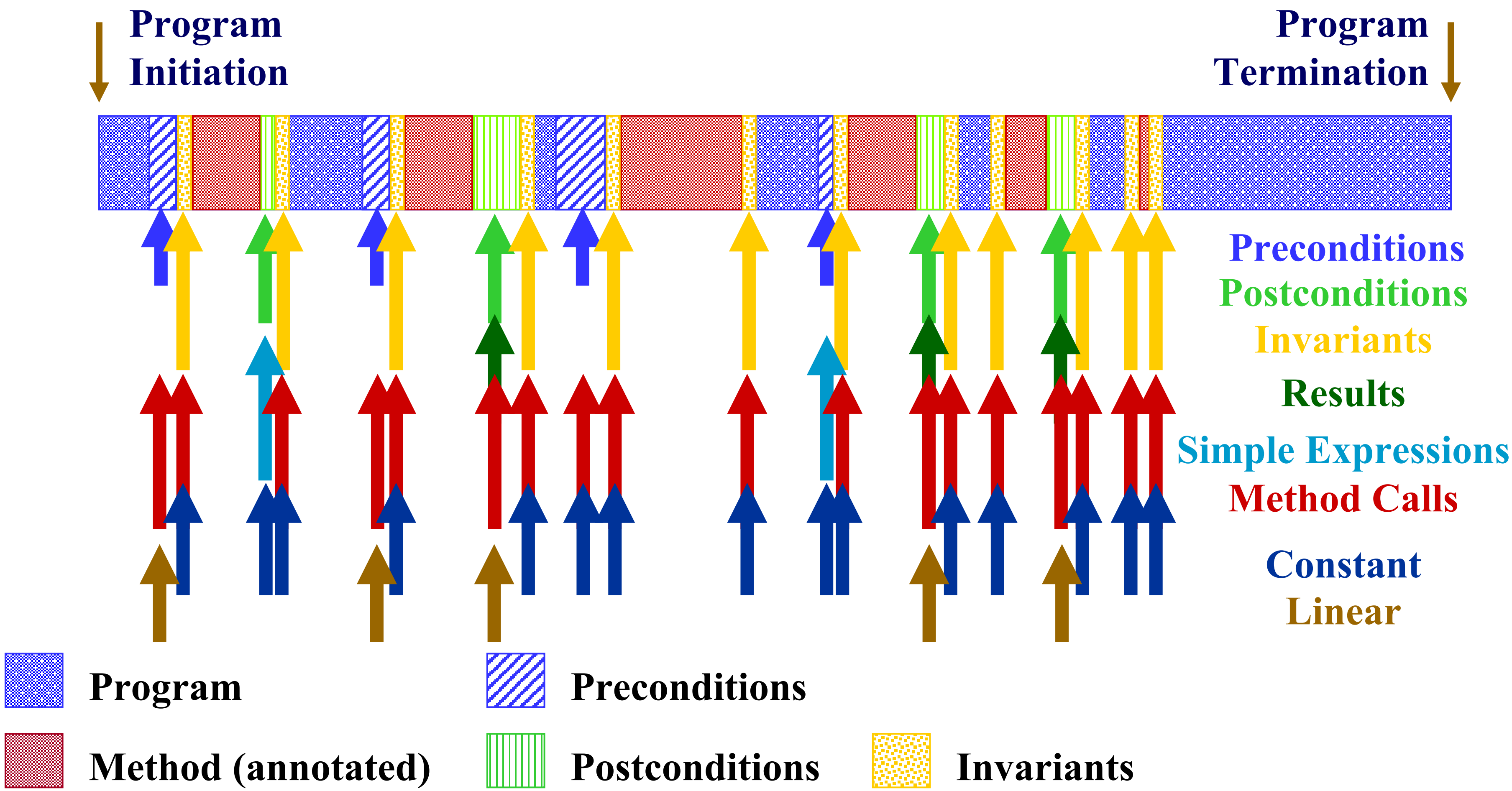
Most common

	All-or-nothing	macros
Selective	Assertion type	Eiffel
Selective	Package/Class	JAF, Jass
Selective	Method	iContract
Selective	Severity	APPC
Selective	Individual	SIFT, ConFract
Sampled		Frequency-based[*], Random[**]

[*] *Chilimbi and Hauswirth, "Low-Overhead Memory Leak Detection Using Adaptive Statistical Profiling," ASPLOS, Oct. 2004.*

[**] *Liblit, Aiken, Zheng, and Jordan, "Bug Isolation via Remote Program Sampling," PLDI '03, June 2003.*

Enforcement decisions are made on a contract clause basis.



Performance-driven variants execute contracts only *if* accumulated enforcement costs do *not* exceed user-specified overhead limit.

Contract characteristics varied across programs, in one case across input sets.

Program	Array Size	Contract Clauses Enforced (by policy)					
		Const.	Linear	SE	MC	Precond	Postcond
MA	*n/a*	50%	50%	0%	100%	50%	50%
A	*All*	100%	0%	50%	50%		
AA	1	75%	25%	25%	75%		
	14587	77%	24%	26%	73%		
	145870	88%	20%	30%	63%		
MT	*n/a*	99.995%	.005%	73%	27%	58%	42%
VT	*All*	95%	8%	0%	100%	80%	33%

One annotated method ≤ Two contract clause enforcement opportunities

Simulated Annealing (SA) sampling performed similar to *AF*, *except* in presence of lots of checks.