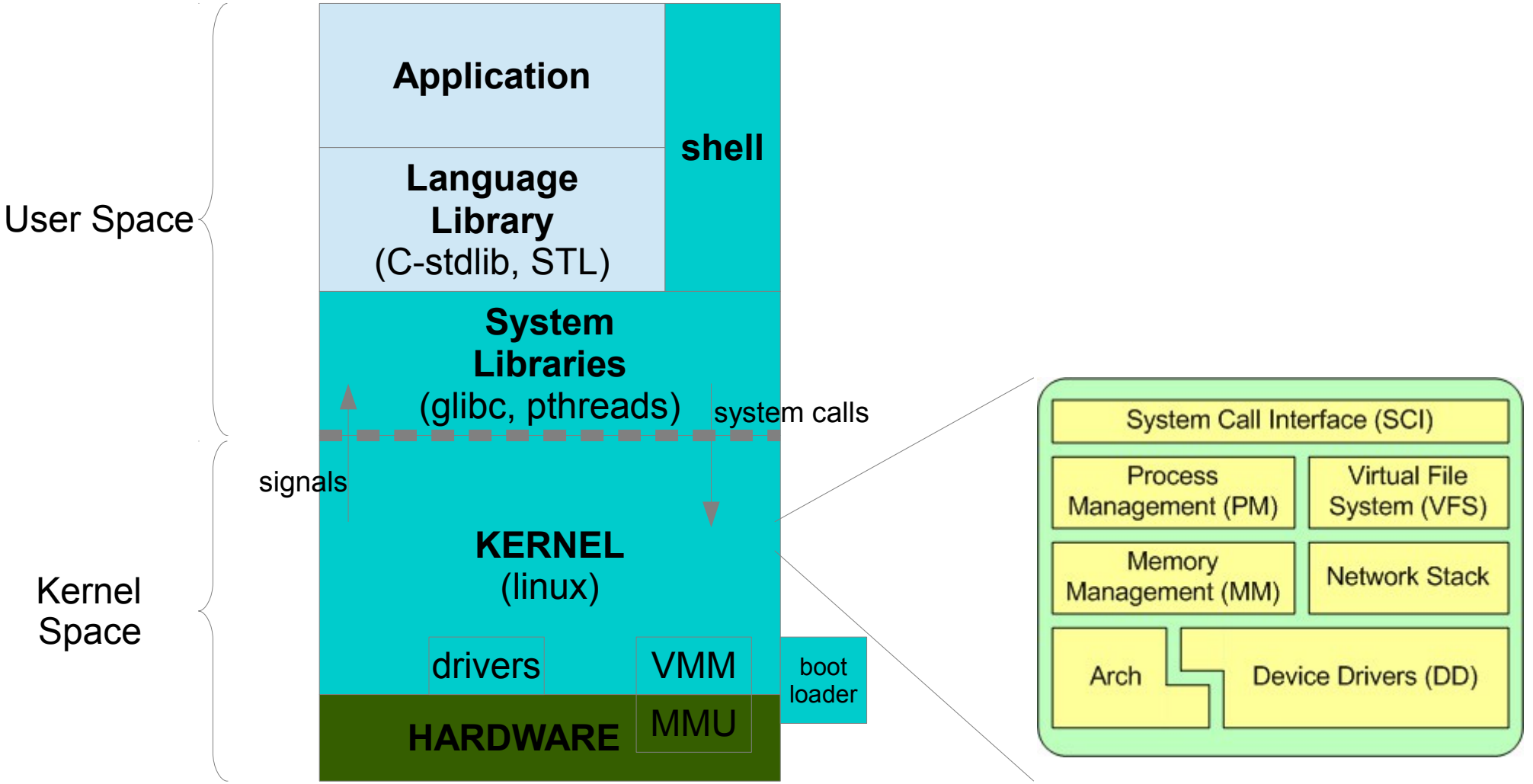
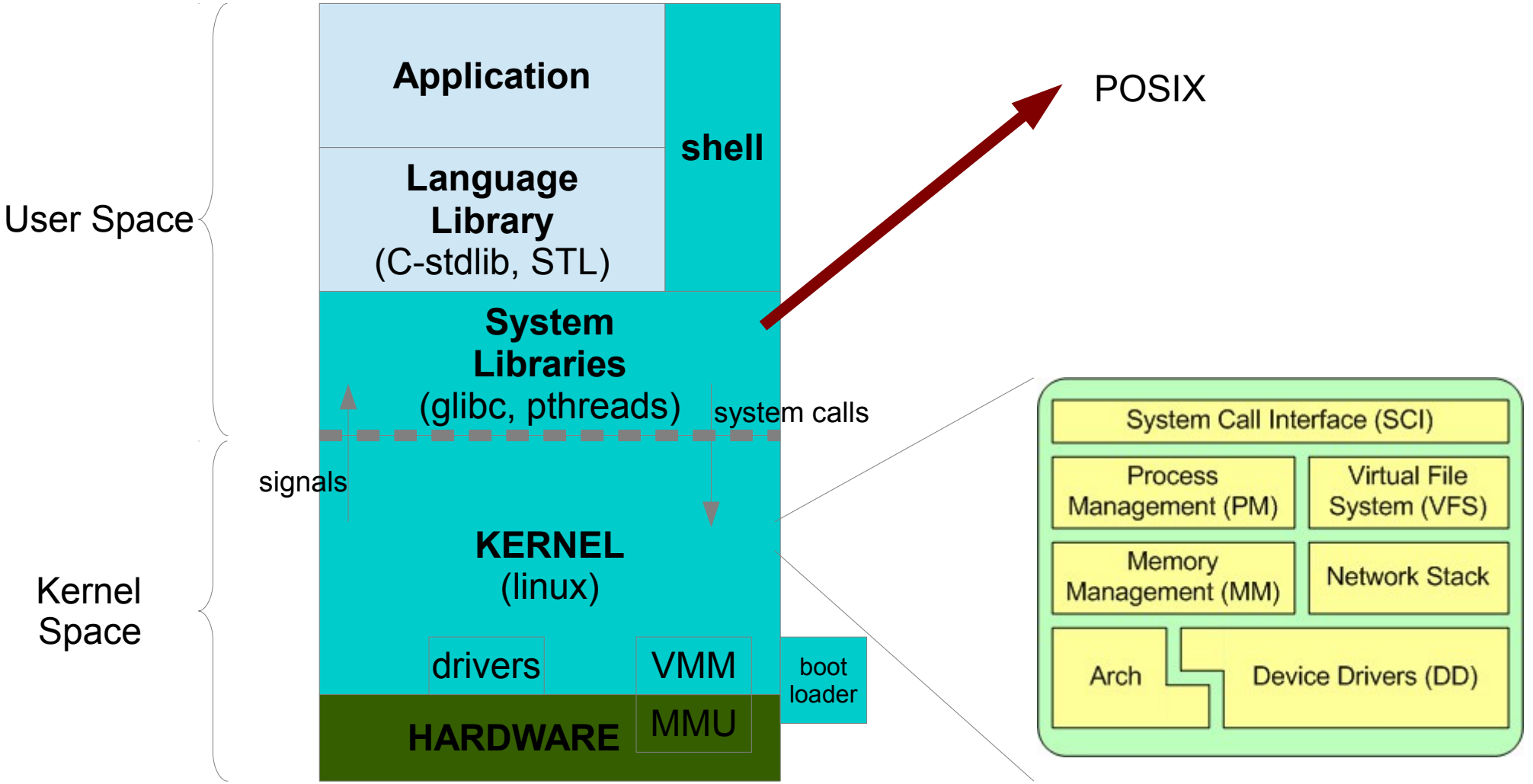
GCC internals intro y optimizationes

