The simplification principle

- The heart of any efficient implementation of a constraint language is the simplification principle:

> Simplify each occurrence of a constraint as much as possible

- This is done at compile-time and run-time (statically and dynamically) and by looking at various-sized pieces of program (locally and globally).

- Examples of simplification:

Static:	"get_variable" in the WAM is a move
Dynamic:	Clause indexing
Local:	WAM unification is done per functor
Global:	The two-stream unification algorithm

- This principle has continued to hold from the early days of Prolog implementation (e.g., the DEC-10 compiler) to the present. It holds for the WAM, for native code systems, and for systems that do global analysis.

- All experimental evidence so far shows no contradiction with the hypothesis that the principle continues to hold until performance has reached that of imperative programming.