- **George C. Necula and Peter Lee, Safe Kernel Extensions Without Run-Time Checking, OSDI '96**
  - **SIGOPS Hall of fame citation:** This paper introduced the notion of proof carrying code (PCC) and showed how it could be used for ensuring safe execution by kernel extensions without incurring run-time overhead. PCC turns out to be a general approach for relocating trust in a system; trust is gained in a component by trusting a proof checker (and using it to check a proof the component behaves as expected) rather than trusting the component per se. PCC has become one of the cornerstones of language-based security.

Recap

- Monolithic vs microkernel
- Threads vs events
- Extensibility: exokernel vs SPIN
- Continuations, scheduler activations
- Resource containers

- Today's paper addresses similar problem as SPIN

PCC

- What problem are we addressing?
  - Extending functionality by moving functionality into the kernel (monolithic) similar to SPIN
  - SPIN achieves safety by using Modula 3
  - Necula achieves safety using proof carrying code (PCC)

1. Create a security policy and give it to application & kernel – policy really describes the secure hardware functionality desired
   - Actual hardware can perform unsafe actions
2. Take the user library, compute the safety proof (using security policy) and send both to the kernel

3. Kernel computes the safety predicate using VC rules. Check the safety predicate against the safety proof. If the checker finishes, then the code is safe (for the safety policy) and so use it without any further checks
   - Using first order logic stuff, they change the proof to typechecking which makes it computationally simple and efficient (1.4 ms). One can write your own type-checker rather than using public library if worried about security
4. If code is modified, safety predicate will not match safety proof. Code modifications which still creates same safety predicate are allowed

5. If proof is modified, either it will be invalid or not correspond to safety predicate

- Signed code proves the trusted origin but not the code itself
- Java type checking – performance problems
-
- Problem with approach: Proofs can be exponentially long
  - Use loops at certain spots can help