Daniel Gutson

Contexto: stack general


User Space
Kernel Space
Application
shell
Language Library
(C-stdlib, STL)
System Libraries
(glibc, pthreads)
system calls
signals
KERNEL
(linux)
drivers
VMM
boot loader
HARDWARE
MMU
System Call Interface (SCI)
Process Management (PM)
Virtual File System (VFS)
Memory Management (MM)
Network Stack
Arch
Device Drivers (DD)


User Space
Kernel Space
Application
Language Library
(C-stdlib, STL)
shell
System Libraries
(glibc, pthreads)
POSIX
system calls
signals
KERNEL
(linux)
drivers
VMM
MMU
boot loader
HARDWARE
System Call Interface (SCI)
Process Management (PM)
Virtual File System (VFS)
Memory Management (MM)
Network Stack
Arch
Device Drivers (DD)

Toolchain

- Conjunto de herramientas

Toolchain

- Conjunto de herramientas
  - No todas del mismo fabricante

Toolchain

- Conjunto de herramientas
  - No todas del mismo fabricante
  - Algunas lenguaje-específicas

Toolchain

- Conjunto de herramientas
  - No todas del mismo fabricante
  - Algunas lenguaje-específicas

- Construye binarios

Toolchain

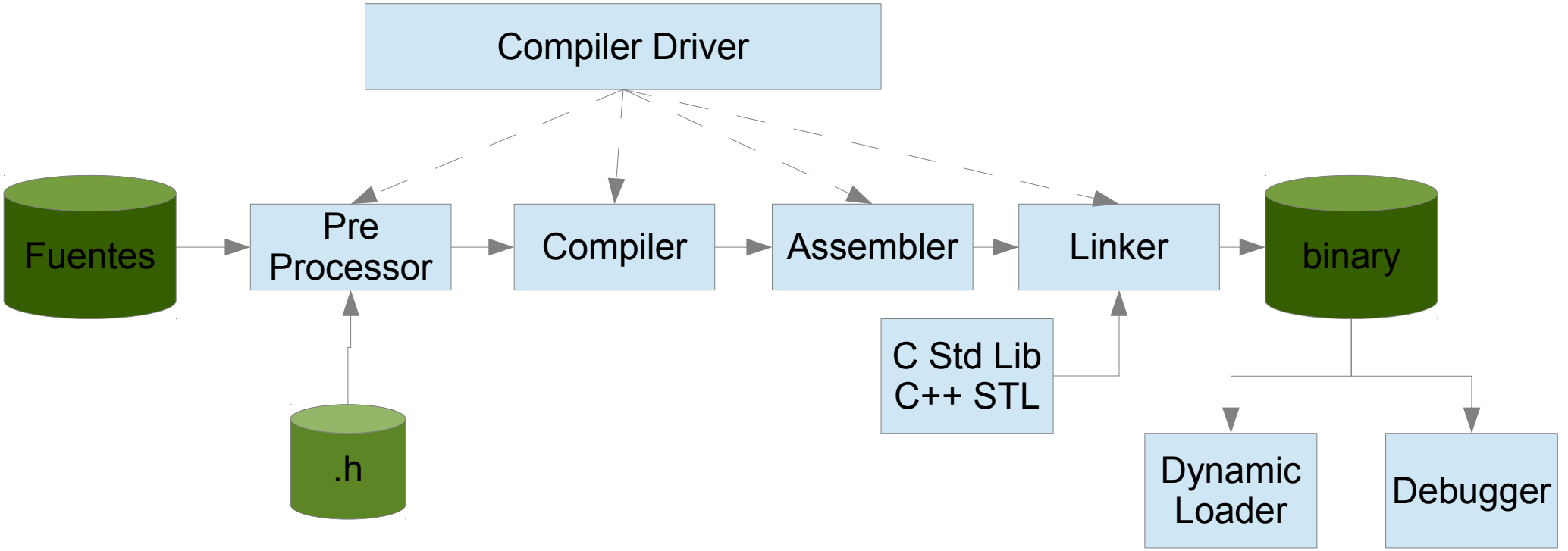
- Conjunto de herramientas
  - No todas del mismo fabricante
  - Algunas lenguaje-específicas

- Construye binarios
  - Responden a una ABI

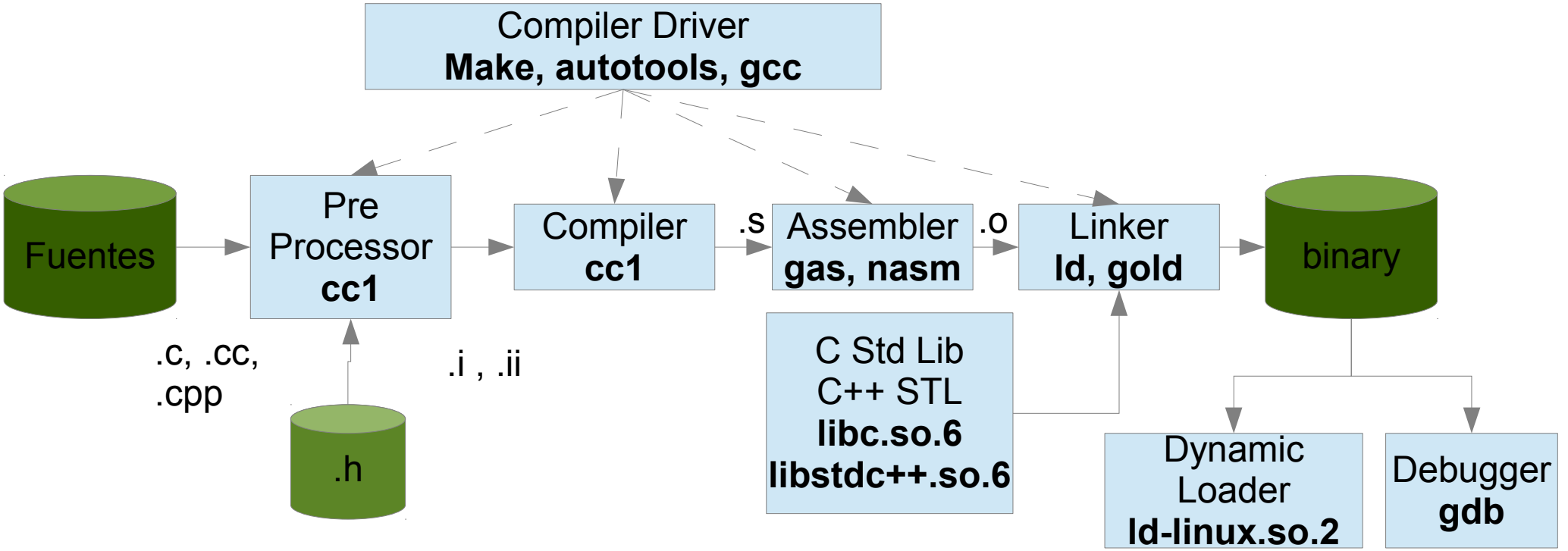
Toolchain

- Conjunto de herramientas
  - No todas del mismo fabricante
  - Algunas lenguaje-específicas
- Construye binarios
  - Responden a una ABI
- Características
  - Host arch.
  - Target arch.

