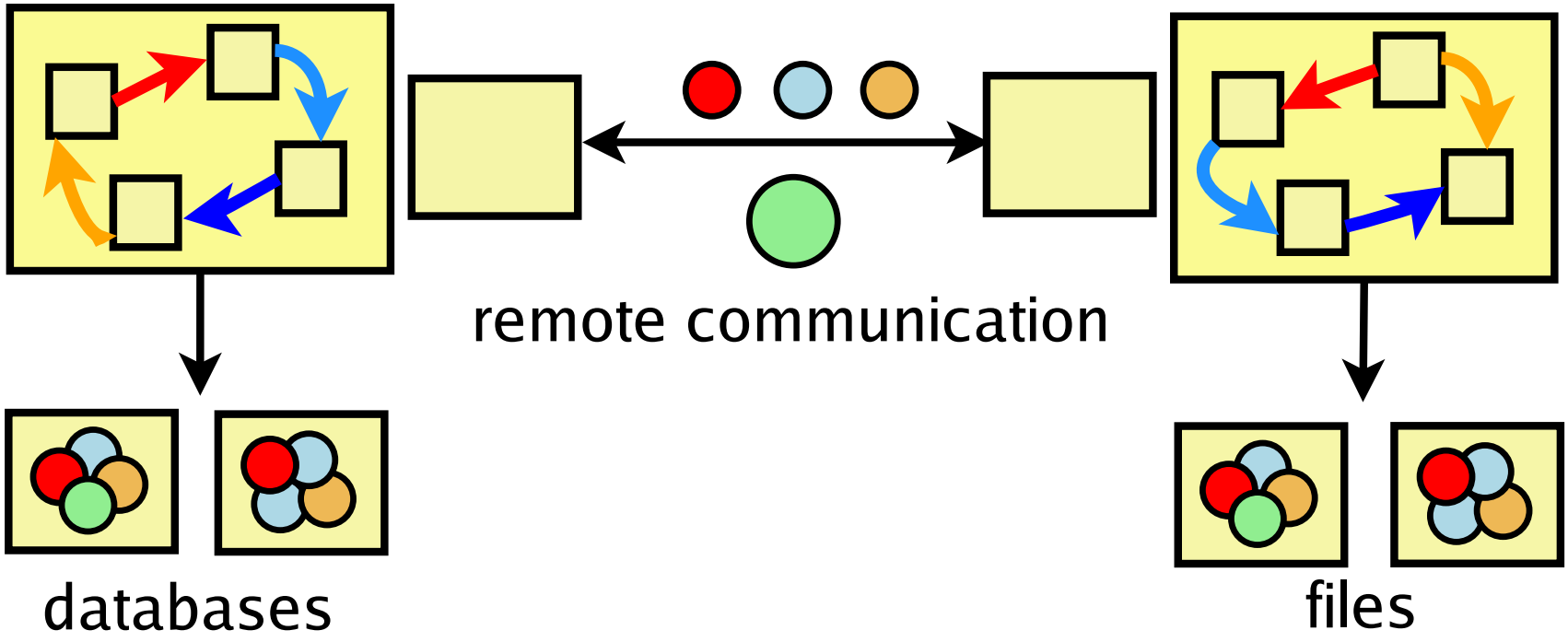
Automatic Program Instrumentation to the Rescue!

