Improving Error Checking and Unsafe Optimizations using Software Speculation

Kirk Kelsey and Chen Ding
University of Rochester

Outline

- Motivation
  - Brief problem statement
  - How speculation can help
- Our software speculation system
  - Compiler Support, Runtime Libraries, Visual Examples
- An Example Testbed System
  - Compiler Support, Runtime Libraries, Visual Examples
- Empirical Results
  - Reducing the cost of correctness checking
- Conclusion

Problem

- There is a potential to improve any program (in terms of either robustness or speed)

but

- Programs can be extremely complex, and
- Code is often inherited by new programmers

thus

- There is uncertainty in making any change

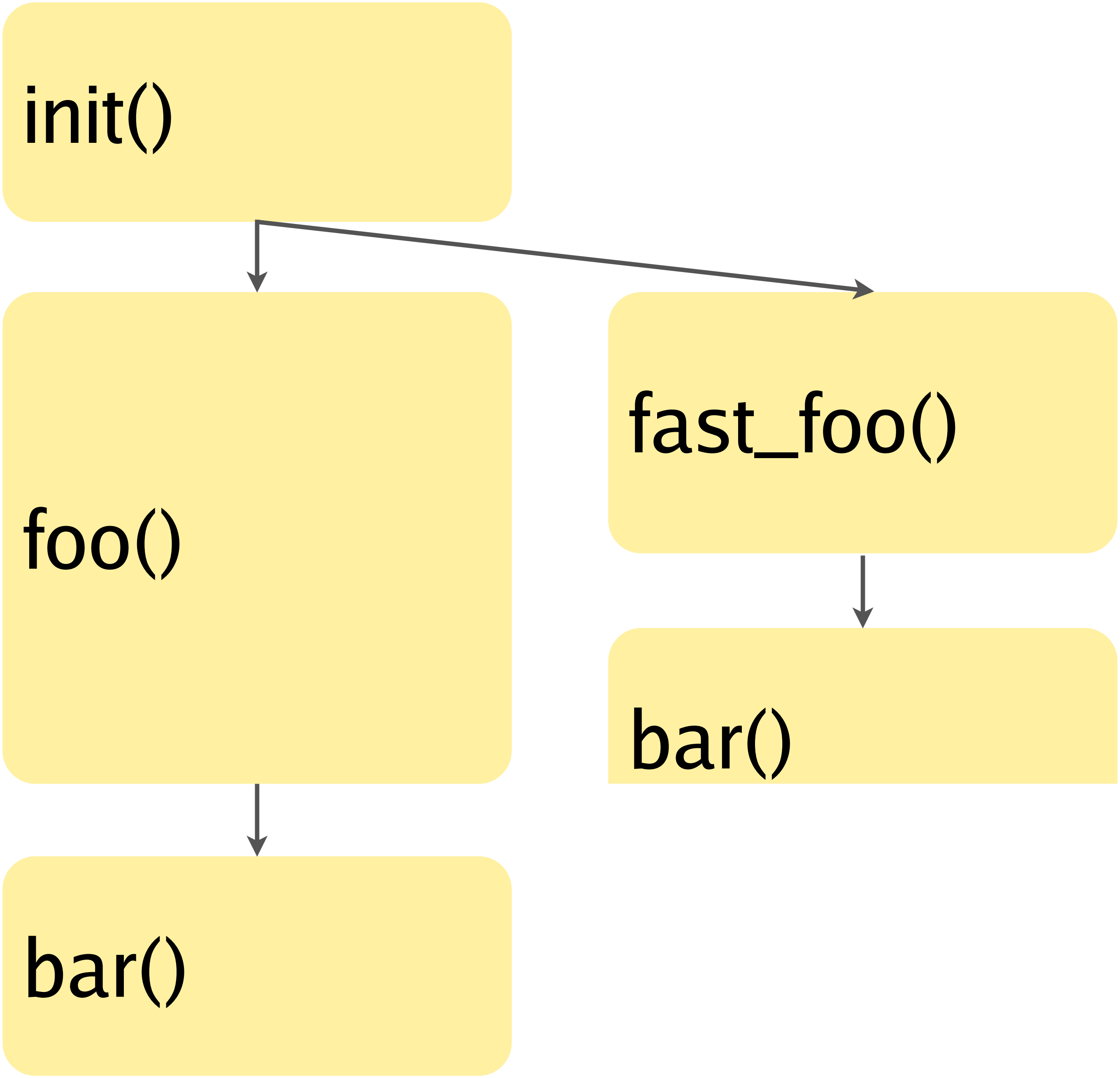
Our Proposal:
Use Software Speculation

- Keep the existing program implementation
- Augment it with an alternative version
- Run the new code speculatively
- Use the original guaranteeing a baseline

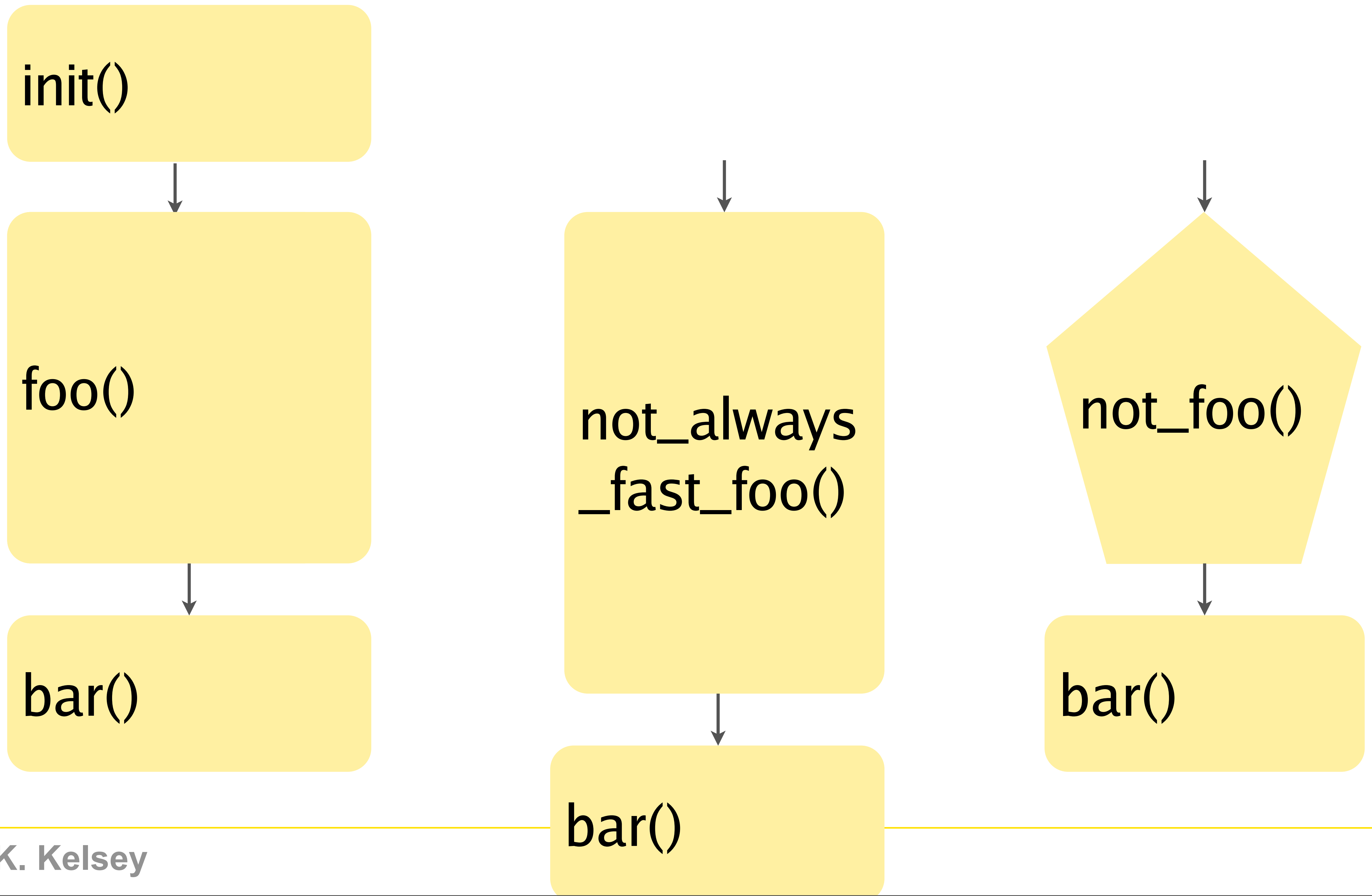
- Since progress is made by whichever execution is faster, we guarantee that the system can only be marginally worse than the original implementation.