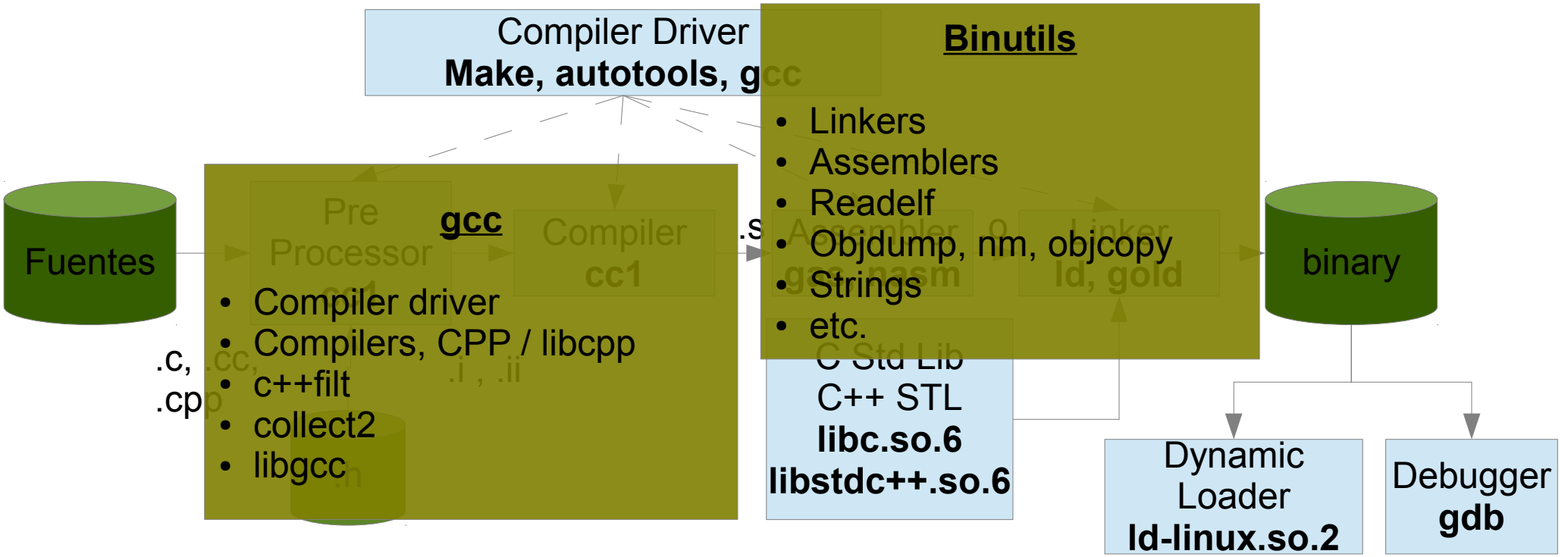
C & C++ Toolchain General



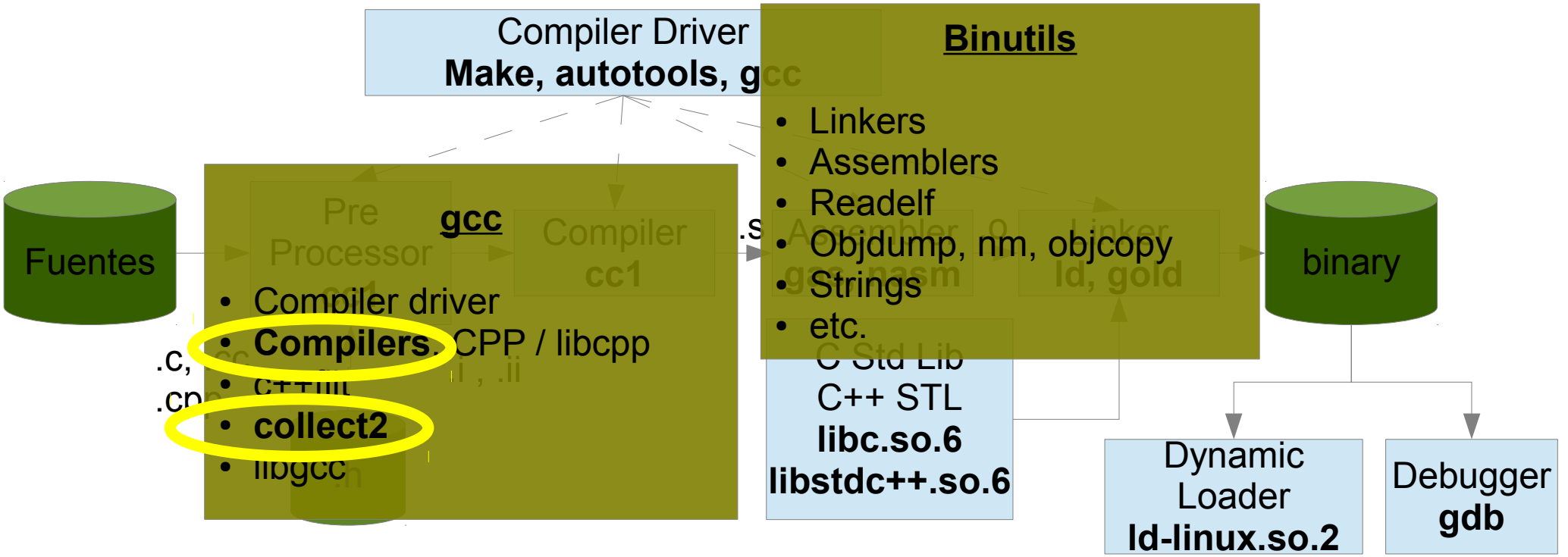
C & C++ Toolchain
GNU



C & C++ Toolchain
GNU

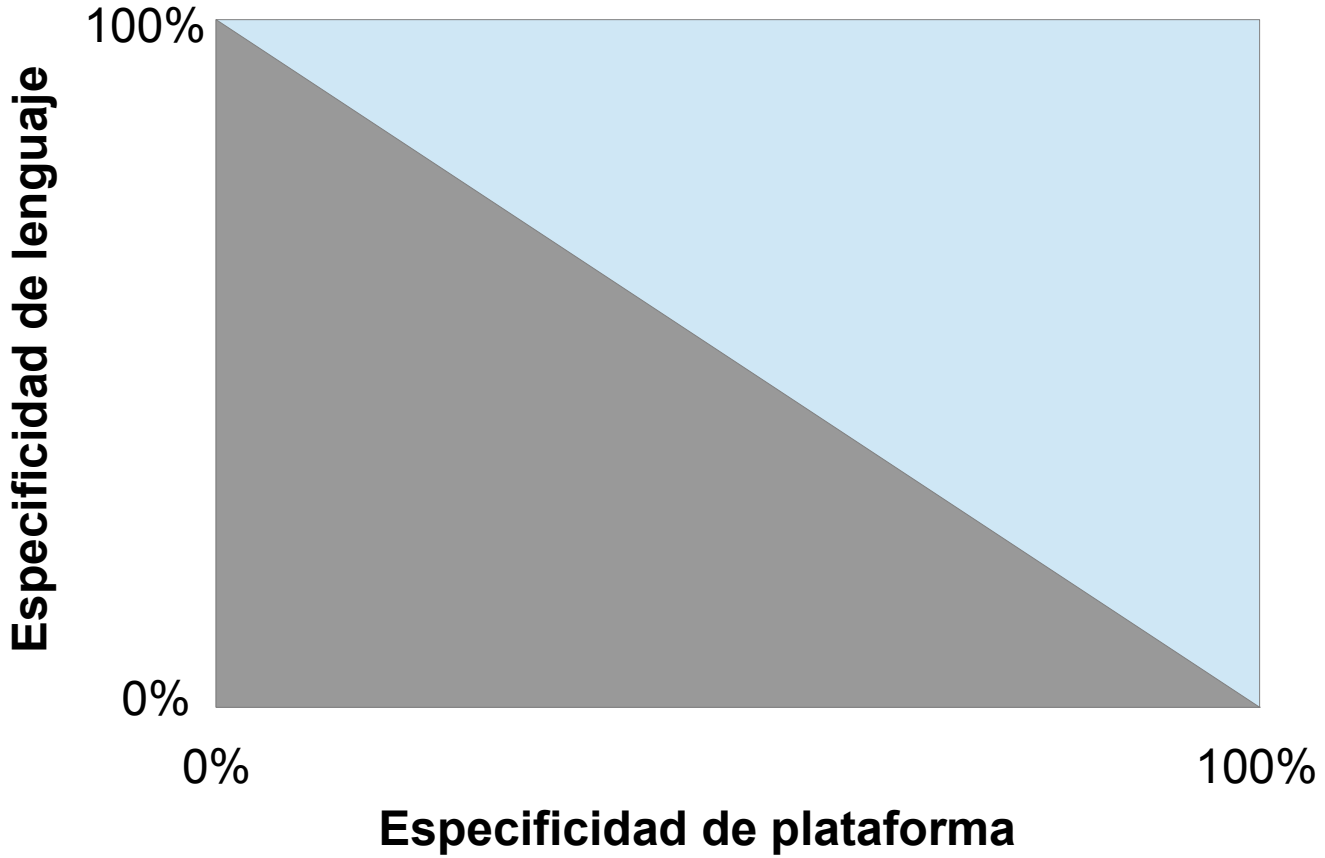


C & C++ Toolchain
GNU



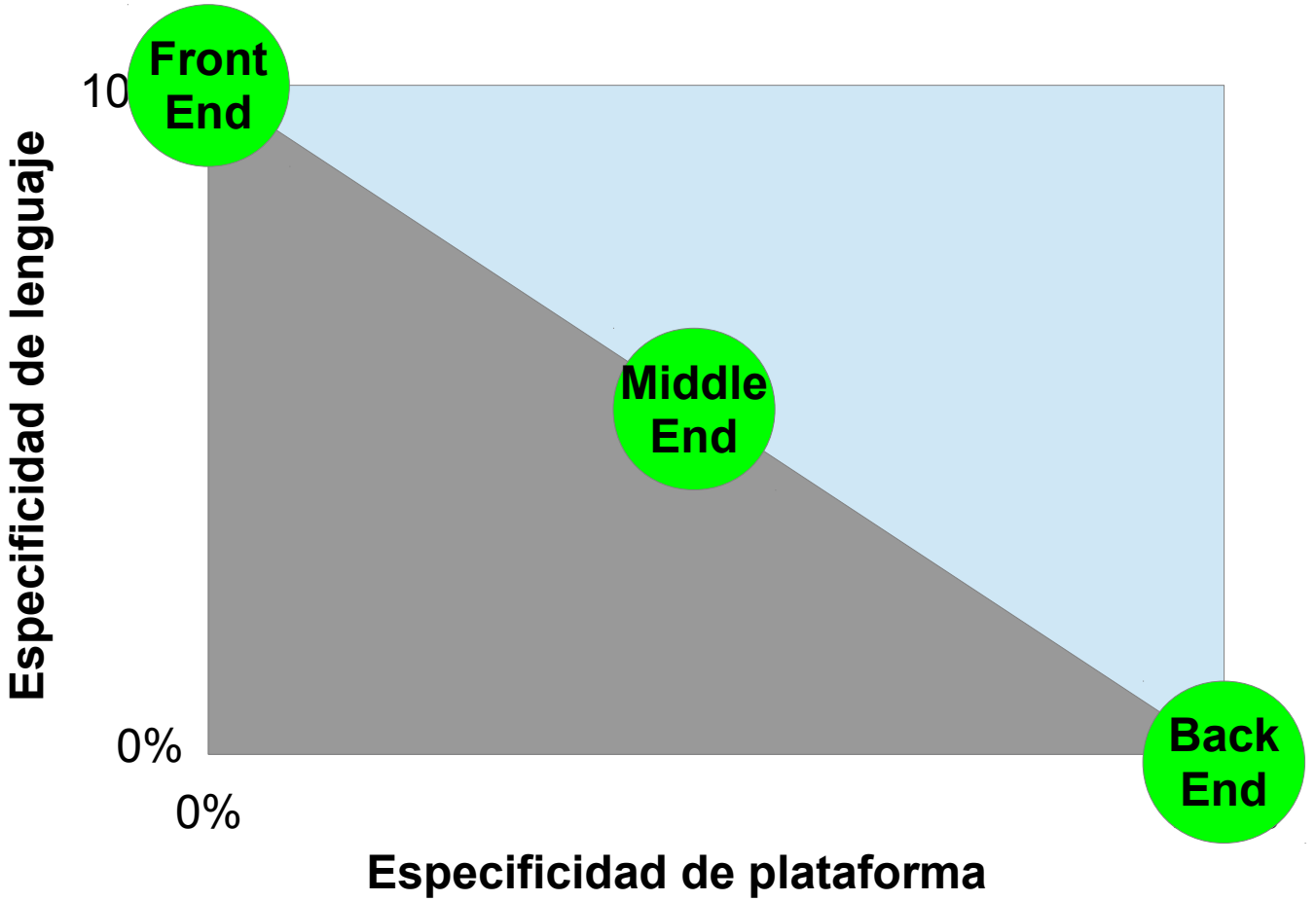
La colección de compiladores: GCC

- Comparten arquitectura común



La colección de compiladores: GCC

- Comparten arquitectura común



La colección de compiladores: GCC

- Comparten arquitectura común
- Comparten karma común
  - La antipatía de los que hacen el linker

Opciones de GCC

- **-f**xxxxxxx **F**ront-end específicas
- **-m**xxxxxxx **M**achine (back-end específicas)
- **-W**xxxxxxx **W**arnings (todos-los-end)
- **-O**xxxxxxx **O**ptimization (todos-los-end)

Opciones de GCC

- **-f**xxxxxxx **F**ront-end específicas
- **-m**xxxxxxx **M**achine (back-end específicas)
  - **-march**xx **Arch**itecture
  - **-mcpu**xxx **CPU**-específicas
- **-W**xxxxxxx **W**arnings (todos-los-end)
- **-O**xxxxxxx **O**ptimization (todos-los-end)

Optimizaciones en GCC

- Optimizaciones en Front-End
  - Reconocen language-idioms
- Optimizaciones en Middle-end
  - Transforman loops (ej Graphite)
- Optimizaciones en Back-end
  - Selección de instrucciones apropiadas
  - Instruction schedulling
- Optimización por vectorización
- WPO (optimizaciones que deberían ser LTO)

Optimizaciones del Front-end C++

- Inlining C++-específico
- TMP
- RVO
- Copy Ellision
- Eliminación de código muerto

Don't reach backend: TMP

- **Declarative**
  - Integer Arithmetic Calculi
    - Uses enums
  - Type calculi
- **Imperative**
  - Float Arithmetics
  - Expression Templates
  - Loop Transformations
  - AOP

  => All use & depend on inlining

  -O3, -ftemplate-depth, cache, etc.

