Enforcing Forward-Edge Control-Flow Integrity in GCC & LLVM

Caroline Tice
Google, Inc.

Tom Roeder
Google Inc.

Peter Collingbourne
Google Inc.

Stephen Checkoway
Johns Hopkins University

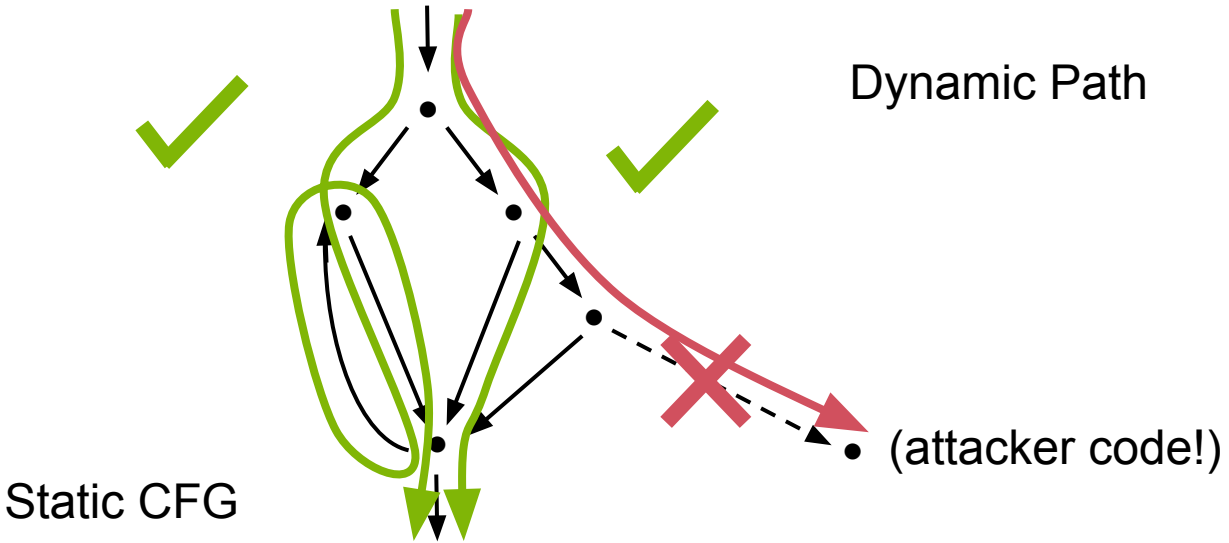
Úlfar Erlingsson
Google, Inc.

Luis Lozano
Google Inc.

Geoff Pike
Google Inc.

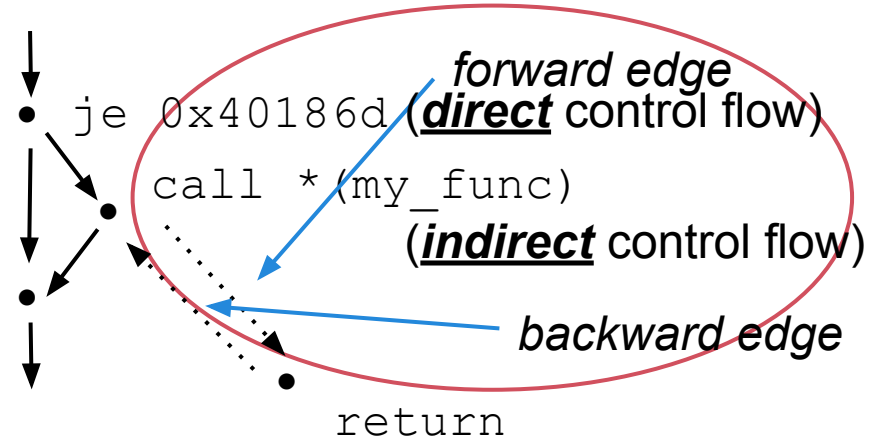
What is Control-Flow Integrity?