- Can be applied to guarantee unsafe optimizations, or to reduce the cost of correctness checking.

Visually...



But what if ...



Fast Track

- Coarse-grained speculative execution using linux processes
- Applies to sequential C/C++ programs
- Requires a runtime library and compiler support
- Automatically selects the faster of two equivalent processes. Our assertion is that two processes are equivalent if:
  – they begin in the same state
  – result in the same memory state
  – are followed by the same instruction sequence

Compiler Support
general

- Based on a modified GCC 4.1
- Moves global variables into heap allocated space
  - inserts new allocation calls
  - adds initialization routines
  - changes accesses so they reference the heap
- Redirects heap [de]allocation so we can track data
- Replaces output functions with buffered versions

Runtime Support

- Forks new processes to execute two versions of code
  - per-process memory space means rollback is simple
  - operating system copy-on-write limits the memory overhead
- Tracks which memory pages each process modifies
  - page fault handers trigger once per page
- Compares memory state after each pair of parallel tracks to ensure correct computation
  - Can ignore data objects known to be “inconsequential” (based on programmer annotation)
  - Correctness checking can customized using a programmer supplied function pointer

Testbed System
& Introduction to Mudflap

- We leverage an existing memory checking system as a demonstration testbed for our speculation system
  – GCC included a memory checking (“pointer debugging”) system called mudflap
  – Mudflap comprises compiler support and runtime libraries
- Use the existing implementation as the fast track
- The overhead of extra checking creates a slow track
- Add more compiler support to automate the process