TMP Declarative – Example: Factorial

```
template <unsigned int N>
struct Facto
{
	enum { result = N * Facto<N-1>::result };
};

template <>
struct Facto<0>
{
	enum { result = 1 };
};

int main()
{
	return Facto<4>::result;
}
```

```
08048494 <main>:
 8048494:	55                   	push   %ebp
 8048495:	89 e5                	mov    %esp,%ebp
 8048497:	b8 18 00 00 00       	mov    $0x18,%eax
 804849c:	5d                   	pop    %ebp
 804849d:	c3                   	ret
```

TMP Imperative – Example: PI

```
template <unsigned int N> struct PI {
	static float sign()	{
		return 1 - (2*(signed(N) % 2));
	}

	static float result()	{
		return PI<N-1>::result() + sign() * 4.0f / ((2*N)+1);
	}
};

template <> struct PI<0>
{
	static float result() { return 4.0f; }
};

int main() {
	cout << PI<50>::result() << endl;
	return 0;
}
```

TMP Imperative – Example: PI

```
template <unsigned int N> struct PI {
	static float sign()	{
		return 1 - (2*(signed(N) % 2));
	}

	static float result()		{
		return PI<N-1>::result() + sign() * 4.0f / ((2*N)+1);
	}
};

template <> struct PI<0>
{
	static float result() { return 4.0f; }
};

int main() {
	cout << PI<50>::result() << endl;
	return 0;
}
```

TMP Imperative – Example: PI

```
template <unsigned int N> struct PI {
        static float sign()     {
                return 1 - (2*(signed(N) % 2));
        }

        static float result()   {
                return PI<N-1>::result() + sign() * 4.0f / ((2*N)+1);
        }
};

te
{

};

int

}
```

```
08048790 <main>:
 8048790:       55                              push   %ebp
 8048791:       89 e5                           mov    %esp,%ebp
 8048793:       83 e4 f0                        and    $0xfffffff0,%esp
 8048796:       56                              push   %esi
 8048797:       53                              push   %ebx
 8048798:       83 ec 18                        sub    $0x18,%esp
 804879b:       d9 05 d0 88 04 08               flds   0x80488d0
 80487a1:       dd 5c 24 04                     fstpl  0x4(%esp)
 80487a5:       c7 04 24 40 a0 04 08            movl   $0x804a040,(%esp)
 80487ac:       e8 bb fe ff ff          call    804866c <std::basic_ostream<char,
std::char_traits<char> >& std::basic_ostream<char, std::char_traits<char>
>::_M_insert<double>(double)@plt>
```

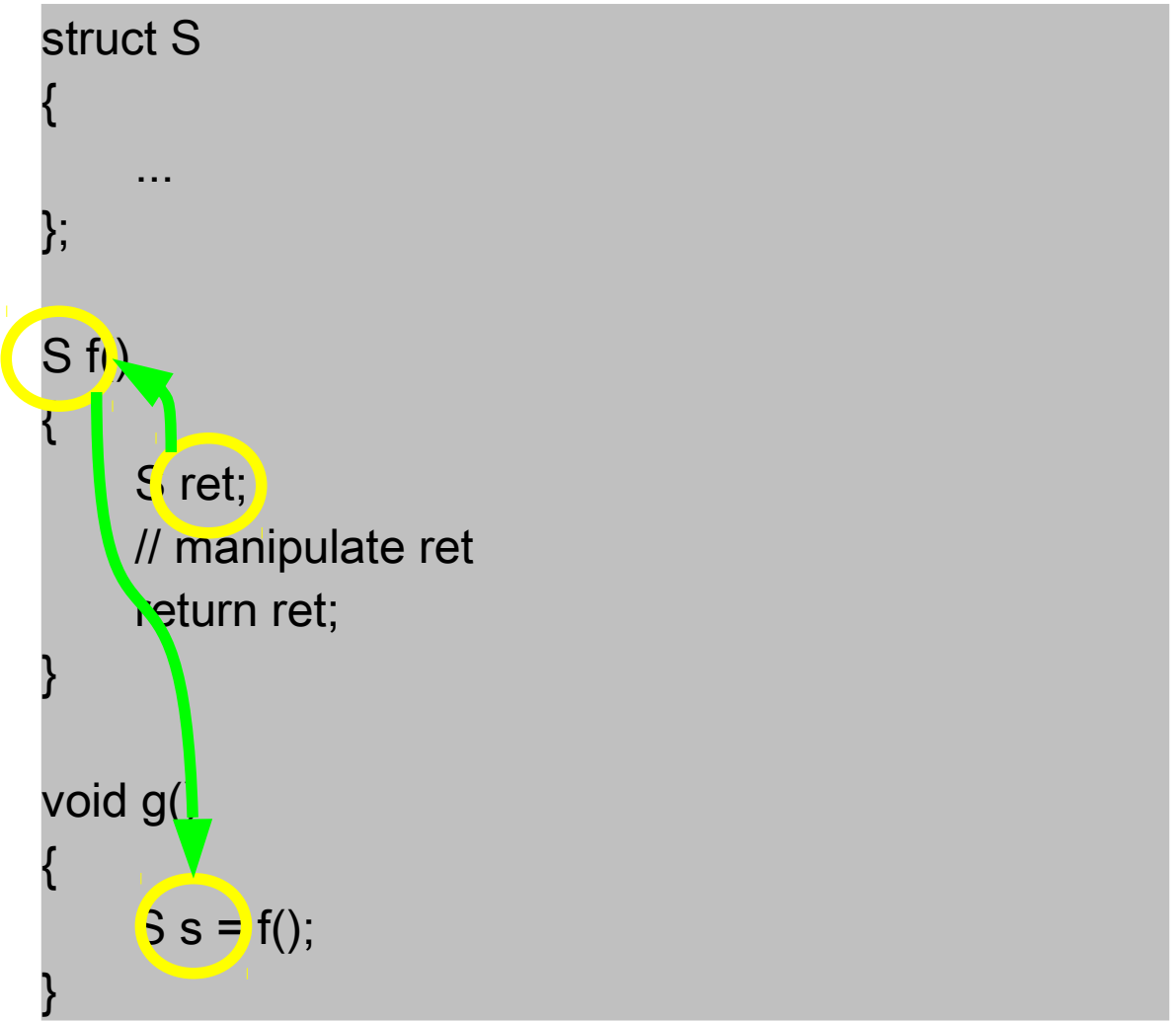
Special Methods – RVO

```
struct S
{
	...
};

S f()
{
	S ret;
	// manipulate ret
	return ret;
}

void g()
{
	S s = f();
}
```

Special Methods – RVO



Special Methods – RVO

```
struct S
{
	S() { cout << “def\n”; }
	S(const S& other) { cout << “copy\n”; }
};

S f()
{
	S ret;
	return ret;
}

int main()
{
	S s = f();
	return 0;
}
```

??

