AIM Compilation Products and Technology

Compilers: Still going strong after ~50 years

Kevin Stoodley
IBM Distinguished Engineer
Toronto Laboratory
stoodley@ca.ibm.com

April 1, 2003

Outline

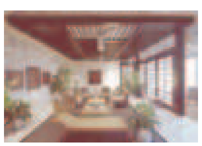
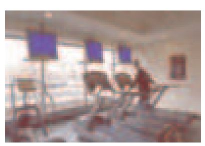
- IBM Toronto Lab stuff
- Proebsting's and Moore's laws
- Compilers then and now
- What's new and interesting (at least in my little world)
- Proebsting's law revisited (and hopefully refuted)

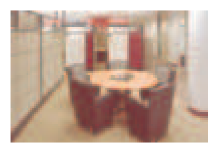
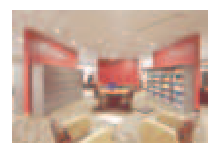
IBM Toronto Lab:

Canada's Premier Software Development Facility





IBM Toronto Lab:
Canada's Premier Software Development Facility

- Largest software development facility in Canada
- Third largest lab in the IBM Software Group
- Worldwide Missions
  - Data Management
  - Application Development Tools
  - Electronic Commerce
- Leading e-business Technologies
  - Relational Database
  - Java Development
  - Computer Language Compilers and Tools
  - WebSphere Development Tools
  - Media Design
  - Globalization

ISO 9001:2000 Certified

Compilation Products and Technology Group

- ~210 reg. staff (+ ~28 coop terms/yr) who cover development, test, support and information development
- 20 year organizational history in compiler technology
- Strong ties to IBM Research (TJ Watson, Haifa, Tokyo)
- Academic collaborations (U of A, U of T, Catalunya, Illinois)
- Designated core competency site in IBM for this technology
- Product missions:
  - C/C++ p-Series running AIX and Linux
  - C/C++ on z- and i-Series running zOS and OS/400
  - Fortran 95+ on p-Series running AIX and Linux
- Component missions:
  - Java JIT compilers for servers (PowerPC, X86, IA64, 390)
  - Java JIT compilers for embedded (PowerPC, X86, ARM, MIPS, SH4)
  - XML parsers (numerous platforms)
  - XSLT processors (numerous platforms)
- Miscellaneous
  - pretty much any IBM compiler that needs an optimizing backend for a platform already supported (Cobol, PL/I, Pascal (!), etc)
- Secret stuff (but nothing that would impress my kids)

Moore's Law

- "CPU performance doubles roughly every 18 months"
- Diverse sources of improvement lead to this simply expressed but remarkably long-standing and robust "result":
  - Fabrication/process
    - directly and indirectly, this is the "biggee"
  - Low level circuit design
  - High level design
  - Macro architecture (memory nest, interconnect, etc)
  - Micro architecture (out-of-order, superscalar, etc)
  - Instruction Set Architecture (ISA)
    - despite the "press", this is perhaps the weakest lever
    - it's certainly the most controversial and prone to zealotry
  - Compiler technology improvements

Proebsting's Law (you can look this up in google)

- Compiler optimization R&D has led to a four-fold performance improvement over the last 36 years
- Roughly this is 4% a year (not quite as good as 60)
- Based on the (generous) observation of the ratio between the performance of an unoptimized (CPU bound) program and the same program compiled with full optimization
- Implication, according to Proebsting, is that we optimizing compiler researchers should turn our talents to more fruitful endeavours

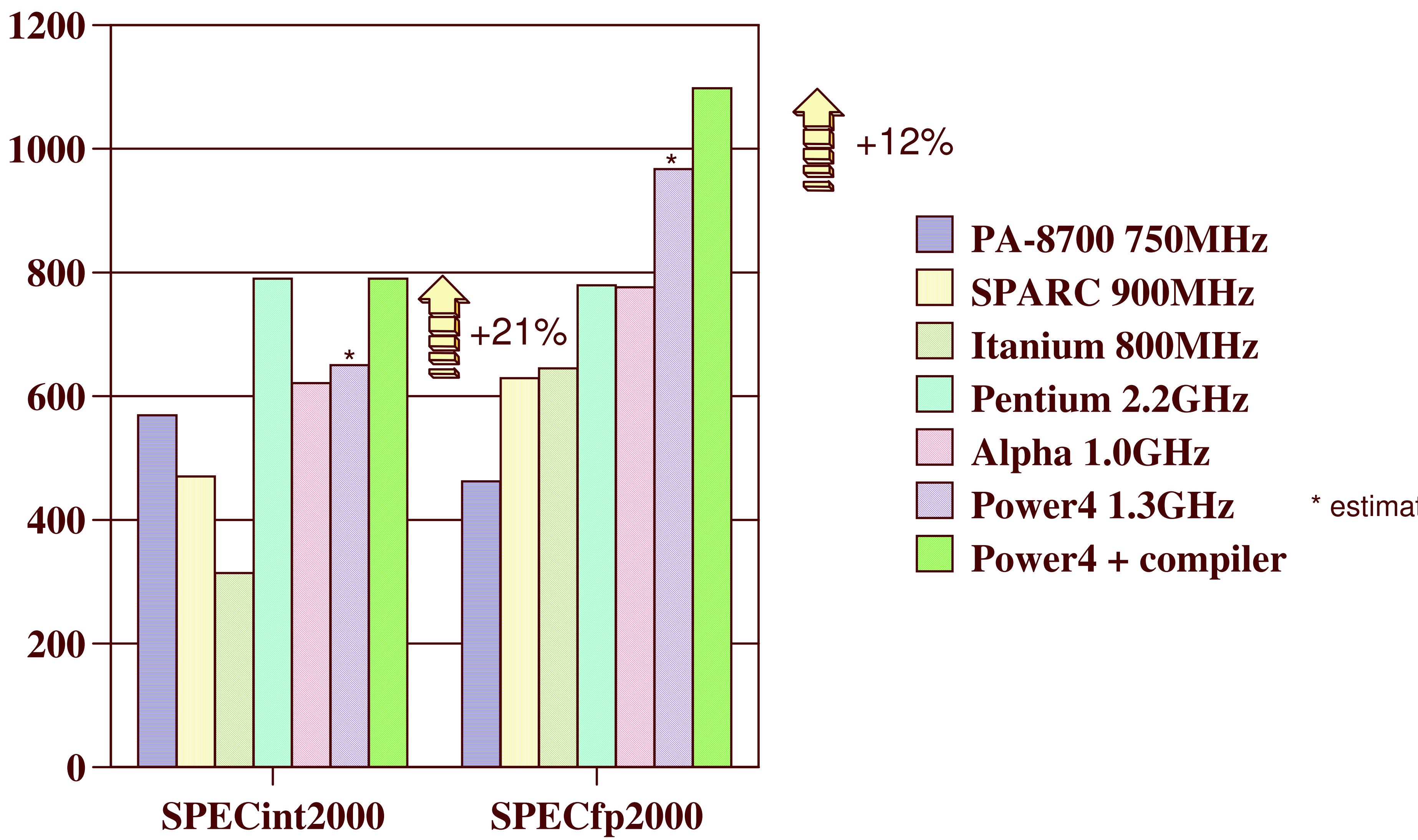
So, I didn't change jobs. Why not?