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College

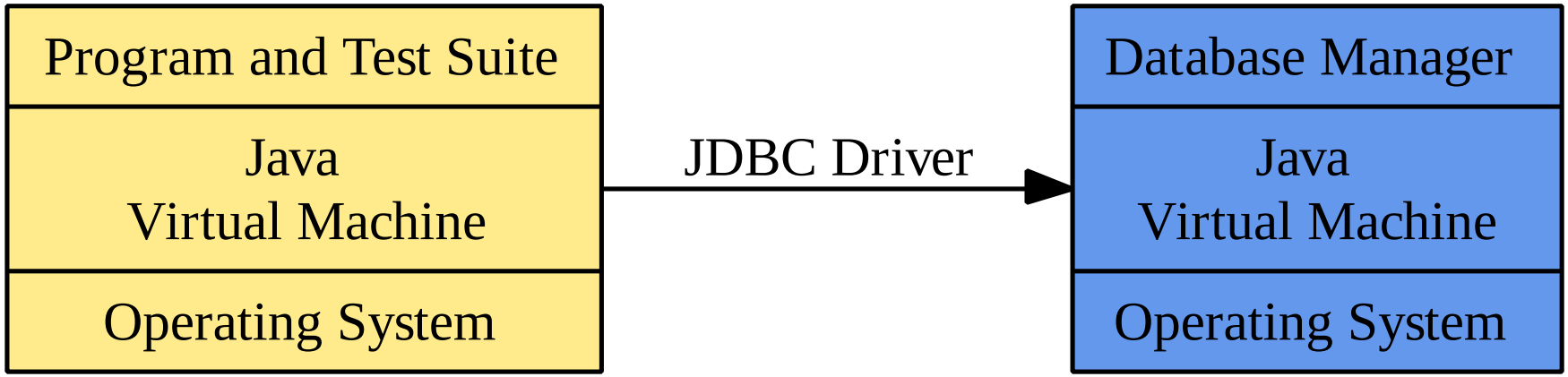
Mary Lou Soffa
Department of Computer Science
University of Virginia

What is My Program Doing?



- **Contribution**: An instrumentation framework to support testing, analysis, debugging, and understanding