Special Methods – RVO

```
struct S
{
	S() { cout << “def\n”; }
	S(const S& other) { cout << “copy\n”; }
};

S f()
{
	S ret;
	return ret;
}

int main()
{
	S s = f();
	return 0;
}
```

```
def                g++ 4.4.3, no options

def
copy

def
copy               g++ -fno-elide-constructors
copy
```

Special Methods – Copy Elision

```
struct S
{
	S(int i) { cout << i << "\n"; }
	S(const S&) { cout << "copy\n"; }
};

int main()
{
	S s1(1);
	S s2 = 2;
	return 0;
}
```

??

Special Methods – Copy Elision

```
struct S
{
	S(int i) { cout << i << "\n"; }
	S(const S&) { cout << "copy\n"; }
};

int main()
{
	S s1(1);
	S s2 = 2;
	return 0;
}
```

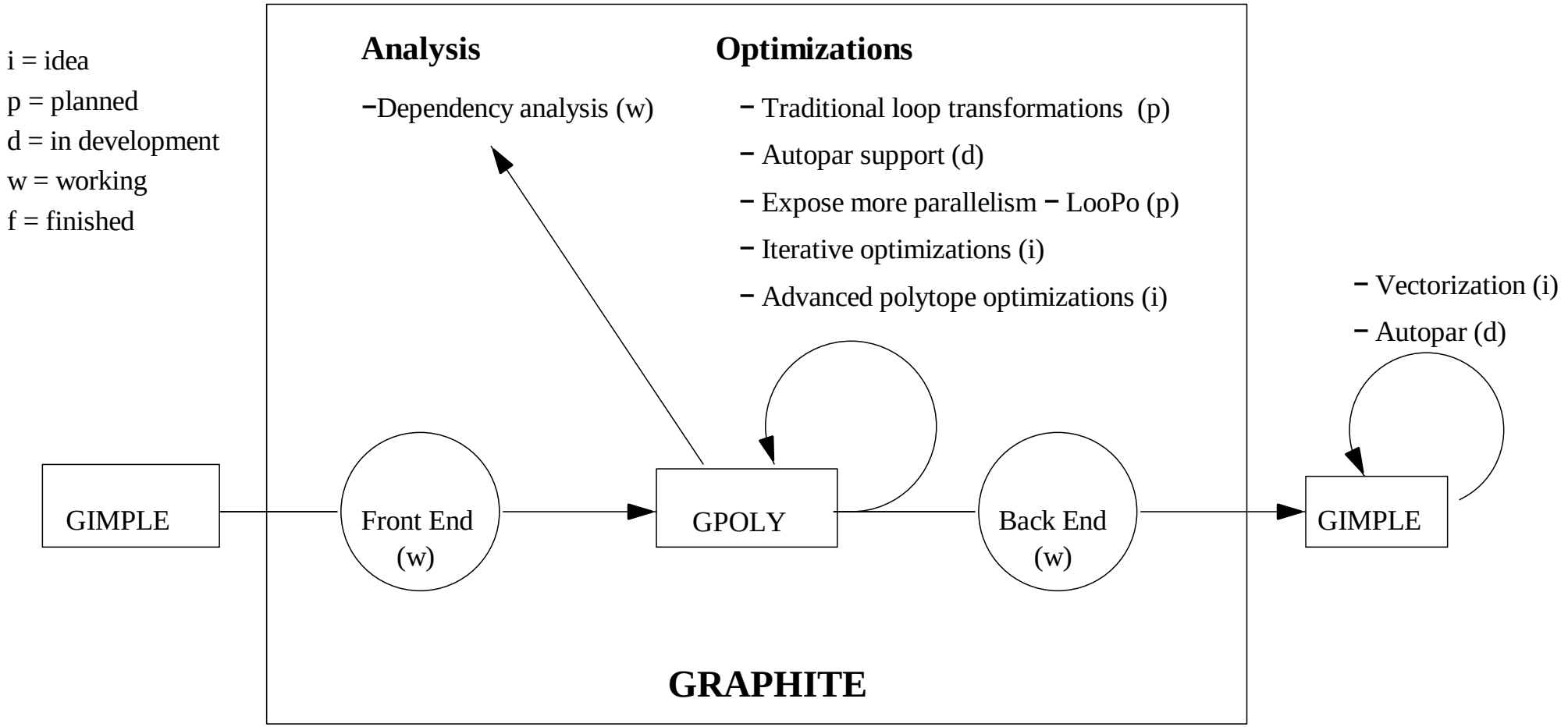
```
1
2
```
g++ 4.4.3, no options

```
1
2
copy
```
g++ -fno-elide-constructors

Optimizationes Middle-end

- Transformaciones de loops
  - -funroll-loops
  - -fprefetch-loop-arrays
  - Graphite

Optimizaciones Middlend
Ej: Graphite



Graphite: Gimple Represented as Polyhedra

Graphite is a framework for high-level memory optimizations using the polyhedral model.

Graphite branch has been merged in August 2008 into trunk.

-floop-parallelize-all

Use the Graphite data dependence analysis to identify loops that can be parallelized. Parallelize all the loops that can be analyzed to not contain loop carried dependences without checking that it is profitable to parallelize the loops.

Optimizaciones Back-end

- Backend:
  - RTL tree + RTL language + RTL Engine + RTL Compiler
- [Middle-End] ----RTL tree→ [Back-end: Treewalker]
- Register Transfer Language:
  - RTL: Lisp-like
    - Puede embeber código C
    - Puede invocar funciones en C
  - Describe: conj. REGLAS
- Cada RTL Rule:
  - Tree Pattern
  - Condición (incluye código C)
  - Output assembly
  - Atributos (arch-specific), puntaje
- Score function

Back-end impl.

- Fuentes involucrados:
  - *arch*.md: Código RTL (MD: Machine Descrip.)
  - *arch.*h: Definiciones (macros) describen arch (qué reg. hay, para qué se usan, …)
  - *arch*.c: Helper functions en C + *arch*_rtx_costs()

RTL

- RTL Model:
  - Máquina abstracta
  - Cantidad infinita de registros (pseudo-registers)
  - Realiza varias pasadas sobre sí mismo.