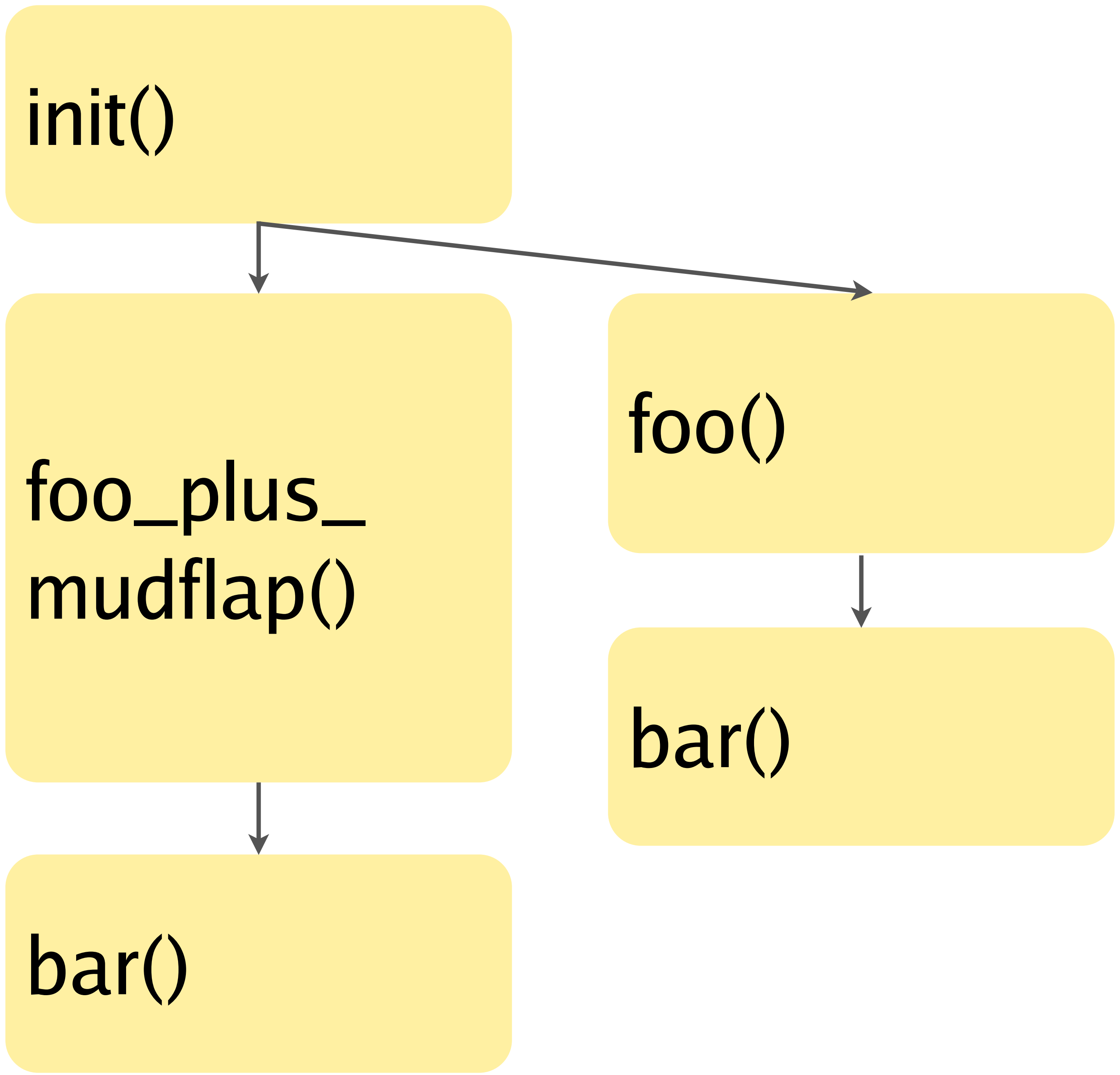
Mudflap
Compiler Support

- Adds code to register some variables at runtime
  - when the address is taken
  - when the bounds are not known statically (e.g. *extern*)
- Inserts runtime checks before pointer dereferences
- Support is activated by a compile time flag (-fmudflap)

Mudflap
Runtime Library

- Tracks ranges of valid memory objects
  - heap allocation
  - objects registered by the static analysis
- Checks for reads of uninitialized objects
- Reports memory leaks
- Wraps some common string and heap accesses (think: strcmp, memcpy)
- Allows for several forms of error reporting
  - spawn gdb
  - output log
  - abort