Control Flow Integrity (CFI) is a security policy that dictates that the software execution must follow the path of a previously determined control flow graph (CFG).



How can branch targets be attacked?

- Targets are hard-coded (in non-writable memory)...
- ...except when on the heap…
- ...or on the stack.



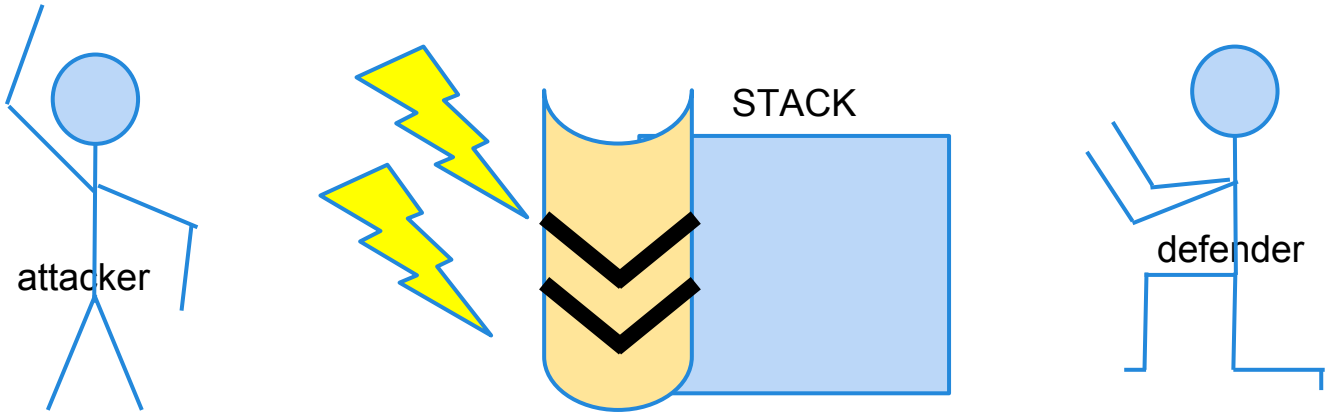
Heap

```
my_func: 0x123
```

Stack

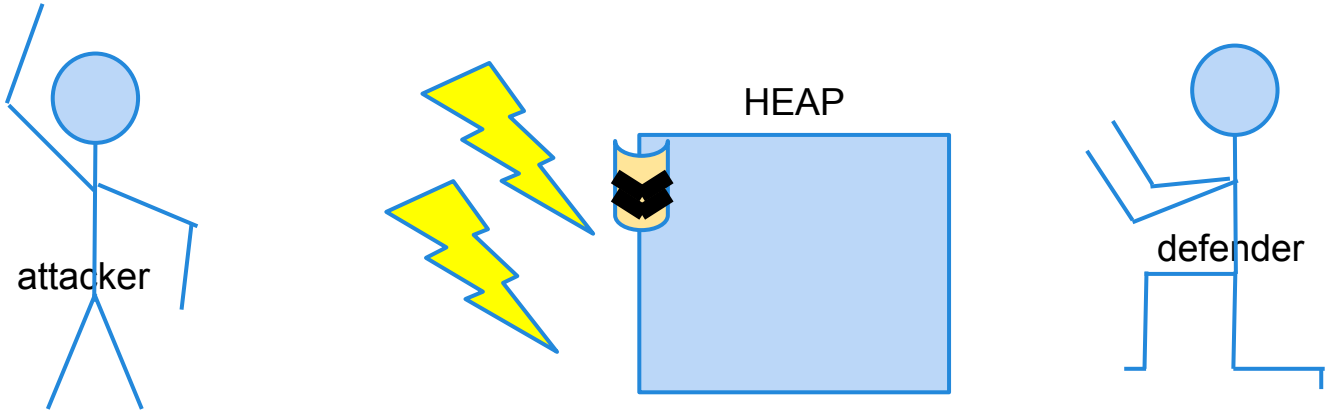
```
<ret addr>:
0xabc
```

So...why focus on *forward-edge*?