- Well, 4X is actually pretty good and it's what we signed up for
  - Compilers don't directly get to manipulate the biggest levers (process, circuit design, logic design)
  - Compiler's major levers
    - ISA
    - Micro-architecture
    - Memory nest (cache geometry, latency, etc)
- 4X is very good bang for the buck
  - semiconductor, CPU architecture and system design R&D spending dramatically outstrip compiler optimization R&D spending
- Those CPU people keep stealing/obviating compiler optimizer ideas and putting them into hardware
  - out-of-order and speculative execution (twice now)
  - instruction trace caches
- Important class of programs does get more than 4X
  - Mostly scientific code

4X is a point in time, but the ground isn't stable

- Compiler has to keep re-winning that 4X improvement
  - ISA changes
    - CISC, RISC, VLIW, EPIC
    - cool new instruction from some bright hardware person
  - Micro-architectural changes
    - in-order v out-of-order
    - speculative execution
    - branch prediction hints and caches
    - instruction grouping
  - Macro-architectural changes
    - large pages
    - pre-fetch streams
  - High level language "improvements"
    - polymorphism, inheritance, type opacity (OO)
    - dynamic typing and loading
    - lazy evaluation
    - generics/templates
    - the next big leap in costly, syntactic sugar

We have to work pretty hard just to keep from falling behind!



Attaining high performance on Power4 class processors

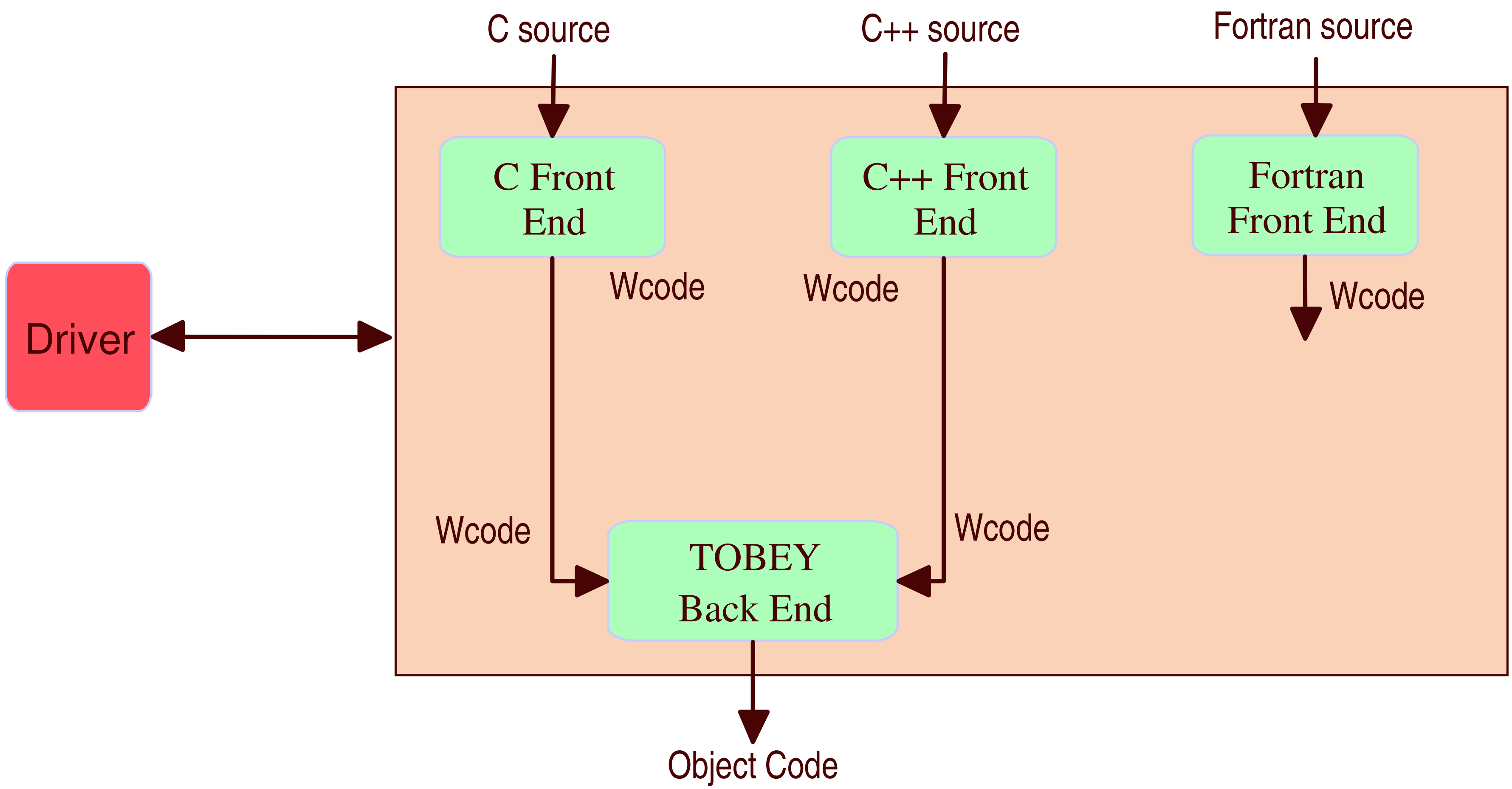
- Significant challenges to achieving correctness and high performance
  - Weakly consistent storage model
    - nagging problems with correctness due to absent or subtly incorrect synchronization
    - increased impact of synchronization sequences on overall performance even with lwsync
  - Microarchitectural changes
    - completely different modelling required for instruction scheduling
    - instruction selection needs to be tuned to avoid previously common instructions which are "cracked"
  - Miscellaneous
    - large page support
    - streams
    - branch prediction hints