Potential Probe Locations

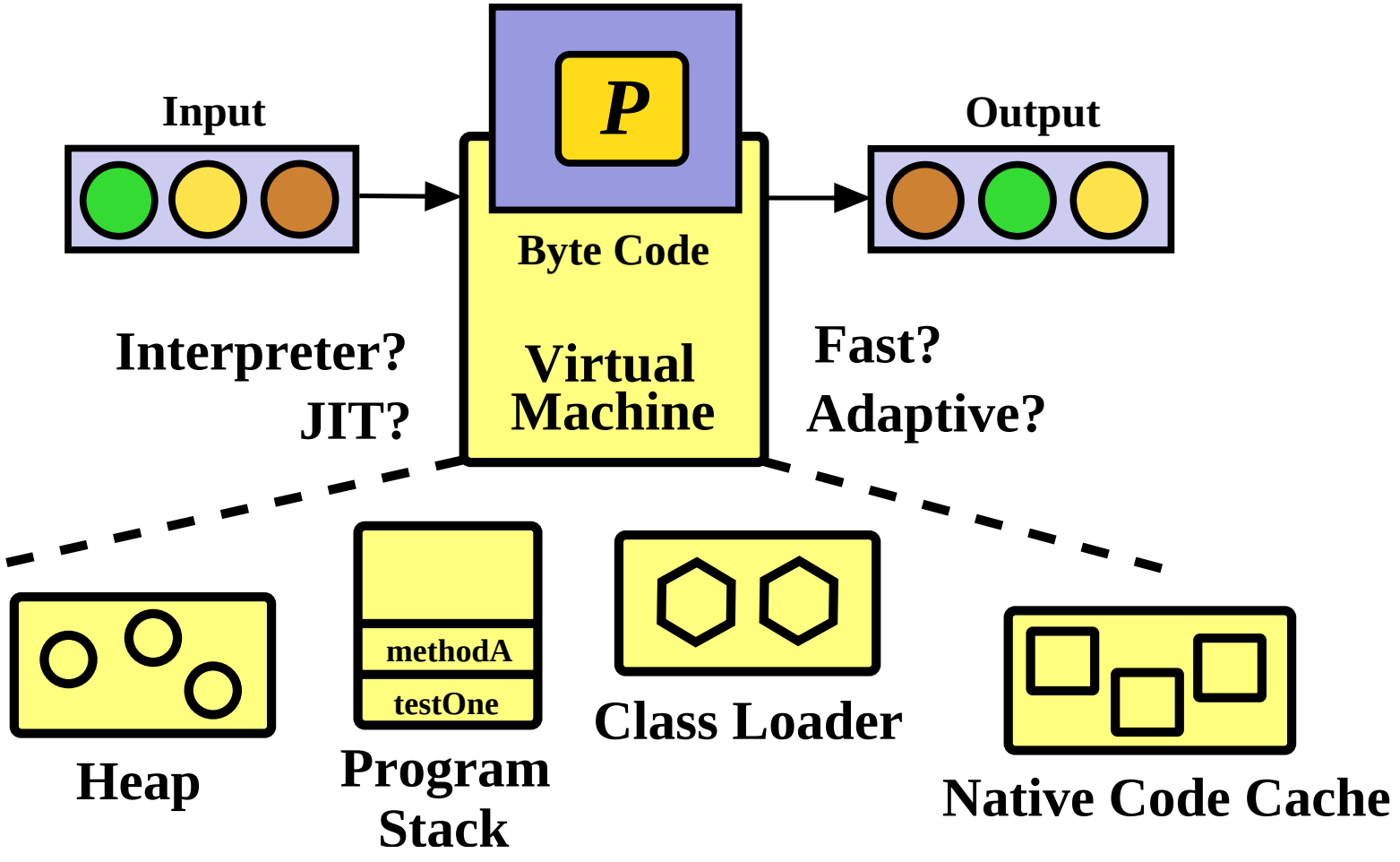


- Instrumentation probes can be placed in *many* locations
- How can we "*best*" capture the behavior of a program?
- Is it possible to *automatically* introduce the probes?
- What tools *already exist*?
- What approach is the most *efficient*?