Attacks	Defenses
Buffer Overflow	Stack Canaries, Layout Reordering
Return-Oriented Programming	Address-Space Layout Randomization (ASLR)
Return-to-libc	Coding patterns that avoid stack buffers

Status of forward-edge defenses?



Attacks	Defenses
Buffer overflow / fake vtables	Heap-Metadata Canaries
Exploit use-after-free bugs	Address-Space Layout Randomization
Heap spraying / feng shui	Various ad hoc CFI attempts

Our Overall Contribution

Practical CFI enforcement in production compilers,
for forward edges.

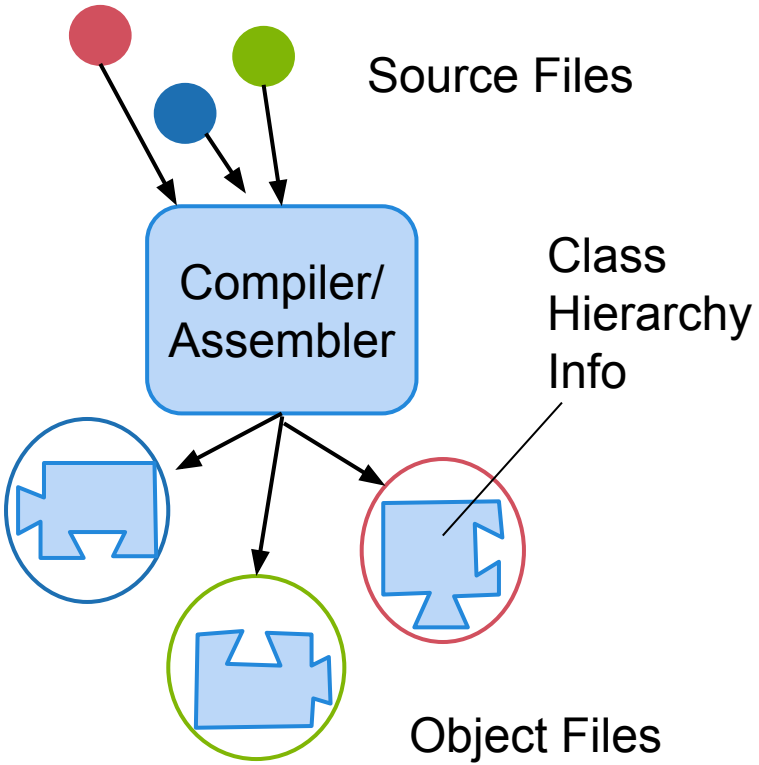
What do we have to offer?

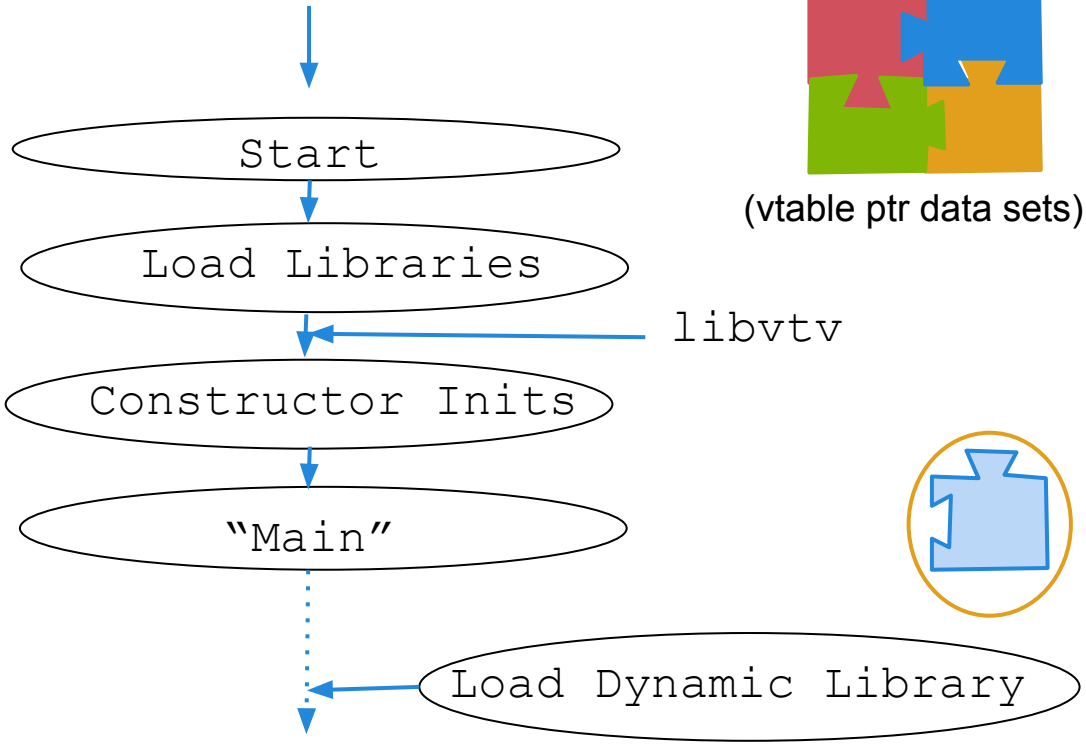
- Integrated forward-edge CFI into GCC & LLVM.
- No restrictions or simplifying assumptions.
- Scales completely.
- Strong security guarantees.
- Low performance degradation.