- La PRIMER OPTIMIZACION:
  - Al buildear GCC :-) → reglas para optimizar reglas

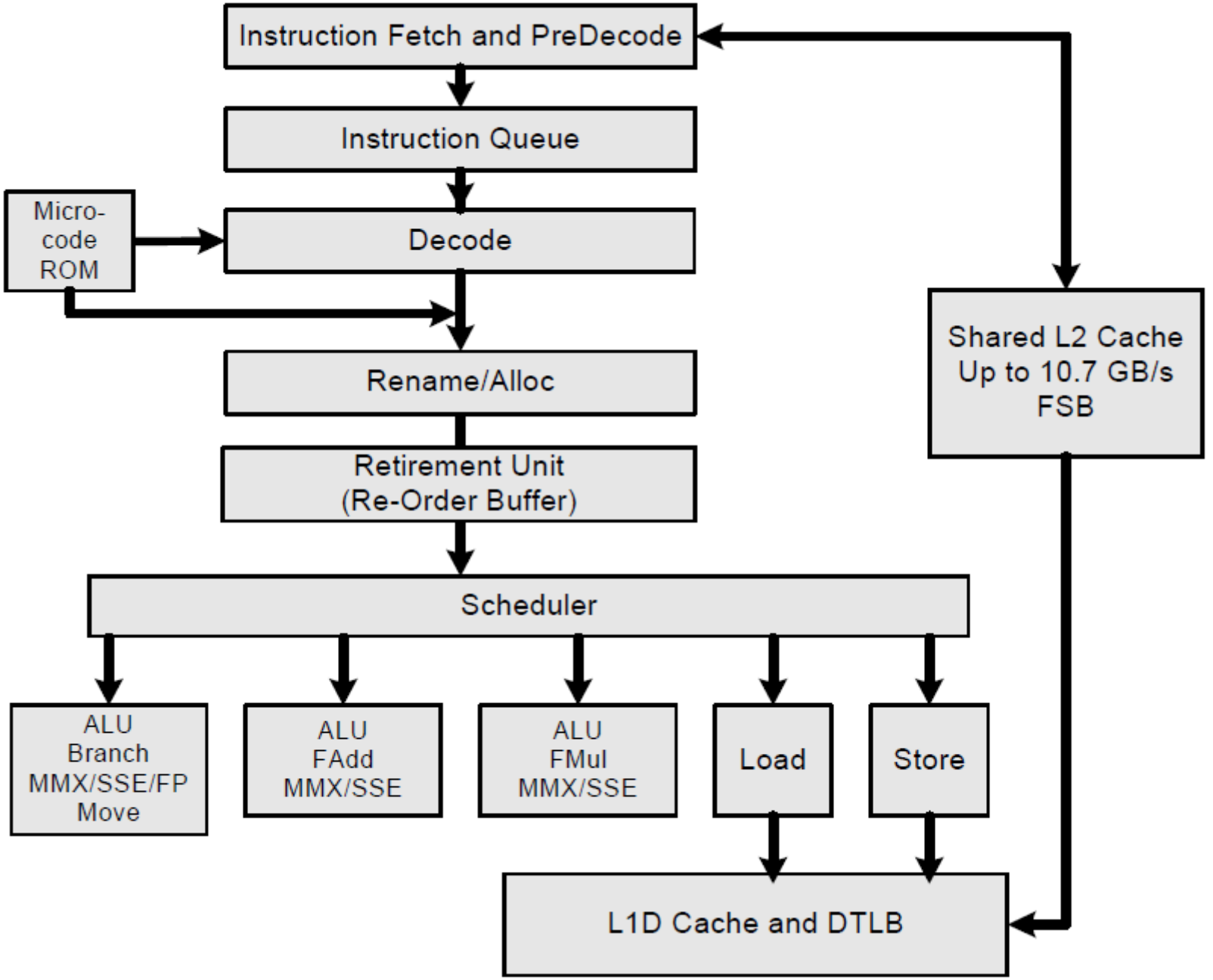
RTL

- Instruction scheduling pass
  - Tiene en cuenta el instruction decoder (ciclos), uOps
  - "***Assembly/Compiler Coding Rule 25. (M impact, M generality)***

    Avoid putting explicit references to ESP *in a sequence of stack operations (*POP, PUSH, CALL, RET*)."*
- Instructioin selection pass
  - Tiene en cuenta subcores, pipeline
- Reloading Pass
  - Asigna *pseudo-registros* en *registros reales*
    - Cuando se queda sin registros reales: baja a memoria.
  - Una regla: cómo hacer reloading (insns, etc.)

- Instruc
  - Tiene
  - "***Assen***

    Avoid p
    *(*POP, F
- Instruc
  - Tiene
- Reload
  - Asigr
    - Cu
  - Una regla: como hacer reloading (insns, etc.)



RTL

- Instruction scheduling pass
  - Tiene en cuenta el instruction decoder (ciclos), uOps
  - "***Assembly/Compiler Coding Rule 25. (M impact, M generality)***

    Avoid putting explicit references to ESP *in a sequence of stack operations (*POP, PUSH, CALL, RET*)."*
- Instructioin selection pass
  - Tiene en cuenta subcores, pipeline
- Reloading Pass
  - Asigna *pseudo-registros* en *registros reales*
    - Cuando se queda sin registros reales: baja a memoria.
  - Una regla: cómo hacer reloading (insns, etc.)

RTL

- Hojas RTL iniciales tienen **tipo específico** (cross-platform)
  - addi, addi3, subi, imul, …
- Con cada pasada
  - Aplican reglas
  - Se van transformando/expandiendo/uniendo en instrucciones

RTL: Ejemplo real

- MADD: Multiply And Add

- En C:

  a = a + b*c

- Input RTL simplificado:
  - NODO_asignación { a, NODO_suma { a , NODO_multiplicacion { b, c } } }

- RTL rule:

```
(define_insn "*madds"
  [(set (match_operand:ANYF 0 "register_operand" "=f")
    (plus:ANYF (mult:ANYF (match_operand:ANYF 1 "register_operand" "f")
                  (match_operand:ANYF 2 "register_operand" "f"))
           (match_operand:ANYF 3 "register_operand" "f")))]
  "ISA_HAS_FP4 && TARGET_FUSED_MADD"
  "madd.s %0,%3,%1,%2"
  [(set_attr "type" "fmadd")
   (set_attr "mode" "<UNITMODE>"))])
```

RTL: Ejemplo real

- MADD: Multiply And Add

- En C:

  a = a + b*c

- Input RTL simplificado:
  - NODO_asignación { a, NOD[illegible]na { a , NODO_multiplicacion { b, c } } }

- RTL rule:

Nombre
(* → debuggig)

```
(define_insn "*madds"
  [(set (match_operand:ANYF 0 "register_operand" "=f")
    (plus:ANYF (mult:ANYF (match_operand:ANYF 1 "register_operand" "f")
                          (match_operand:ANYF 2 "register_operand" "f"))
               (match_operand:ANYF 3 "register_operand" "f")))]
  "ISA_HAS_FP4 && TARGET_FUSED_MADD"
  "madd.s %0,%3,%1,%2"
  [(set_attr "type" "fmadd")
   (set_attr "mode" "<UNITMODE>"))])
```

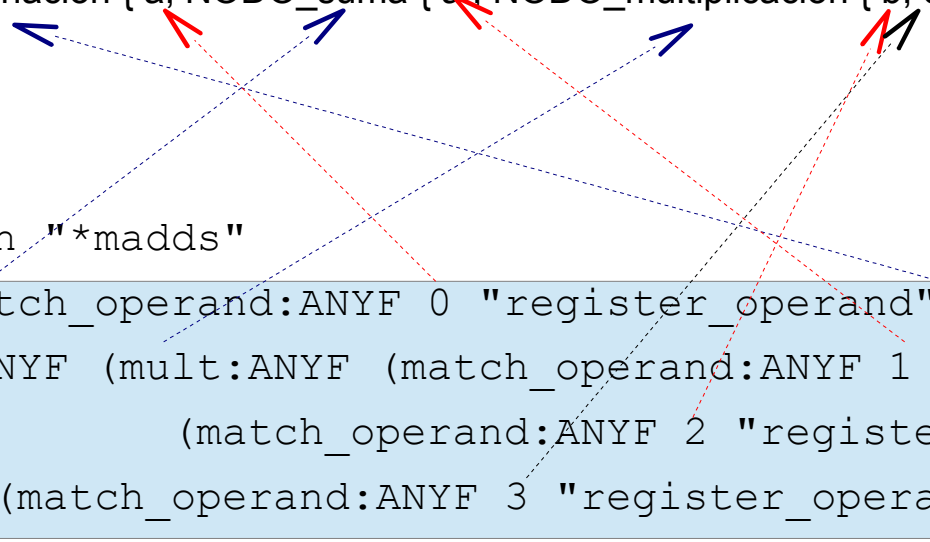
RTL: Ejemplo real

- MADD: Multiply And Add

- En C:

  a = a + b*c

- Input RTL simplificado:
  - NODO_asignación { a, NODO_suma { a , NODO_multiplicacion { b, c } } }