Understanding Static Instrumentation

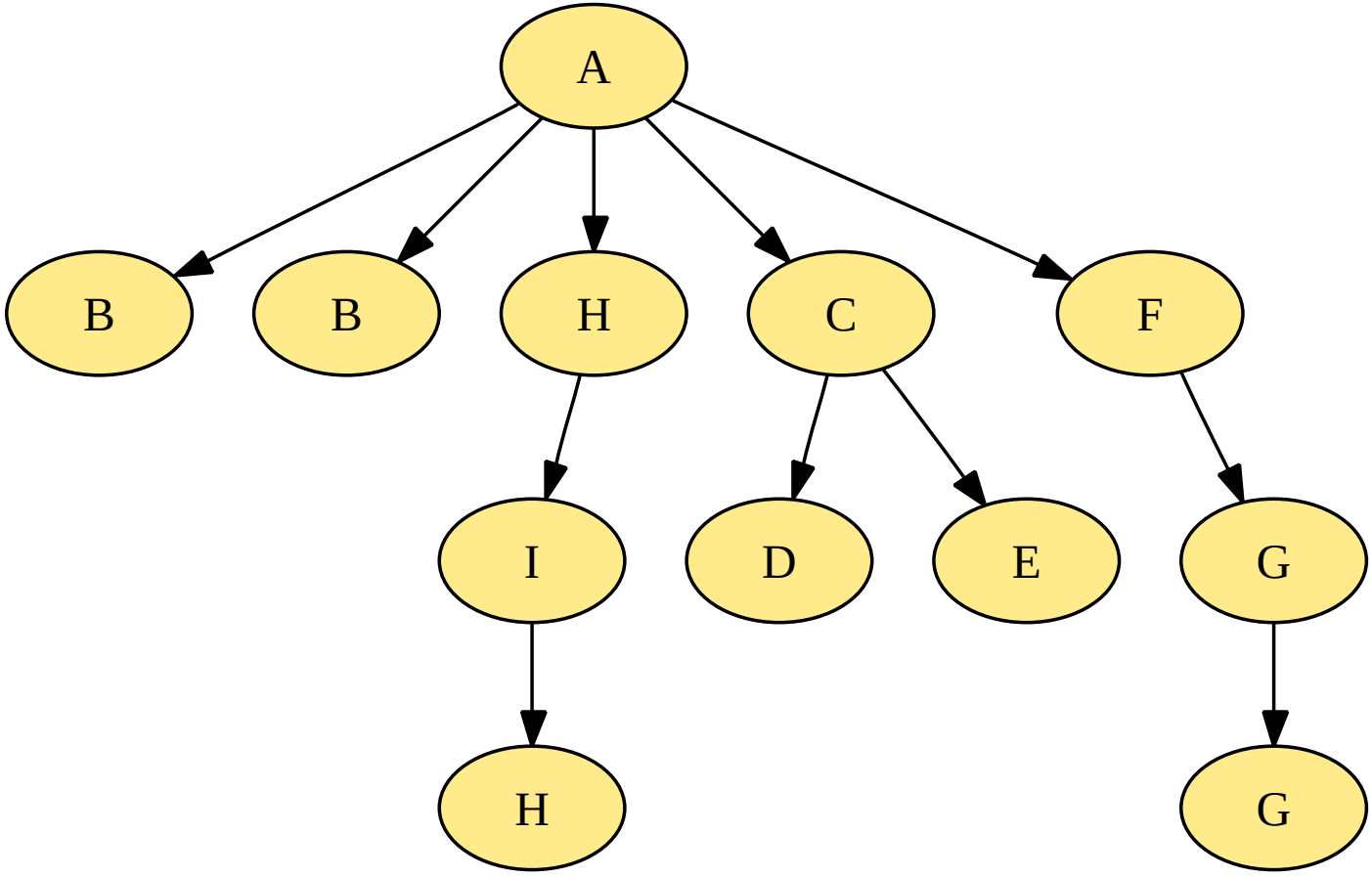
- Insert the probes into the *source code* or *bytecode*
- Instrumentation occurs *before* program execution
- Less *flexible* if a program regularly changes
- What is the impact on *space overhead*?
- Aspect-oriented programming versus bytecode instrumentation