What *exactly* did we do?

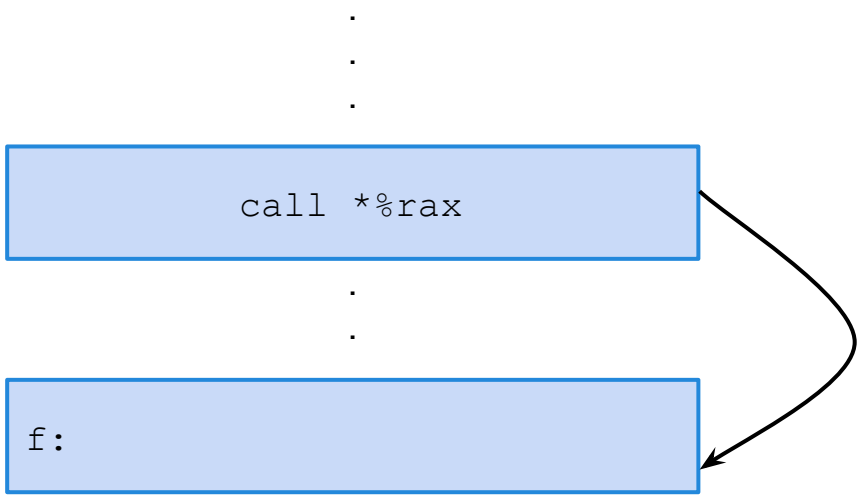
- Vtable Verification (VTV), in GCC 4.9
- Indirect Function Call Checker (IFCC), in LLVM
- Indirect Function Call Sanitizer (FSan), in LLVM