- RTL rule:

```
(define_insn "*madds"
  [(set (match_operand:ANYF 0 "register_operand" "=f")
    (plus:ANYF (mult:ANYF (match_operand:ANYF 1 "register_operand" "f")
                          (match_operand:ANYF 2 "register_operand" "f"))
               (match_operand:ANYF 3 "register_operand" "f")))]
  "ISA_HAS_FP4 && TARGET_FUSED_MADD"
  "madd.s %0,%3,%1,%2"
  [(set_attr "type" "fmadd")
   (set_attr "mode" "<UNITMODE>"))])
```

RTL: Ejemplo real

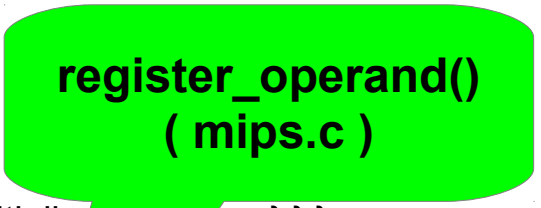
- MADD: Multiply And Add

- En C:

  a = a + b*c

- Input RTL simplificado:
  - NODO_asignación { a, NODO_suma { a , NODO_multiplica b, c } } }

- RTL rule:




```
(define_insn "*madds"
  [(set (match_operand:ANYF 0 "register_operand" "=f")
    (plus:ANYF (mult:ANYF (match_operand:ANYF 1 "register_operand" "f")
                          (match_operand:ANYF 2 "register_operand" "f"))
               (match_operand:ANYF 3 "register_operand" "f")))]
  "ISA_HAS_FP4 && TARGET_FUSED_MADD"
  "madd.s %0,%3,%1,%2"
  [(set_attr "type" "fmadd")
   (set_attr "mode" "<UNITMODE>"))])
```

RTL: Ejemplo real

- MADD: Multiply And Add

- En C:

  a = a + b*c

- Input RTL simplificado:
  - NODO_asignación { a, NODO_suma { a , NODO_multiplicacion { b, c } } }

- RTL rule:

```
(define_insn "*madds"
  [(set (match_operand:ANYF 0 "register_operand" "=f")
    (plus:ANYF (mult:ANYF (match_operand:ANYF 1 "register_operand" "f")
                          (match_operand:ANYF 2 "register_operand" "f"))
               (match_operand:ANYF 3 "register_operand" "f")))]
  "ISA_HAS_FP4 && TARGET_FUSED_MADD"
  "madd.s %0,%3,%1,%2"
  [(set_attr "type" "fmadd")
   (set_attr "mode" "<UNITMODE>"))])
```

#define ISA_HAS_FP4 ...
(mips.h)

RTL: Ejemplo real

- MADD: Multiply And Add

- En C:

  a = a + b*c

- Input RTL simplificado:
  - NODO_asignación { a, NODO_suma { a , NODO_multiplicacion { b, c } } }

- RTL rule:

```
(define_insn "*madds"
  [(set (match_operand:ANYF 0 "register_operand" "=f")
    (plus:ANYF (mult:ANYF (match_operand:ANYF 1 "register_operand" "f")
                  (match_operand:ANYF 2 "register_operand" "f"))
            (match_operand:ANYF 3 "register_operand" "f")))]
  "ISA_HAS_FP4 && TARGET_FUSED_MADD"
  "madd.s %0,%3,%1,%2"
  [(set_attr "type" "fmadd")
   (set_attr "mode" "<UNITMODE>"))])
```

RTL: Ejemplo real

- MADD: Multiply And Add

- En C:

  a = a + b*c

- Input RTL simplificado:
  - NODO_asignación { a, NODO_suma { a , NODO_multiplicacion { b, c } } }

- RTL rule:

```
(define_insn "*madds"
  [(set (match_operand:ANYF 0 "register_operand" "=f")
    (plus:ANYF (mult:ANYF (match_operand:ANYF 1 "register_operand" "f")
                  (match_operand:ANYF 2 "register_operand" "f"))
            (match_operand:ANYF 3 "register_operand" "f")))]
  "ISA_HAS_FP4 && TARGET_FUSED_MADD"
  "madd.s %0,%3,%1,%2"
  [(set_attr "type" "fmadd")
   (set_attr "mode" "<UNITMODE>"))])
```

Scheduling rules, cost

RTL: Ejemplo real 2

- http://gcc.gnu.org/ml/gcc-patches/2009-07/msg00009.html

[PATCH] Workaround for Janus 2CC core errata

- *From*: **Daniel Gutson <dgutson at codesourcery dot com>**
- *To*: **gcc-patches at gcc dot gnu dot org**
- *Date*: **Wed, 01 Jul 2009 03:26:14 -0300**
- *Subject*: **[PATCH] Workaround for Janus 2CC core errata**

The attached patch implements a work-around for an errata in the Janus 2CC core.

I tested this by running the gcc test suite, including the two new test cases I added in this patch.

If accepted, please commit if for me since I don't have write access.

Thanks,
Daniel.

WPO/LTO vs Linker (gcc vs binutils)

- Contexto: ld (binutils) → lenguaje inespecífico
- OK para C, Fortran (70's)
  - Ej: init. variables globales: .bss
- NO OK para C++ (90's)
  - Ej: ctor/dtor objetos globales

collect2

- GCC's fake linker
  - Responsible for collecting calls for ctors & dtors in the .o files
  - Creates tables
  - Creates __do_global_ctors_aux() and the __static_initialization_and_destruction_0()
    - The latter: one per .o having globals
    - Has a call to __cxa_atexit
- Procedure:
  - Links the program once with the *real linker*
  - Collects data from binary (using nm)
  - Generates and compiles a temporary .c file
  - Re links everything with the new .o file with the *real linker*
- Verbose: add -debug to the linker (from gcc: -Wl,-debug)

Uso de collect2 para optimizar

- “Ve todo”
- Saca ventaja si se usa *gold*: -fuse-linker-plugin
- IPA: Inter-Procedure analysis: callgraph
  - LGEN:
    - genera summary
    - graba summary
  - WPA
    - lee summary
    - “execute” (calcula opt.)
    - escribe opt. data
  - LTRANS:
    - lee opt. Data
    - Transforma (“transform”)

IPA: algunas optimizaciones

- Puede detectar funciones no usadas
- Permite cross-inlining en loops ajustados
- Propagation: puede saber valores de argumentos
- Puede saber valores de punteros
  - a objetos
  - a funciones

    => ahorra indirecciones.

Preguntas?

No, gracias.

Gracias!

Daniel.gutson@fudepan.org.ar