Let's not even talk about IA64

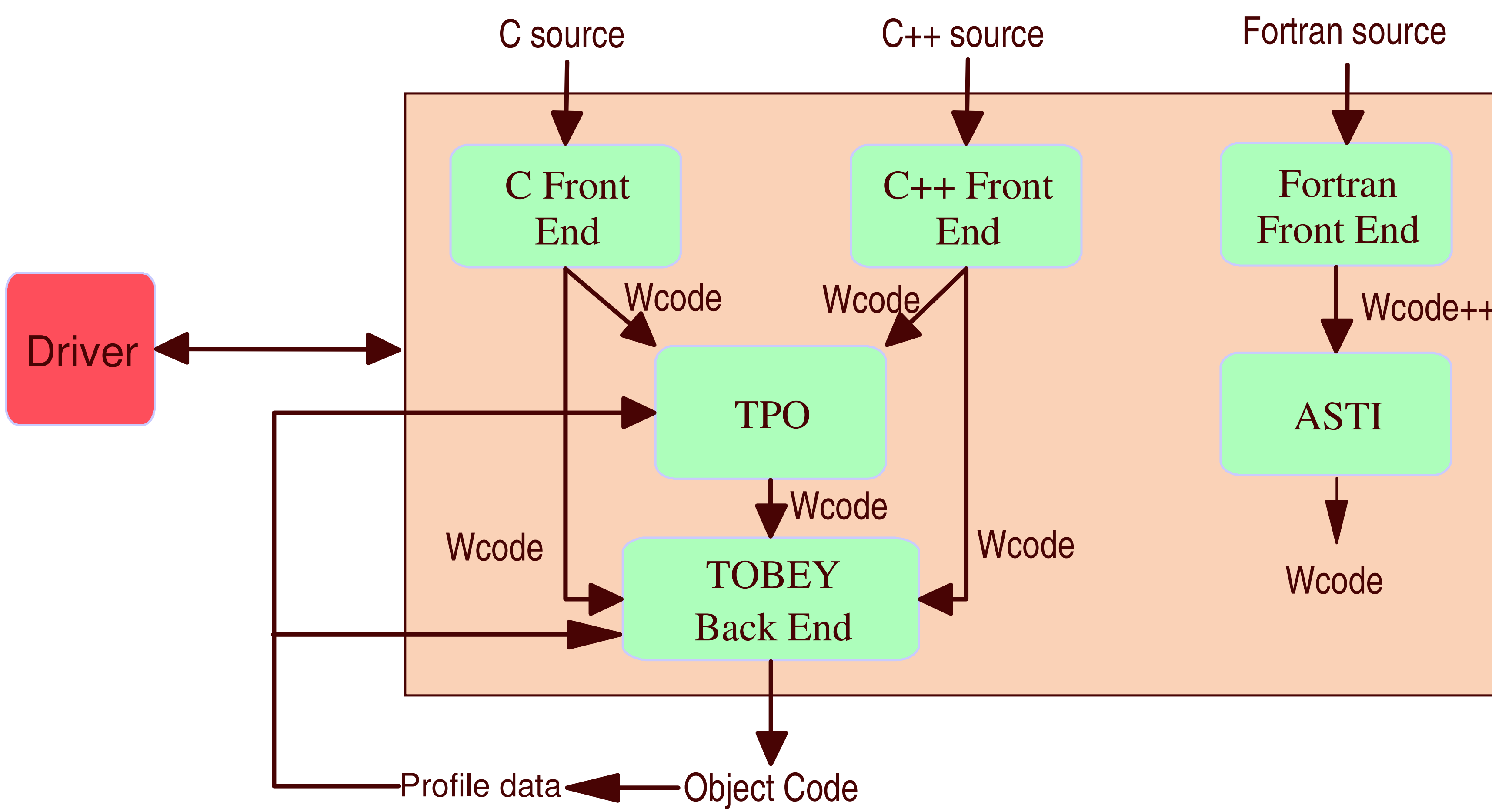
And then there's the cost...

	Cost (order of magnitude)	Cost Growth
New Silicon Process	$ 1 billion	very fast
New CPU Architecture	$ 100 million	fast
New Compiler	$ 2 million	slow

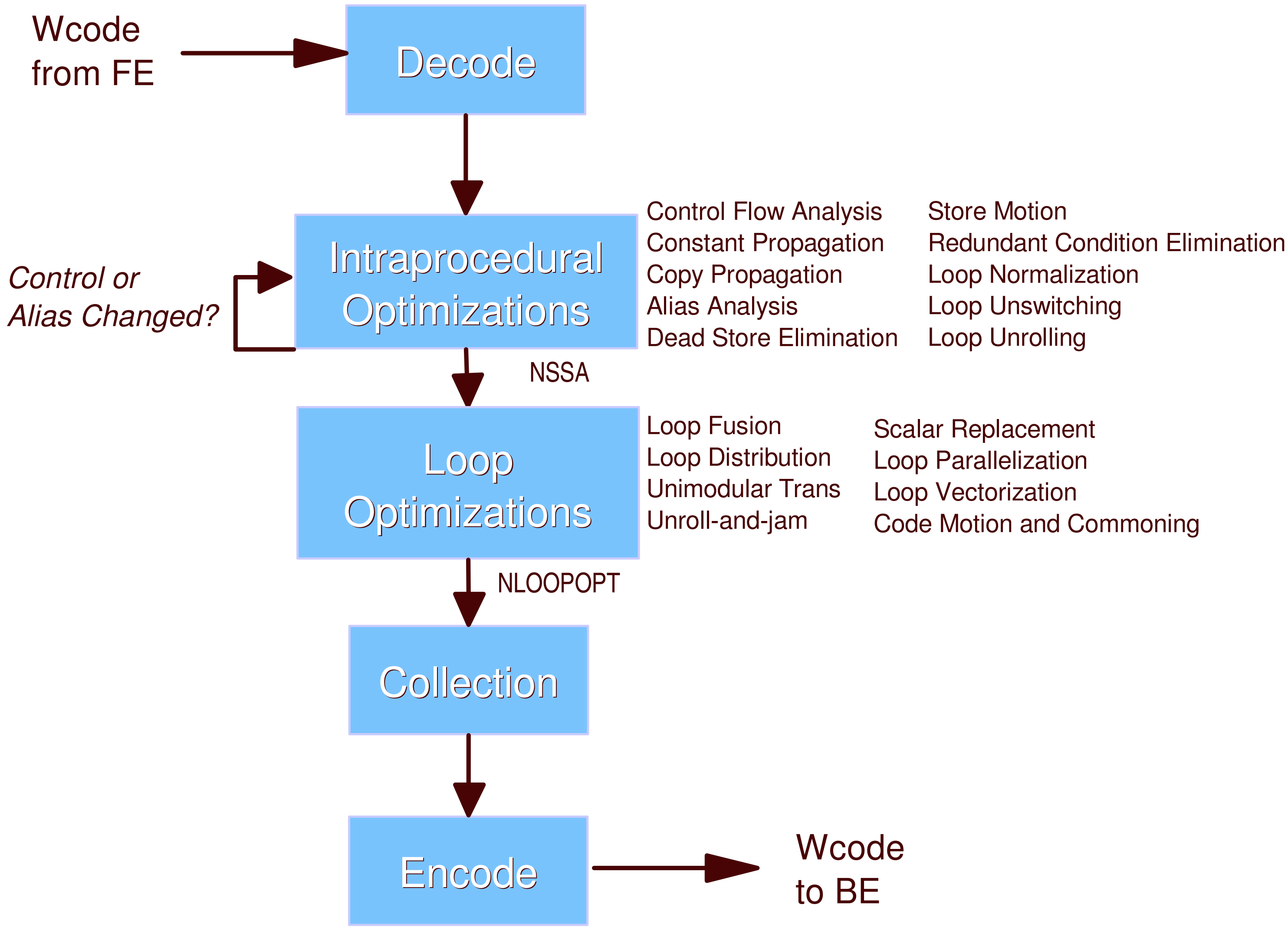
Inside a (static) Compilation (then)