Dynamic Instrumentation



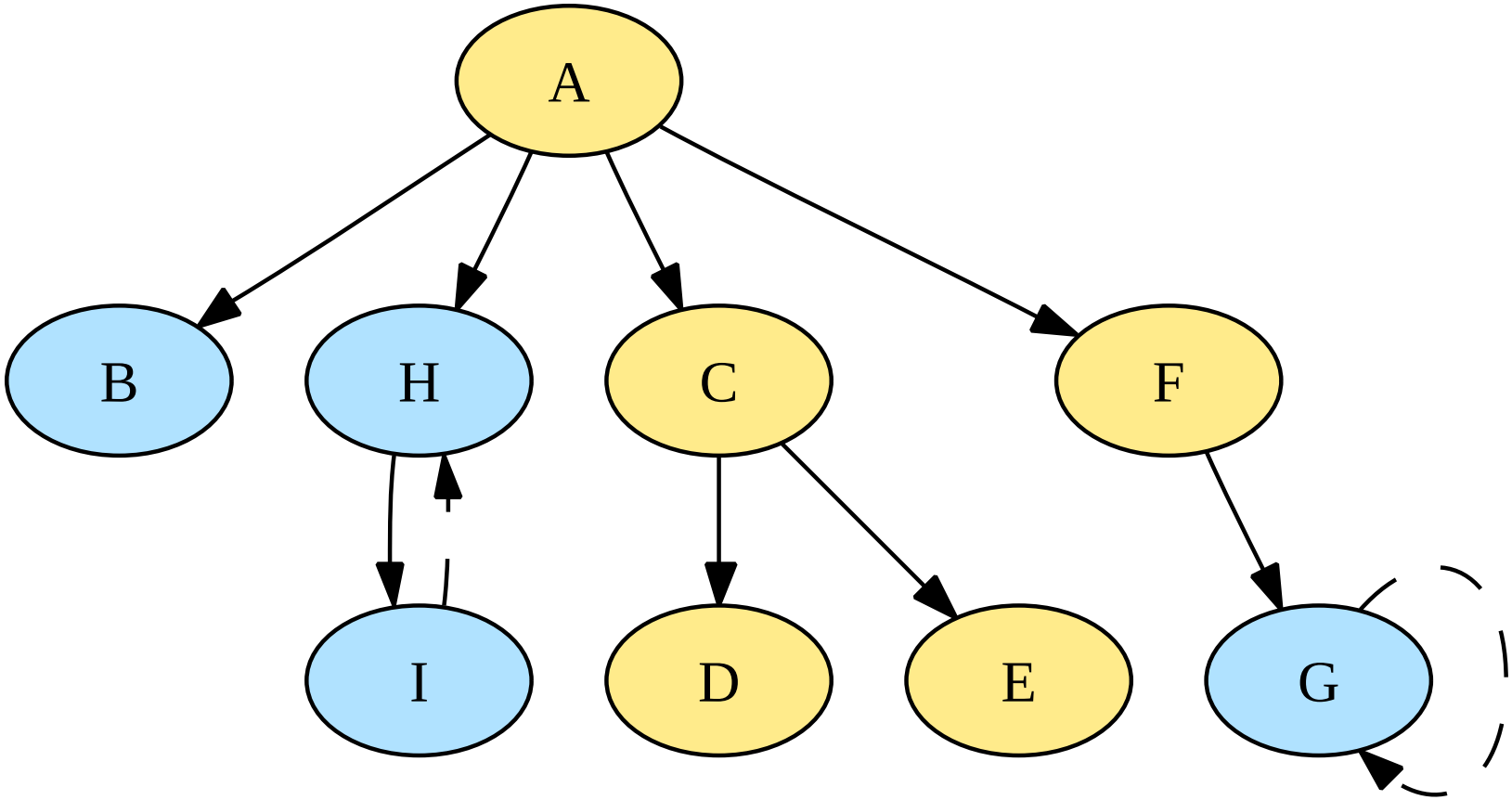
- Perform dynamic instrumentation at established interface(s)