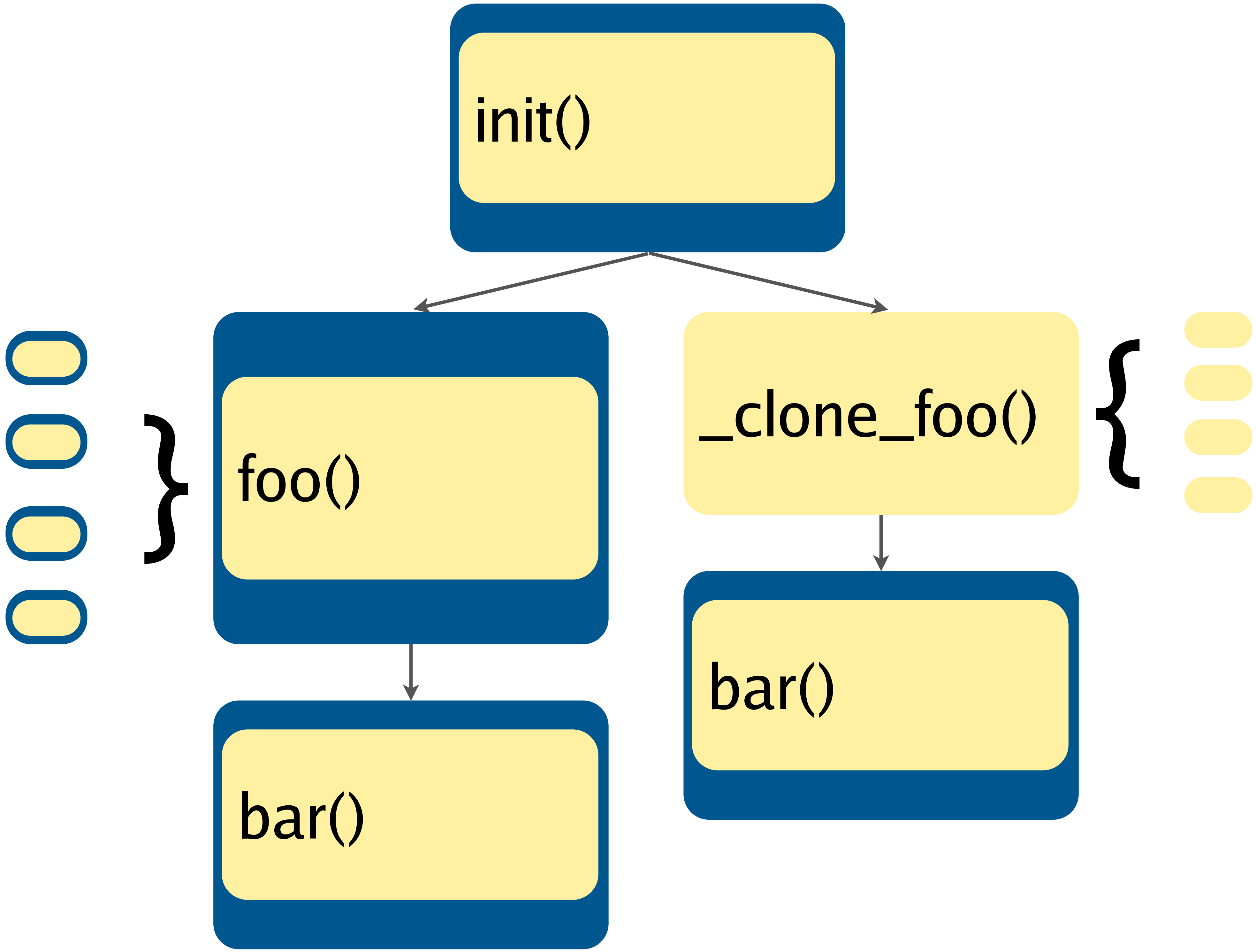
Visually...



More
Compiler Support

- Custom GCC (v4.1) optimization pass
- Generates a copy of each program function (a clone)
- Changes call sites in function copies to call the clone of the original target
- Instructs mudflap to skip instrumentation on each clone
- Programmer must indicate what to fast track
  - by default mudflap is used
  - mudflap variable registration cannot be avoided

Visually...