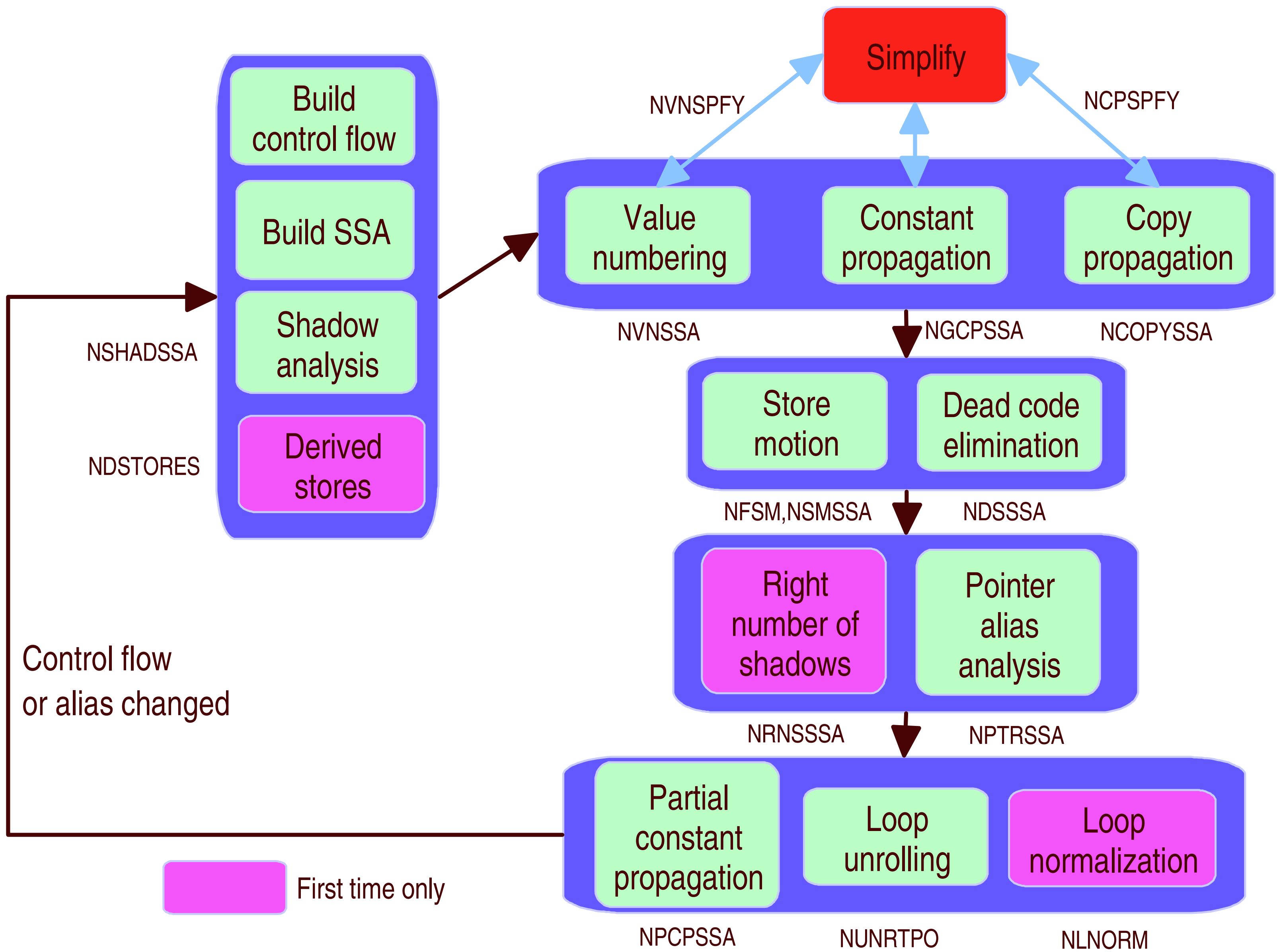
Inside a (static) Compilation (now)