Constructing Dynamic Call Trees



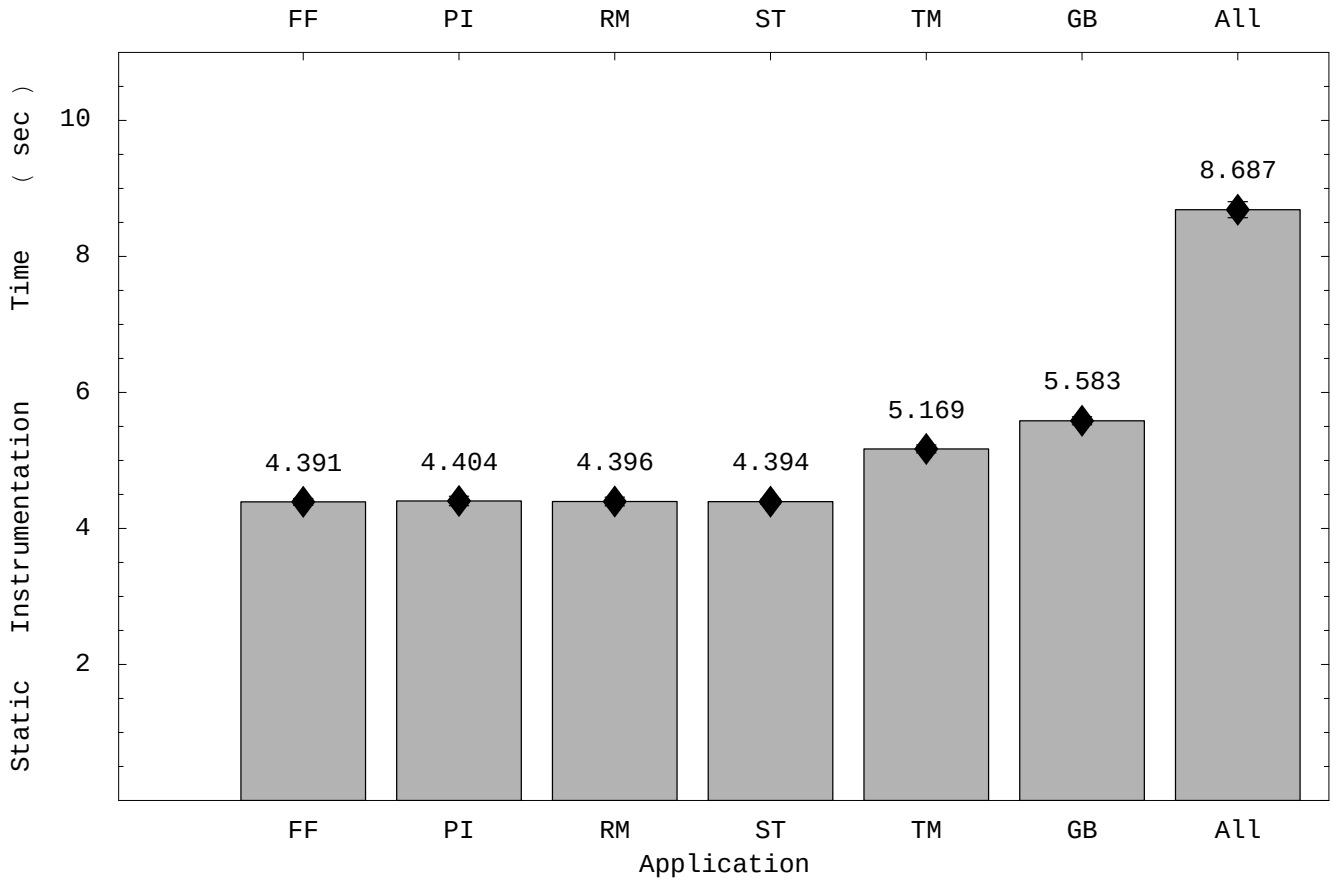
Number of Nodes = 13, Number of Edges = 12