VTV Pointer Data Set details

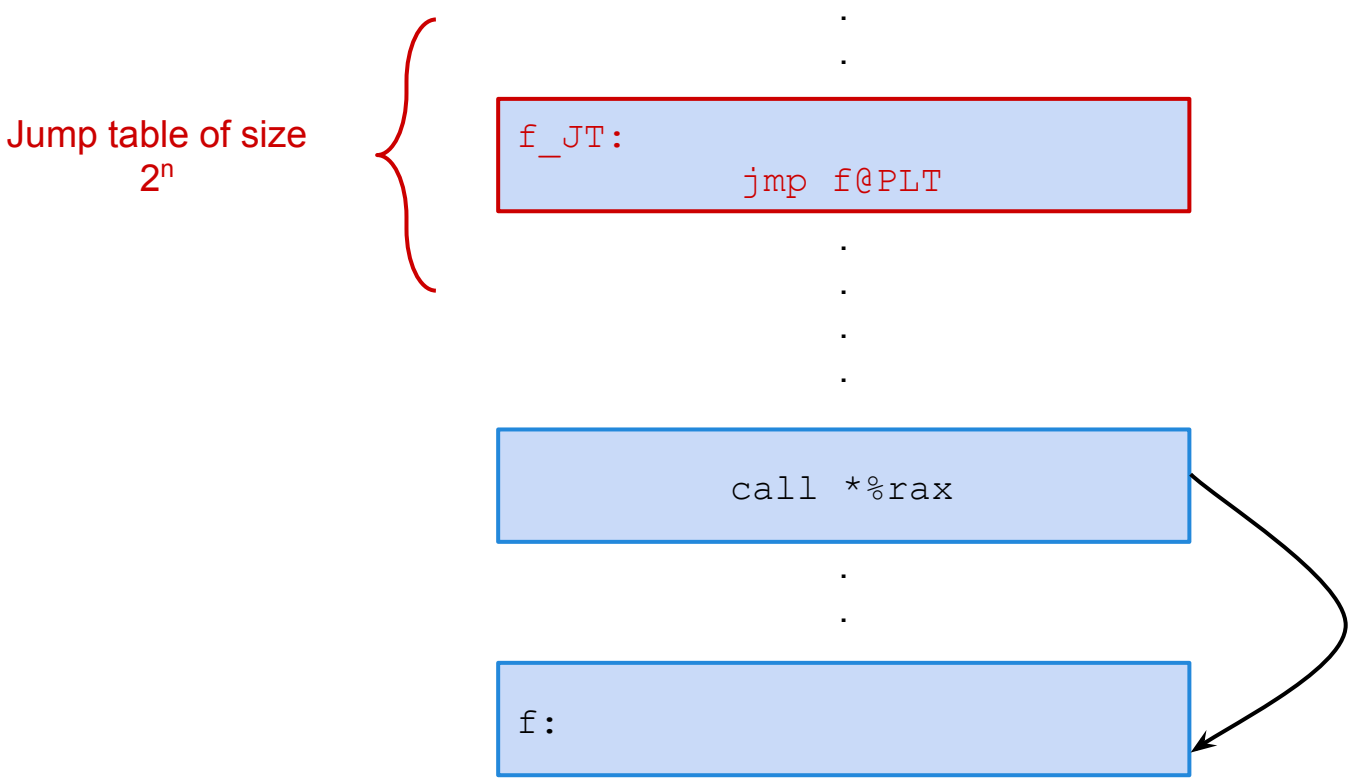




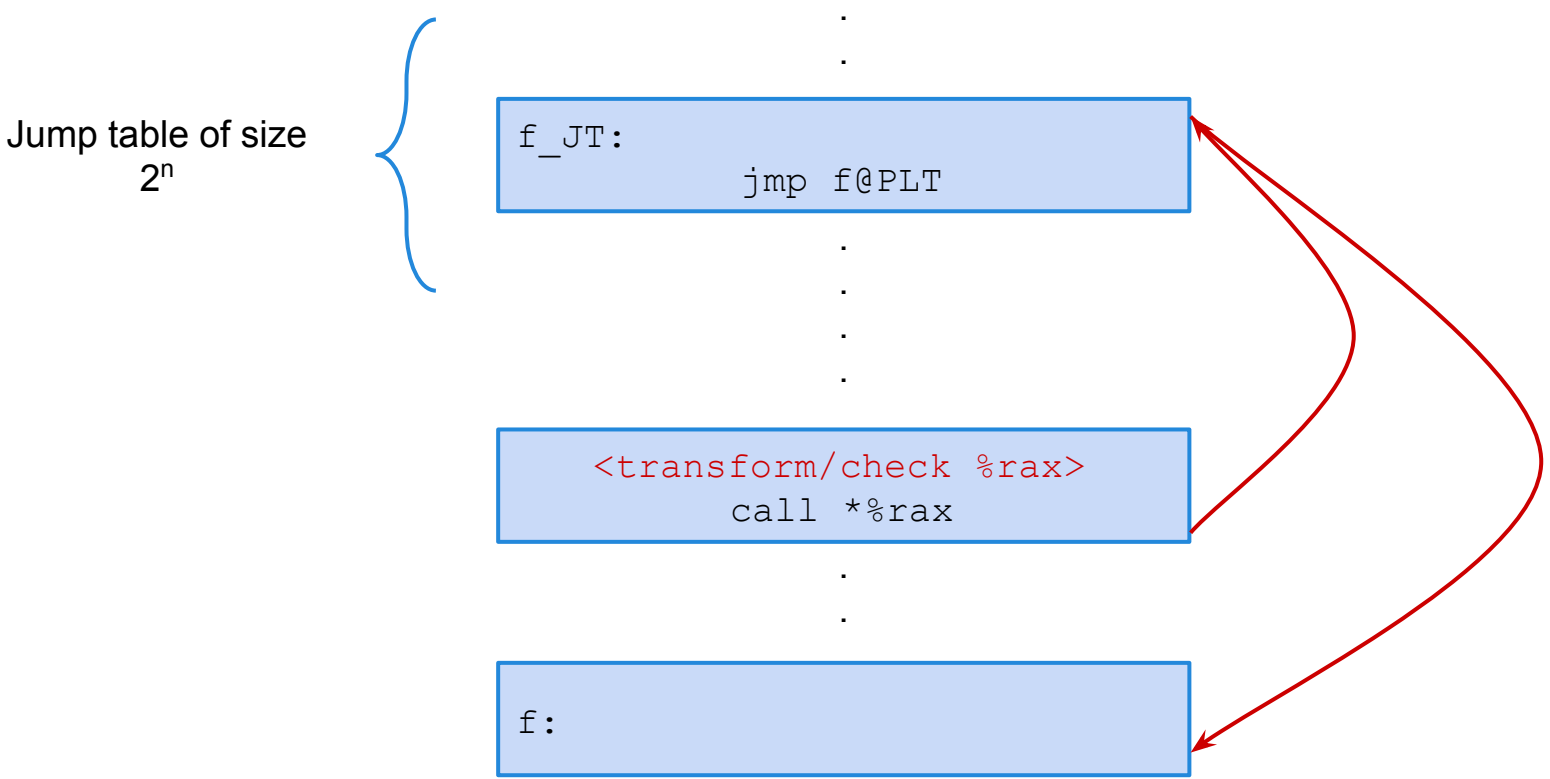
IFCC: Function Verification Details



IFCC: Function Verification Details



IFCC: Function Verification Details



Measurements: What & How?

- Security
  - AIR = Average Indirect-target Reduction [Zhang '13]