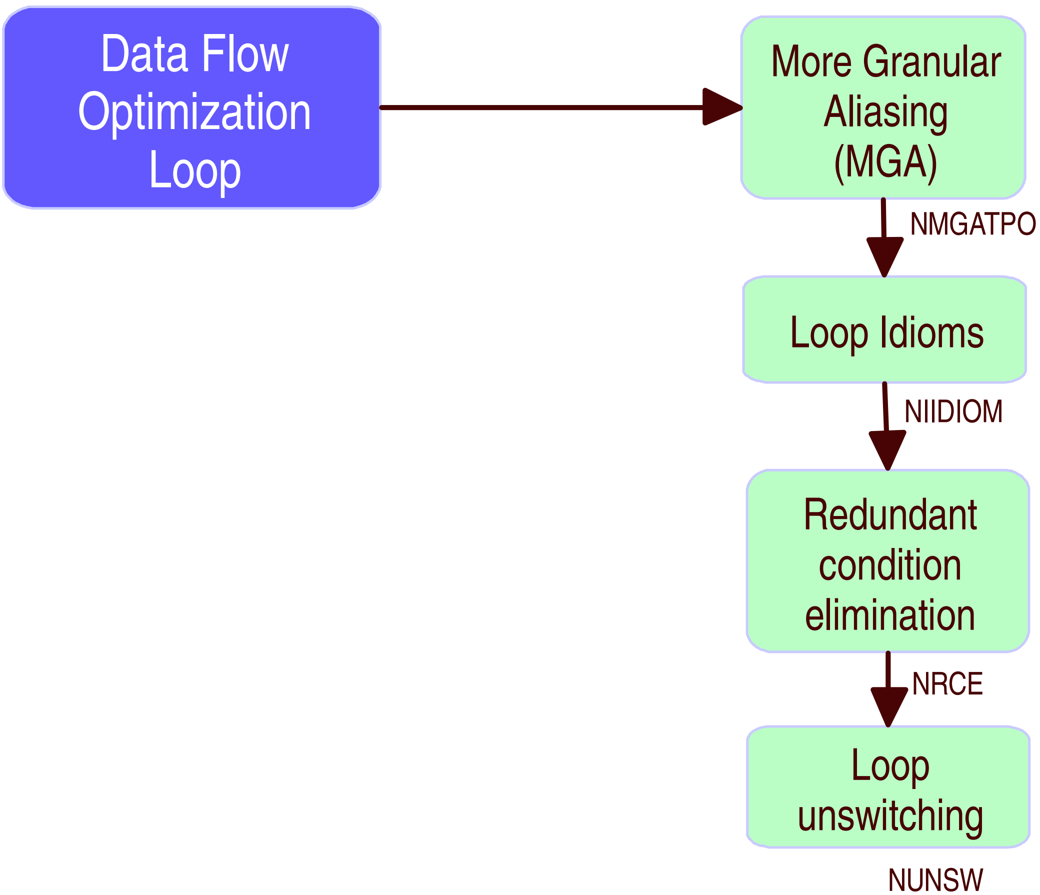
Inside TPO Compile Time Optimization



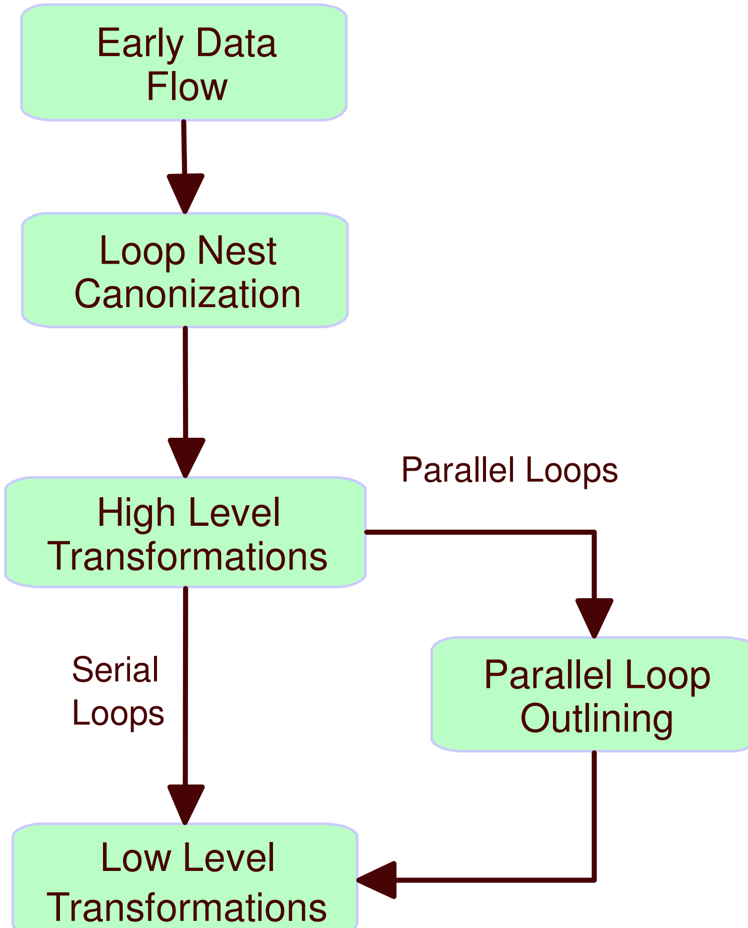
Data Flow Optimization Loop



Late Data Flow Optimizations



Loop Optimization



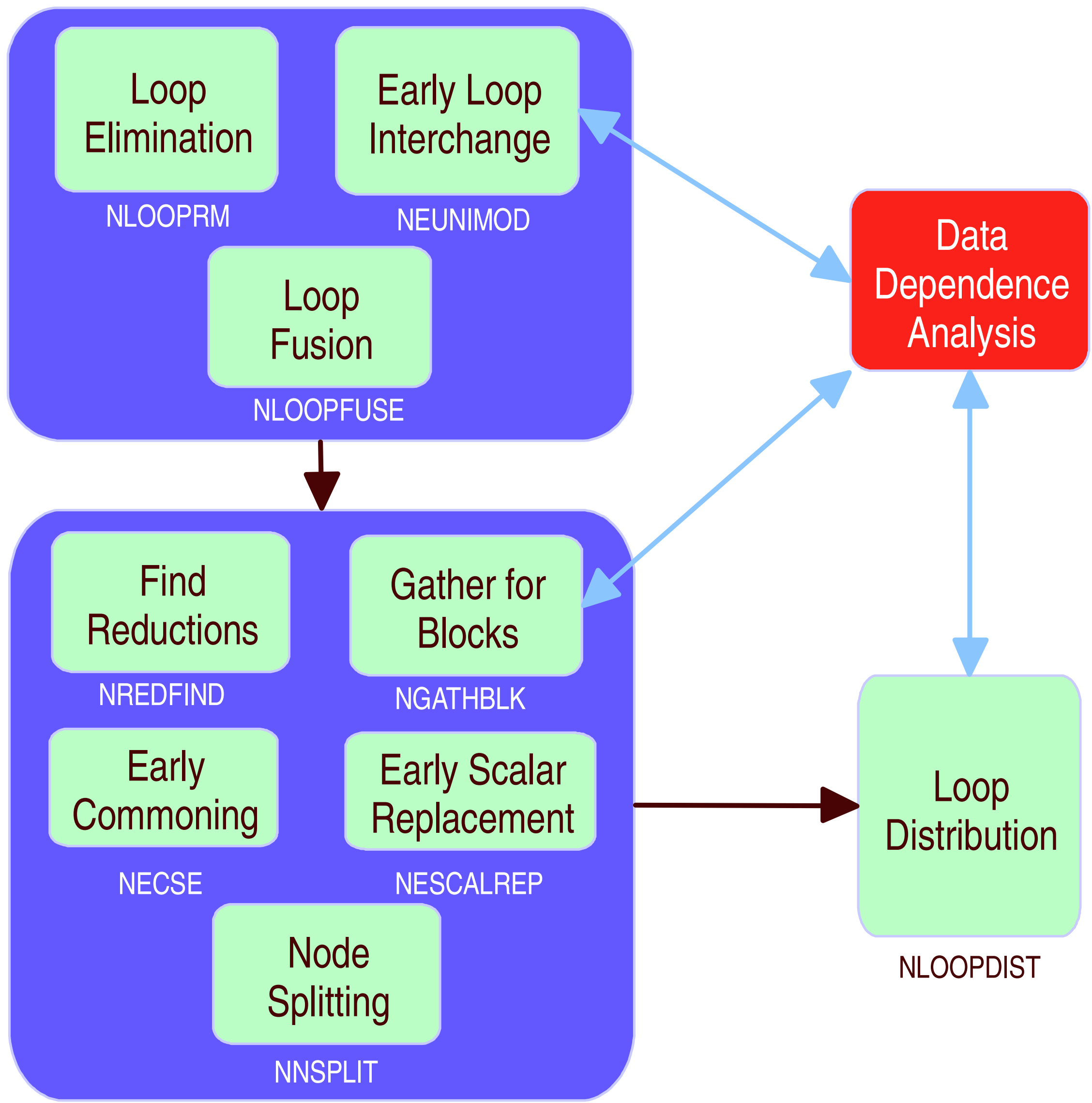