Using Calling Context Trees



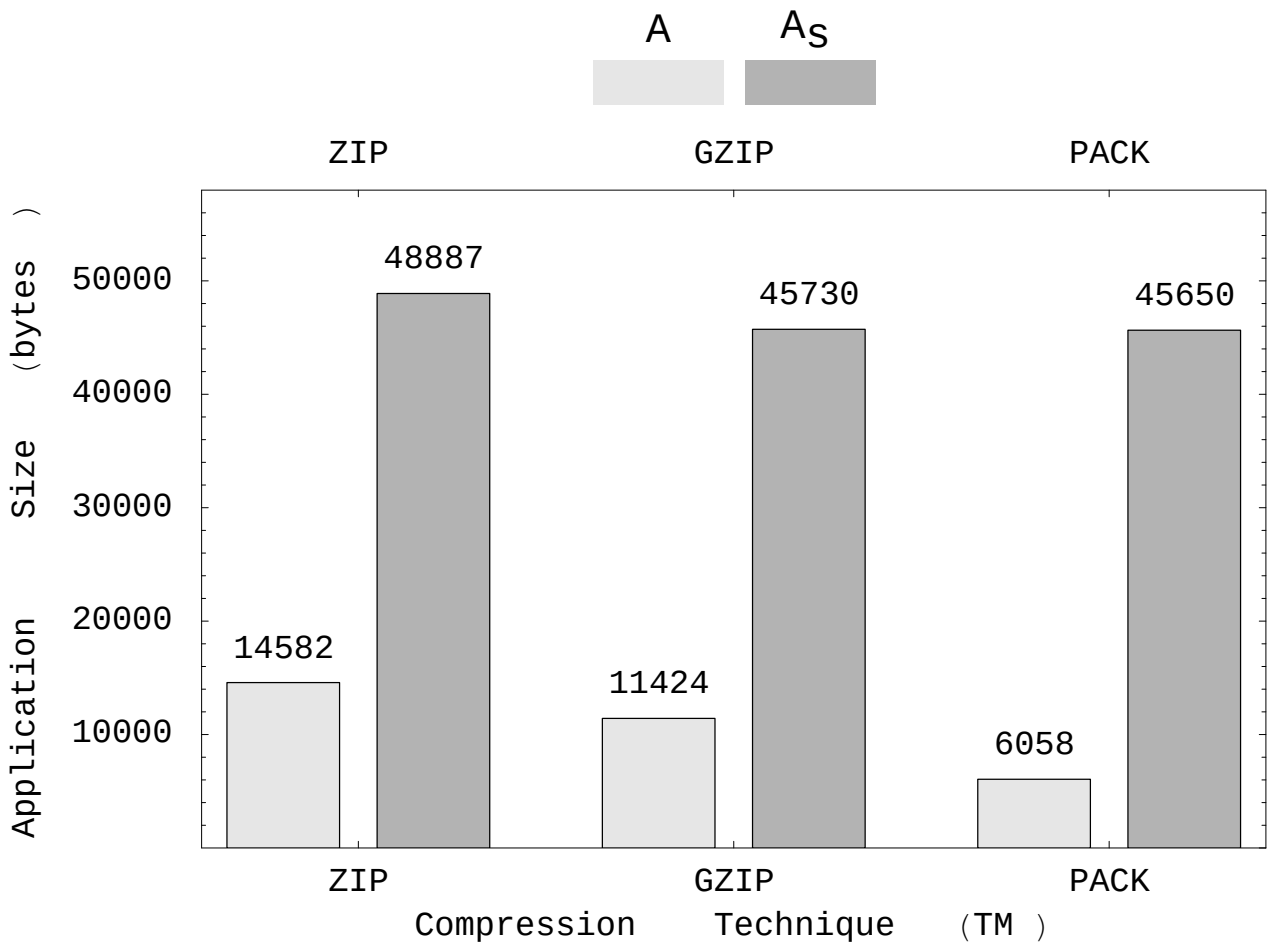
Number of Nodes = 9, Number of Edges = 10

Static Instrumentation Time



- Instrumentation never takes longer than nine seconds

Space Overhead of the Instrumentation



- Increase in the number of bytecodes is substantial

Size of the Instrumented Applications

Compr Tech	Before Instr (bytes)	After Instr (bytes)
None	29275	887609
Zip	15623	41351
Gzip	10624	35594
Pack	5699	34497