Finally, in code

The programmer needs to do very little:

```
if( BeginFastTrack())
{
    _clone__foo( );
} else {
    foo( );
}
PostFastTrack();
```

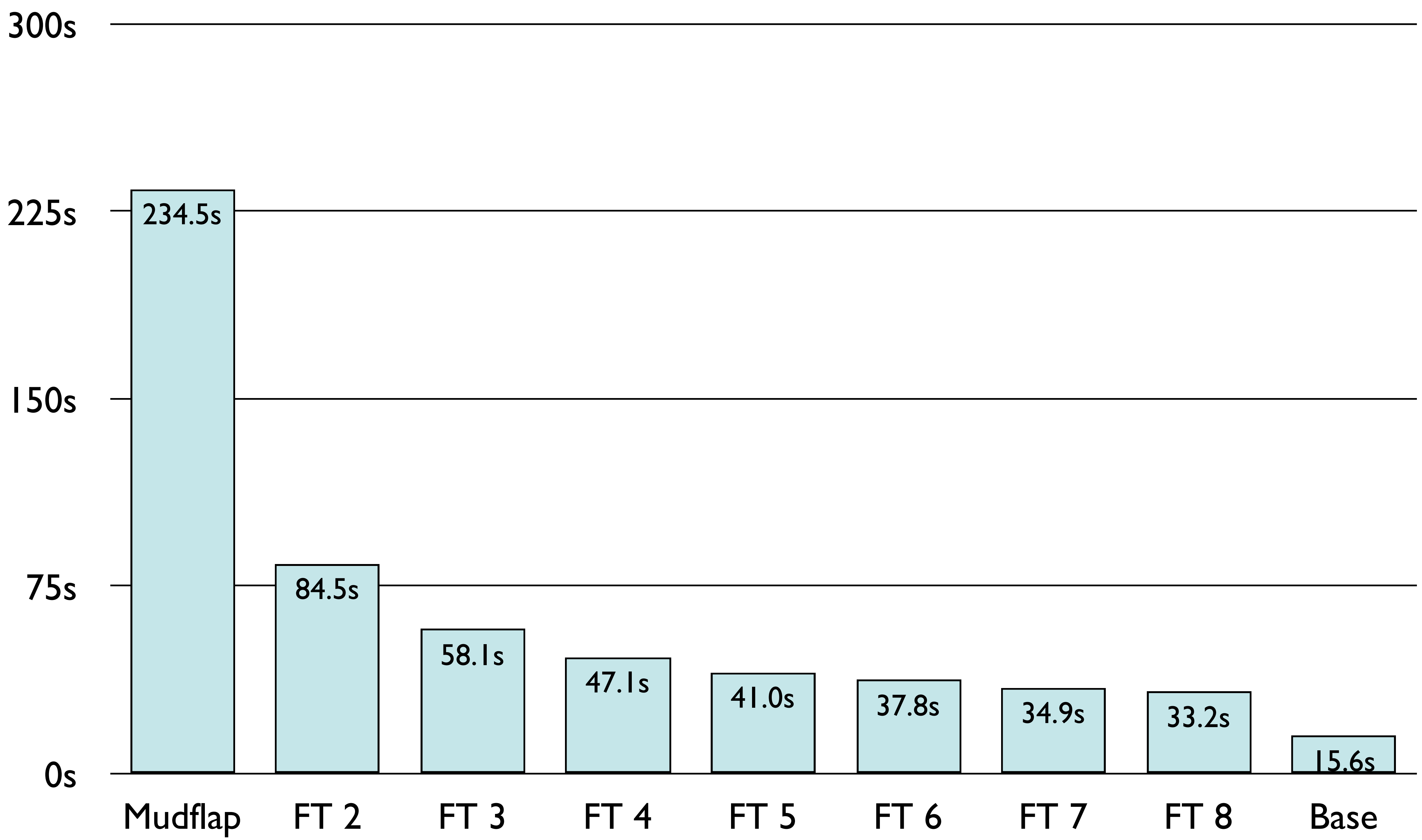
return value based on calls to fork

mudflap free version of foo() is automatically generated

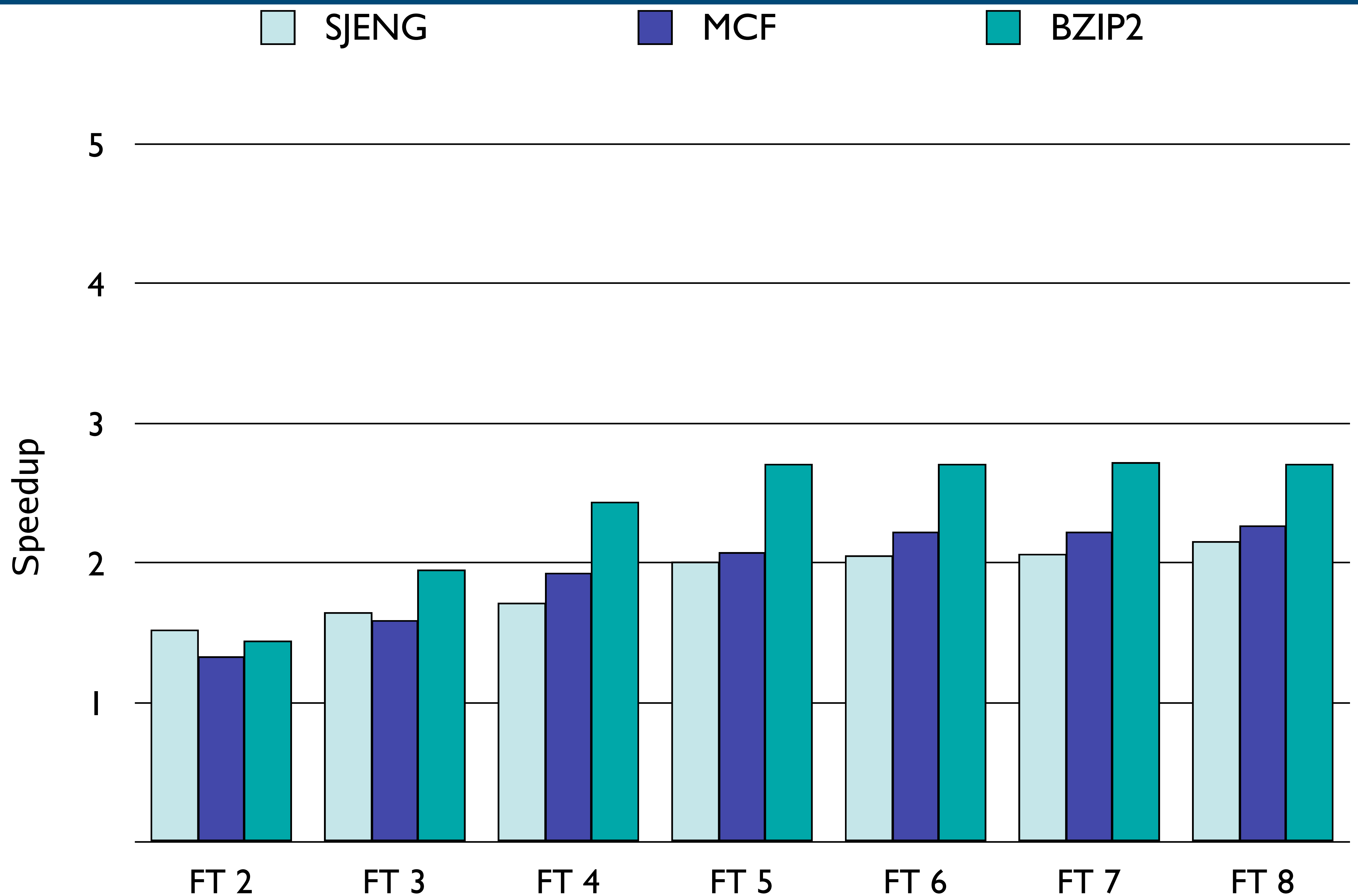
original function foo()

setup asynchronous memory checking

Results: hmmer execution times



Results: Speedup Over Mudflap



Conclusion

- Software speculation can reduce the overhead of correctness checking
- The same technique can ensure the correctness of unsafe optimizations, or be used to select amongst different heuristic approaches.
- The Fast Track system is a working example of such a technique

Thank You

Kirk Kelsey & Chen Ding
University of Rochester