Control Flow Optimization
Data Flow Optimization
Loop Normalization

Aggressive Copy Propagation
Maximal Loop Fusion

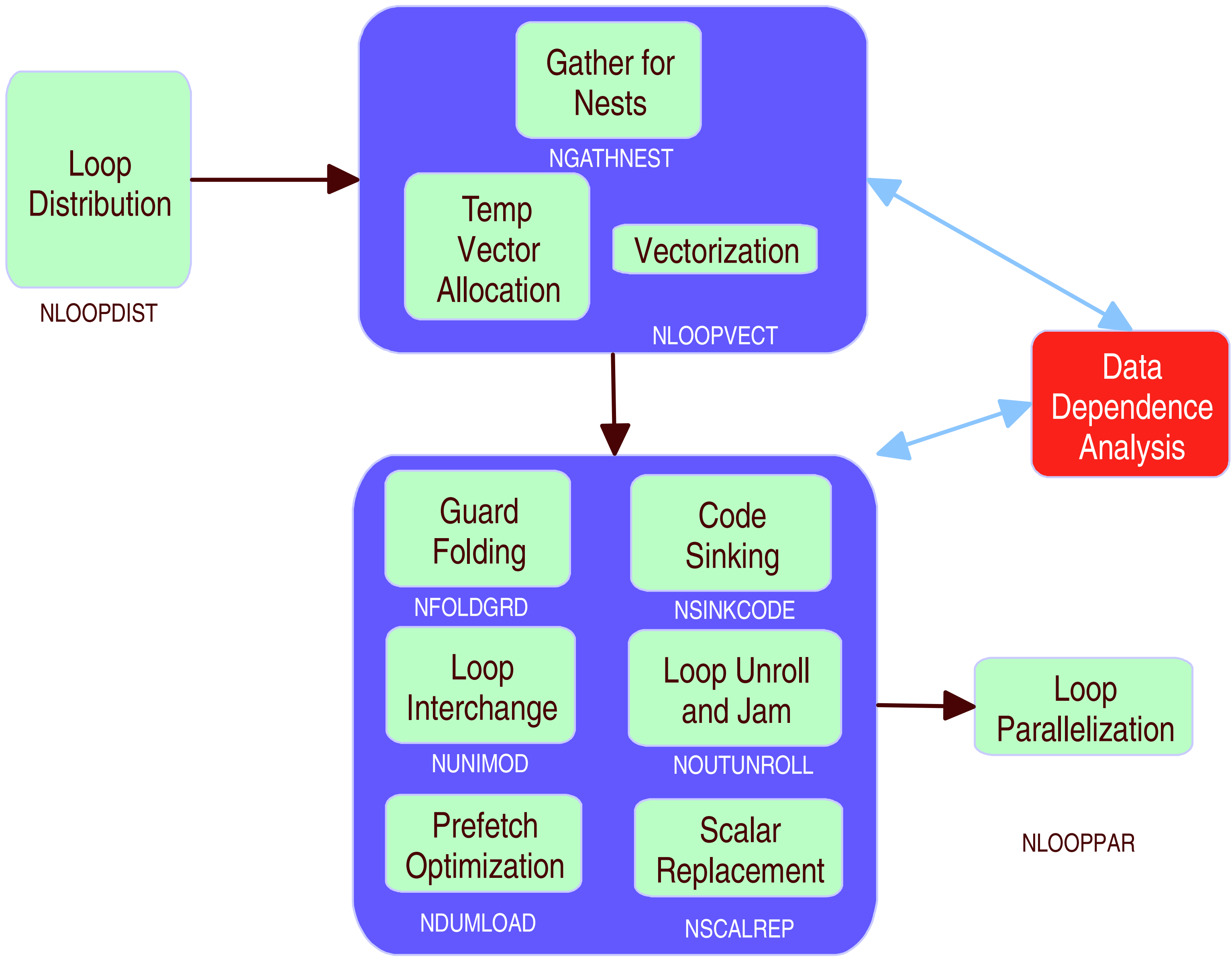
Loop Nest Partitioning
Loop Interchange
Loop Unroll and Jam
Loop Parallelization