    - ~ fraction of protected indirect control flow insns.
    - *f*AIR = "forward-edge AIR"

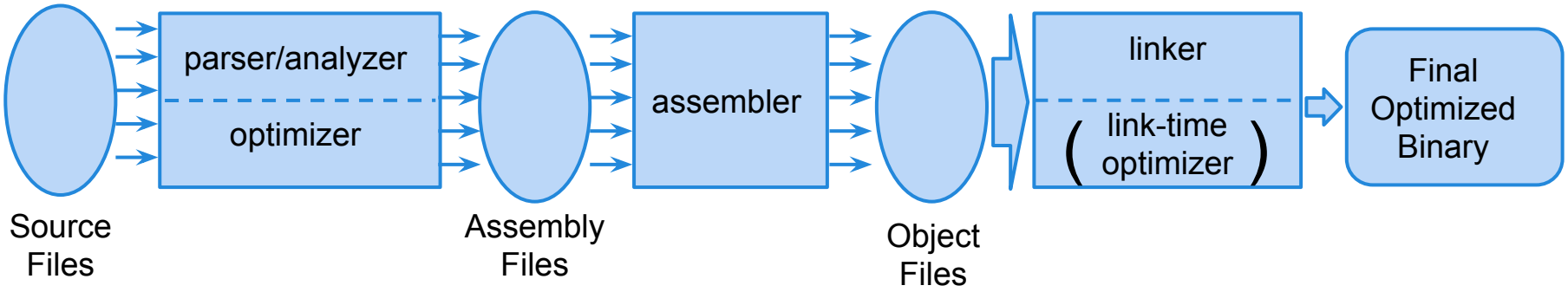
- Run-time performance degradation
  - SPEC 2006 C++ benchmarks
  - Chromium web browser
    - dromaeo, sunspider, octane benchmarks

How do they compare?