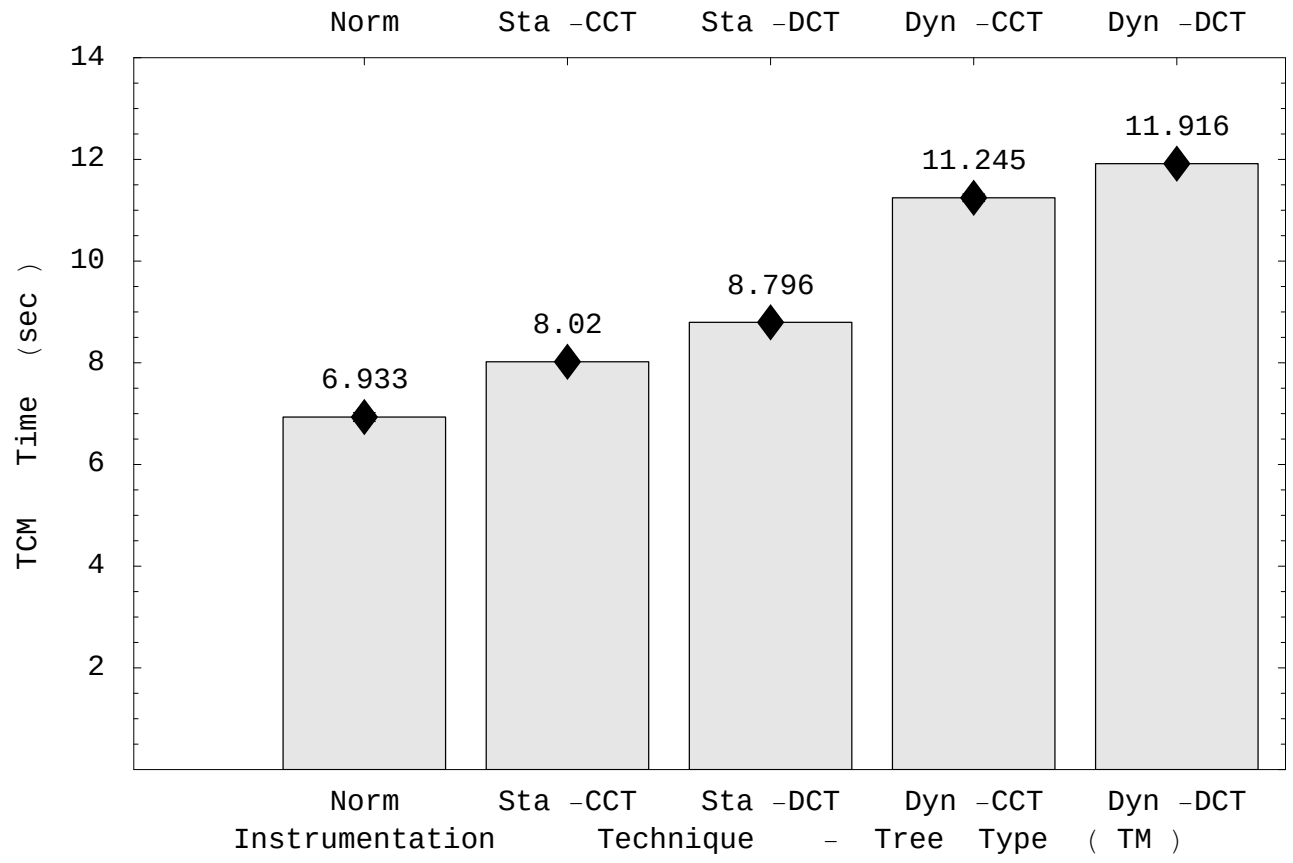
- Average static size across all case study applications
- Compressed the bytecodes with general purpose techniques
- Specialized compressor nicely reduces space overhead

Size of the Instrumentation Probes

Compression Technique	Probe Size (bytes)
None	119205
Zip	40017
Gzip	34982
Pack	35277

- 420% average increase in space overhead!
- *Why?* Reflection vs. extra bytecode instructions
- Is this increase in space overhead *acceptable*?

Static and Dynamic Time Overhead



- What *trends* can you find in this graph?
- Which tree and instrumentation technique would *you* pick?

Test Execution Time Overhead

Instr Tech	Tree Type	TCM Time (sec)	Percent Increase (%)
Static	CCT	7.44	12.5
Static	DCT	8.35	26.1
Dynamic	CCT	10.17	53.0
Dynamic	DCT	11.0	66.0

- Normal average testing time of 6.62 seconds
- Which tree and instrumentation technique is most *efficient*?
- Which configuration would *you* select?

Average Tree Storage Time

Tree Type	Tree Representation	Tree Storage Time (msec)
CCT	Binary	144.9
DCT	Binary	1011.72
CCT	XML	408.17
DCT	XML	2569.22

- Strengths and weaknesses of tree representations
- Is it *ever* better to store the tree in XML?
- Which configuration would *you* use?

Conclusions and Future Work

- Automatic program instrumentation can *save the day*!
- A complete framework for recording call trees
- *Useful applications*: test coverage monitoring, performance analysis, regression test suite reduction
- Characterizing the DCT and CCT for object-oriented programs
- Automatic visualization of method input and output
- What are your suggestions?
- I value your comments and participation!