Inner Loop Unrolling
Loop Vectorization
Strength Reduction
Redundancy Elimination
Code Motion

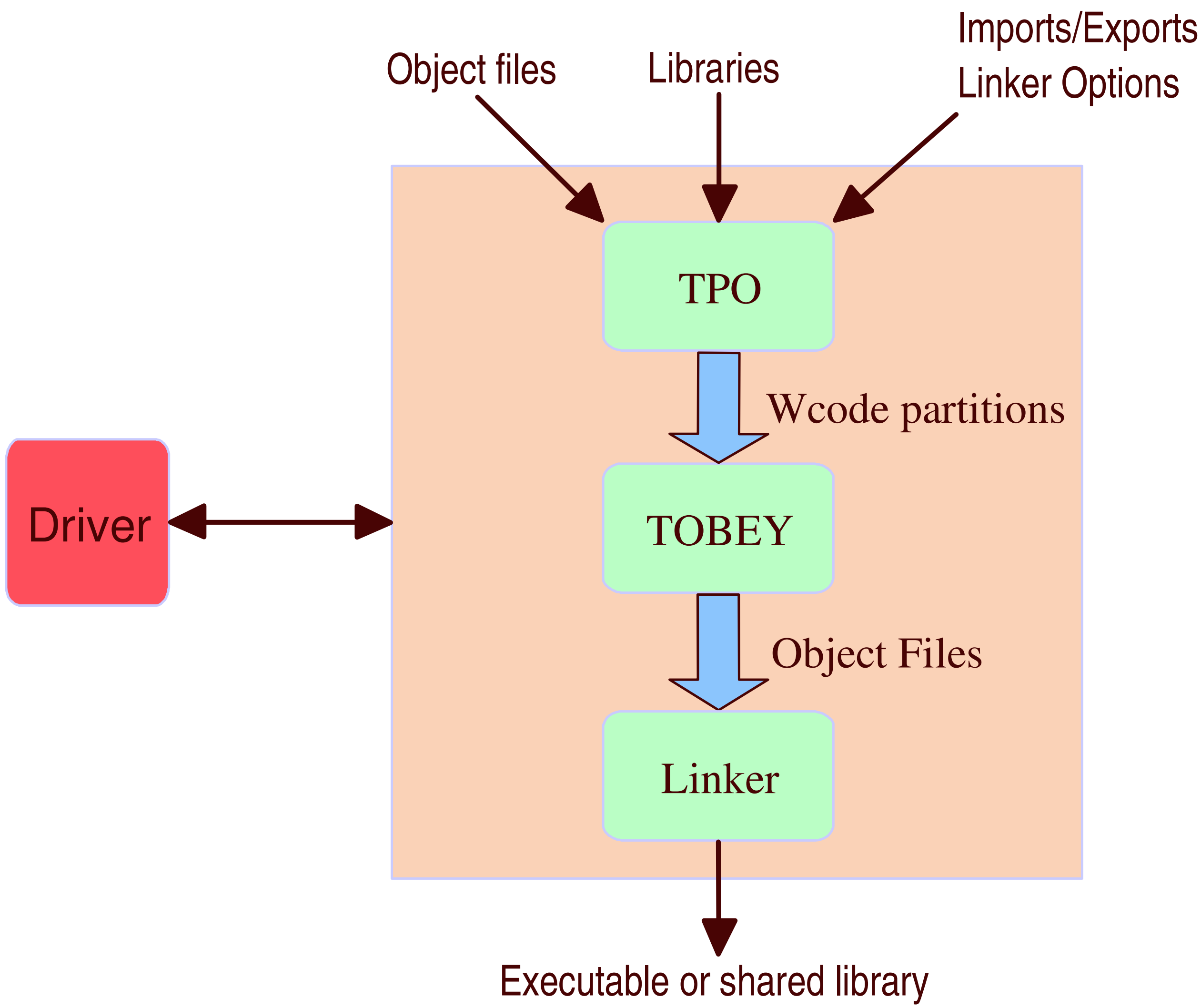
Loop Nest Canonization and Distribution



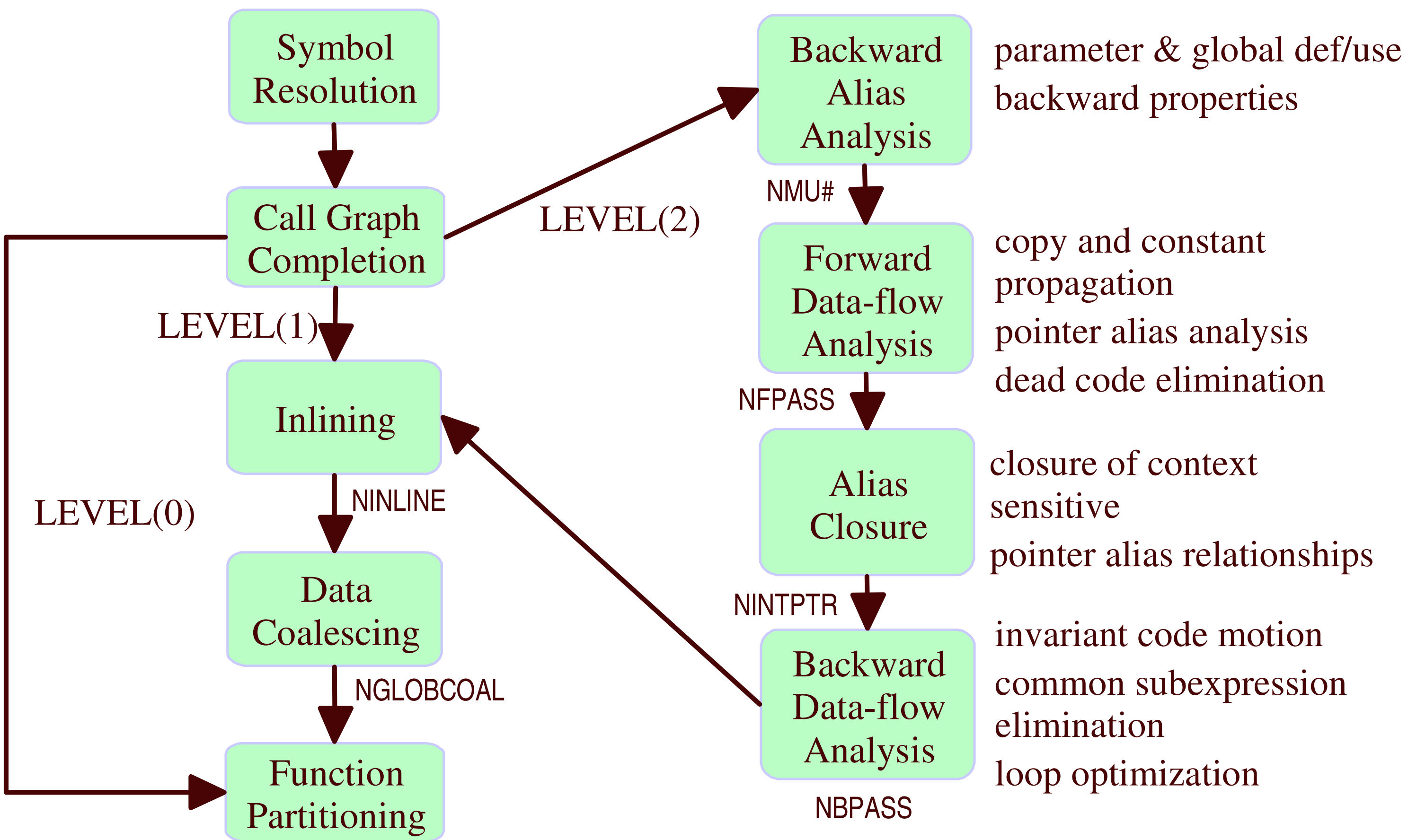
Loop Nest Optimization (pre-Parallelization)



Inside an Link-time Compilation



Inside TPO Link Time Optimization



What's new and interesting in compilers?

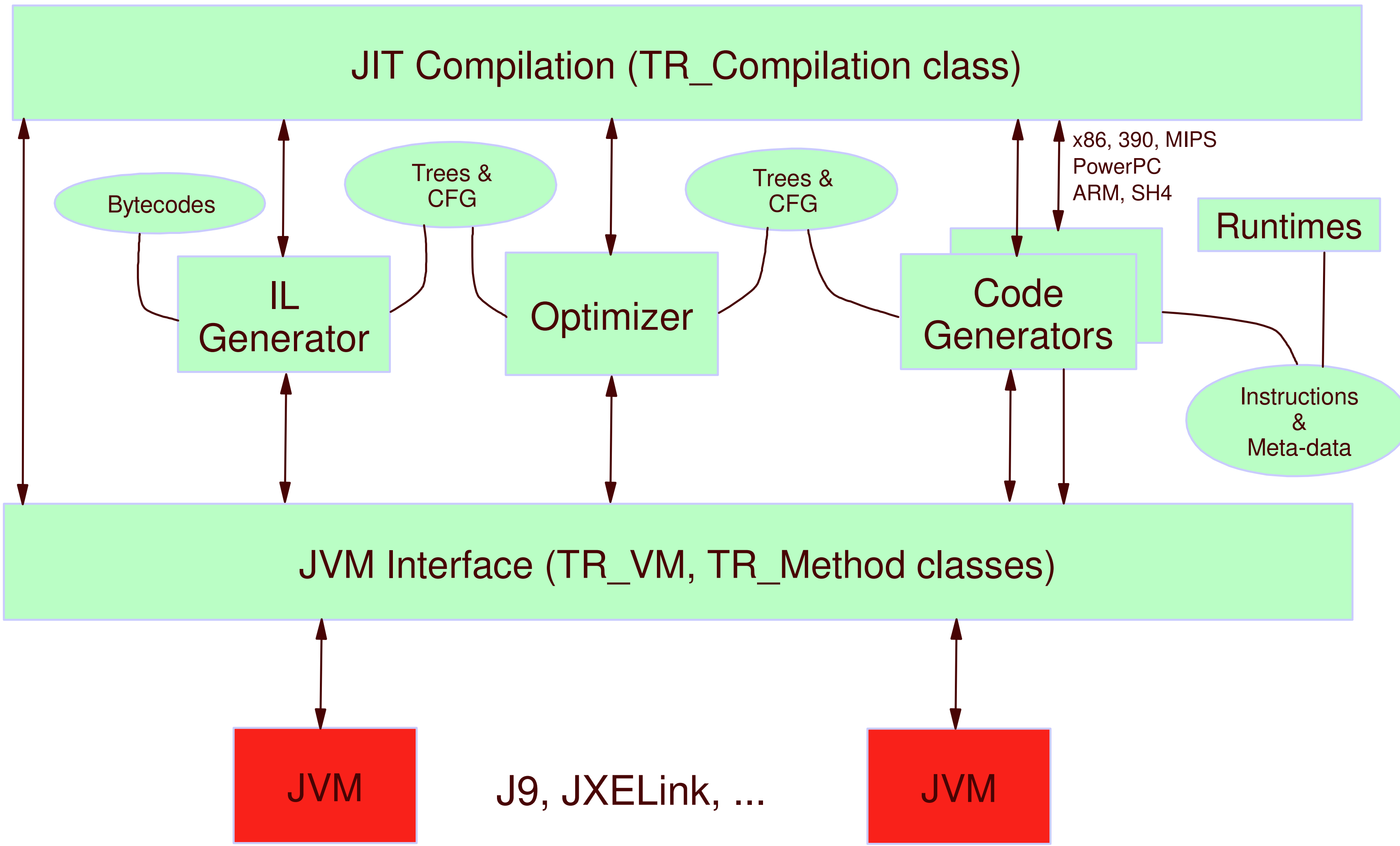
- A shift in mainline use of compilers
  - Traditionally, compilers run by programmers
  - Increasingly, compilers run by users (although they ideally don't know it)
  - Dynamic optimizing compilers:
    - Java JITs
    - CLR JIT
  - Dynamic compiler compilers
    - XML parsers
    - XSLT processors
  - Compiler-centric web application development model
    - Java Server Pages (JSP)

Testarossa JIT Technology

Testarossa Design Goals

- Clean separation of concerns along 3 major axes
  - JVM implementation (VM and OS services)
  - Java implementation (object model, runtime specializations, GC, threads, runtime interfaces)
  - Hardware targets
- Java centric design
- Portable and maintainable C++ implementation with some special purpose assembler
- Fast compile time
- Small footprint
- Configurable optimization framework
  - extremely complete suite of classical & Java-specific optimizations
- High performance code with deep platform exploitation
- Hot Code Replace (HCR) and Full-speed Debug (FSD)
- Complete solutions: optimizing transformations fully operational in the presence of exception handling, security manager, stack trace, unresolved or volatile entities, etc
- Dynamic recompilation with profile directed optimizations
- Aggressive specialization and speculative optimizations

A peek under the hood



Diverse set of ISAs supported

- Fully supported targets
  - x86
  - 32-bit PowerPC (bi-endian)
  - ARM
  - SH4
- Under development
  - 64-bit PowerPC
  - MIPS (bi-endian)
  - 31-bit 390
  - X86-64
- In plan for 2003
  - 64-bit 390

"...Palmtops to Terraflops technology..."