	VTV	IFCC	FSan
Security	*f*AIR = 95.2%	*f*AIR = 99.8%	NA
Performance Penalty	1.6% - 8.7%	1.9% - 3.4% (0.6% - 5.8%)	2.2% - 9.1%
Precision guarantees	Target is in a *correct* class for call site.	Target is a function (of correct arity) in original program.	Target has correct signature.
Applicability	- C++ only - Virtual calls only	- Any LLVM lang. - All indirect calls	- C++ only - All indirect calls

What have we learned?

- Fully integrating CFI w/compiler helps with performance.
- Do security analysis on *final* compiler output.
  - Some optimizations could affect security passes.
- Incremental compilation: Incomplete data => false +'s (execution aborts -- MUST AVOID!!).



What have we learned (cont.)?

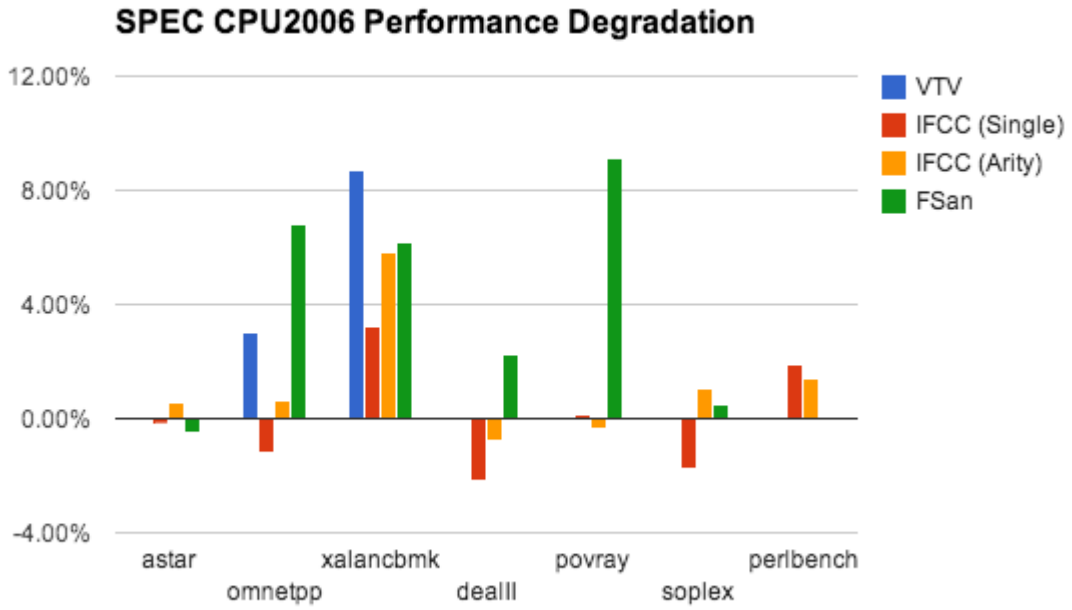
- Need to support dynamic library loading.
  - Purely statically linked binaries are hard to find!
- Mixed verified & unverified code (e.g. libraries):
  - Another source of incomplete information!
  - Requires ability for programmer intervention:
    - IFCC: Allows explicitly disabling verification for specified functions.
    - VTV: Allows modifying the failure function, e.g. w/whitelist or secondary verification.

Questions & Answers...

Practical CFI enforcement in production compilers for forward edges.

Back up slides come next.

Performance - SPEC CPU2006



Performance - Chromium