Complete suite of classical and Java optimizations

- Platform neutral optimizer performs IL-IL transformations
  - parameterized by platform specific code to handle different cpu capabilities (eg. # regs)
- Multiple optimization strategies for different code quality/compile time tradeoffs
  - used to compose optimizations into a collection of transformations
  - spend compile time where it makes biggest difference
  - can also tradeoff JIT size for optimization quality
- Extremely generalized solutions and infrastructure
  - Eg. Inliner capable of functioning effectively in presence of exception handling, security manager, stack trace, etc.

Profile directed sampling recompilation

- Sampling thread drives compilation based on hotness
- Initial compilation is low-opt
- Hot methods are recompiled at increasingly higher optimization levels
- "Scorching hot" methods
  - recompiled with profiling instrumentation
    - edge counts with inferences, value profiling, virtual call sites, etc
  - run long enough to gather representative data
  - recompiled at highest opt level and exploiting profile data
    - basic block scheduling
    - inlining
    - loop versioning
    - devirtualization
    - aggressive replication
    - speculative opts

Example speculative opt driven by profile data: Escape Analysis

- Sometimes it is advantageous to split an allocation so that on one path it is local

```
o = new C;
...  // some code 'a'
if (condition){
    ... // some code 'b' (object o does not escape the threa
} else {
    x.foo(o); // cannot prove object o does not escape
}
return;
```

becomes

```
if (condition){
    o = new C;
    ... // some code 'a'
    ... // some code 'b' (object o does not escape the threa
} else {
    o = new C;
    ... // some code 'a'
    x.foo(o); // cannot prove object o does not escape
}
return;
```

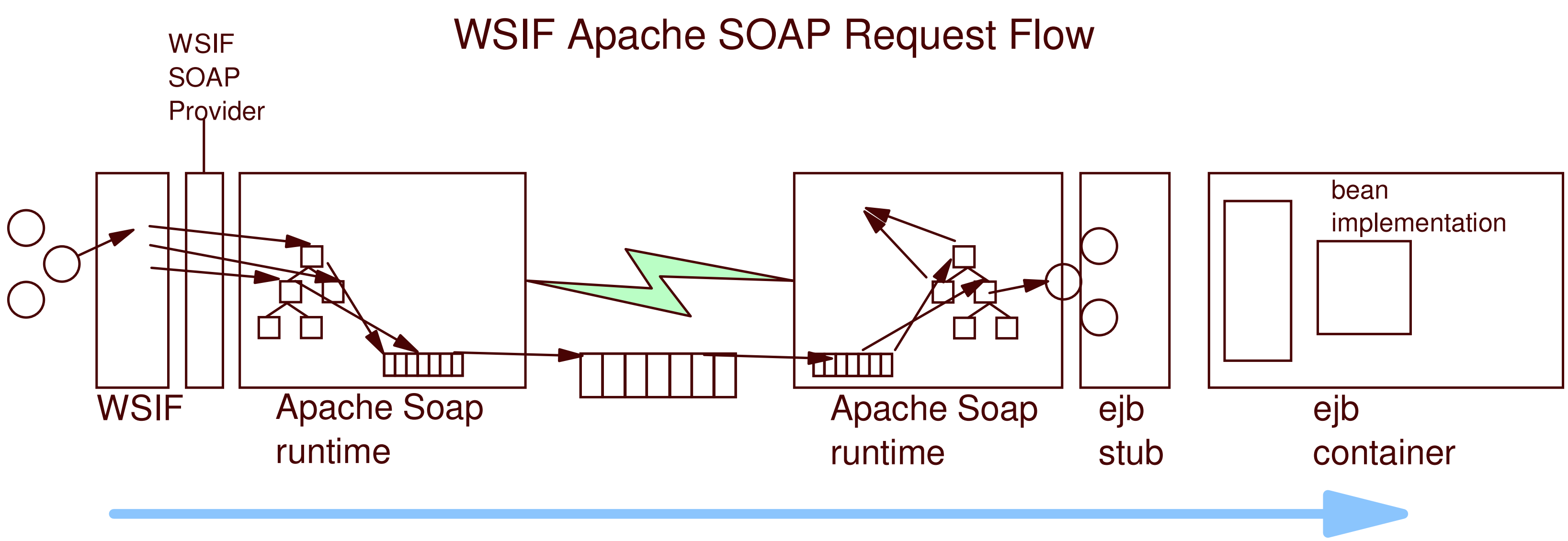
Recompilation infrastructure is basis for

- Aggressive speculative optimizations
  - pre-existence based devirtualization and inlining
  - other class hierarchy based optimizations
  - single threaded optimizations
- Hot Code Replace (HCR)
  - Fix/Enhance code while running and without restarting
- Advanced problem determination and performance monitoring features
- Phase change adaptations

XML

- Three scenarios with very different needs
  - Tools, mapping and config
  - Database (relational and native store)
  - Web services
- Schema is particularly interesting
  - Allows an XML specification of grammar for validating instances of XML documents

Example of End-to-End Web Services Flow



- **Client**
  1. Application Objects in Java
  2. Invoke over WSIF - binds to SOAP as provider
  3. Apache Soap-provider serializes App Objects into DOM using XML Parser
  4. DOM Elements are serialized to stream that is put on wire
  5. If Security is applied, the SOAP Envelope is digitally signed before being placed on wire

- **Server**
  1. Apache SOAP Run-time builds DOM from stream using XML Parser
  2. If Security is applied the Soap Envelope is validated before de-serialization
  3. Apache SOAP de-serializes DOM in Application Objects
  4. Runtime determines and invokes the target Object and operation

Back to Proebsting's Law

- Proebsting's conclusion depends on unexamined assumptions
  The contrasting reality is:
  - CPUs change constantly in myriad different ways that have challenging implications for developers of optimizing compilers
  - Programming languages and methodologies increasingly trade programmer convenience for compiler burden
  - Cost of compiler optimization R&D is dramatically lower than process, circuit and CPU R&D and growing relatively much more slowly
  - Competitive forces mean nobody in industry can afford to flinch anyway
  - Loosely coupled runtime models shift burden to optimizing compilers
- It's non-trivial to keep up, let alone make forward progress
  - Static analyses must become dynamic
  - Early, accurate predictions more important than late-breaking complete knowledge
  - New optimization goals are emerging (eg. power consumption)

The good news is...

- Compiler field is more vital than it has been in a long time
- We haven't run out of challenges or new ideas
- Industry and Academic collaborative research in compilers is increasing
- Compilers are becoming more pervasive and are no longer restricted to use by the development community