C2 language

Bas van den Berg

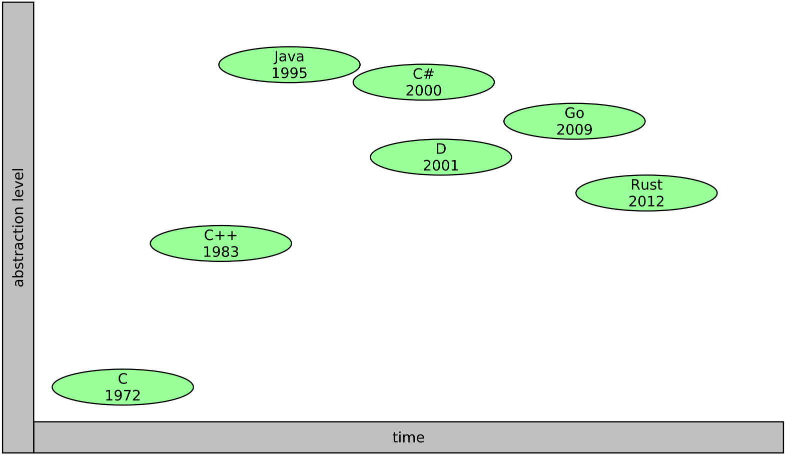
Fosdem 2015, Brussels

Goal

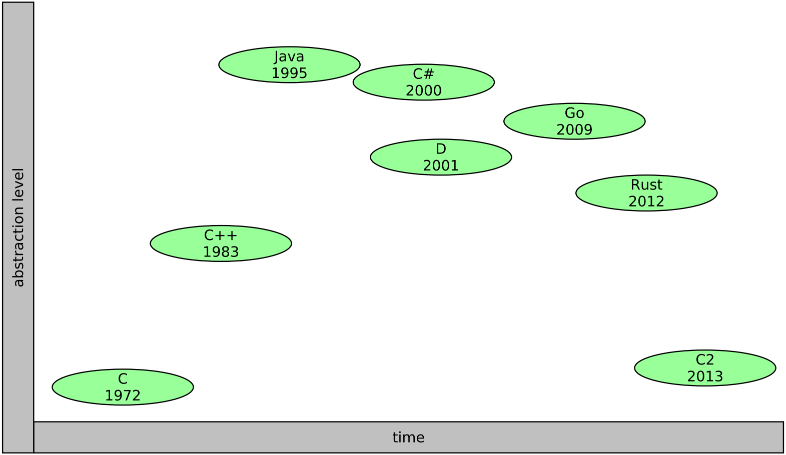
Goal of this presentation:

- show the C2 language
- show how you can re-use LLVM/Clang components
- get feedback/ideas

Programming language evolution



Programming language evolution



C2 design goals

- C2 is an *evolution* of C
- higher development speed
- same/better speed of execution
- integrated build system
- stricter syntax + analyser
- enable+build better tooling
- easy integration with C (and vice-versa)
- wider *scope* than C

C2 explicit non-goals

- higher-level features (garbage collection, classes, etc)
- completely new language

C - good things

Strong points:

- many developers
- huge code base
- high-performance runtime
- abstraction/domain

C - things to improve

Weak points:

- #include system
- tricky syntax

  `8[buffer]`

  `char *(*(**foo [][8])())[]`
- many other tools needed

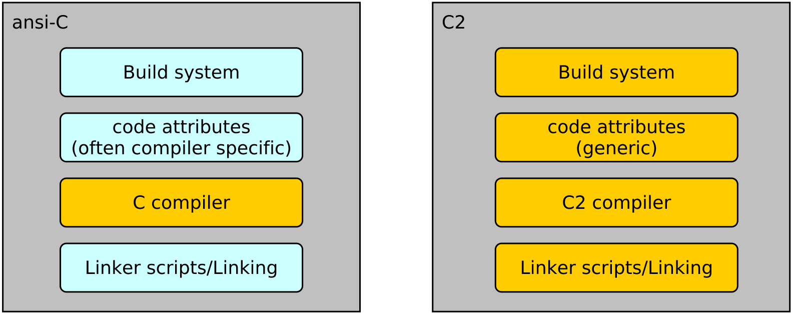
  `make, analysers, heavy use of pre-processor`
- lots of typing

  `header files, forward declarations, etc`
- compiler allows too much

  `using uninitialized variable is a warning!?!`

⟹ each item slows down development!

Language Scope



ansi-C

Build system

code attributes (often compiler specific)

C compiler

Linker scripts/Linking

C2

Build system

code attributes (generic)

C2 compiler

Linker scripts/Linking

⟹ widening the language scope allows for huge improvements and ease of use.

Introduction C2

C2 - examples and some features

Example: Hello World!

hello_world.c2

```
module hello_world;

import stdio as io;

func int main(int argc, char*[] argv) {
    io.printf("Hello World!\n");
    return 0;
}
```

Spot the six (!) differences...

Example: Hello World!

hello_world.c2

```
module hello_world;

import stdio as io;

func int main(int argc, char*[] argv) {
    io.printf("Hello World!\n");
    return 0;
}
```

Spot the six (!) differences...

⟹ mostly function bodies are almost identical

Feature: multi-pass parser

example.c2

```
module example;

func int foo() {
    Number n = getNumber();
    return n;
}

func Number bar() {
    Number b = 10;
    return b;
}

type Number int;
```

⟹ declaration order doesn't matter (even between files!)

Feature: modules

gui.c2

```
module gui;

import utils local;

Buffer buf;

func void run()
{
    utils.log("ok");
    log("also ok");
}
```

utils_buf.c2

```
module utils;

public type Buffer int[10];
```

utils_log.c2

```
module utils;

public func void log(int8* msg)
{
    ...
}
```

⟹ no header files, only define everything once.

⟹ no filenames are specified in code.

Feature: Incremental arrays

foo.c2

```
type Friend struct {
    char[32] name;
    int      age;
}

Friend[] friends = {}

friends += { "john", 25 }

#ifdef MORE_FRIENDS
friends += { { "alice", 30 },
             { "santa", 60 } }
#endif
```

⟹ this avoids multiple-includes of .td files (like Clang does)

Feature: BitOffsets

foo.c (ANSI-C)

```
unsigned int b = (a >> 8) & 0xFF;
```

Feature: BitOffsets

foo.c (ANSI-C)

```
unsigned int b = (a >> 8) & 0xFF;
```

foo.c2

```
func void foo() {
    uint32 a = 0x1234;
    uint32 b = a[15:8]; // will be 0x12
    uint8  c = a[7:0];  // will be 0x34
}
```

⟹ often used in drivers

⟹ TBD if also allowed on LHS: $a[16:13] = 3;$

⟹ TBD combine with *reg32* or *reg64* builtin-type?

Feature: recipe file (v1)

recipe.txt

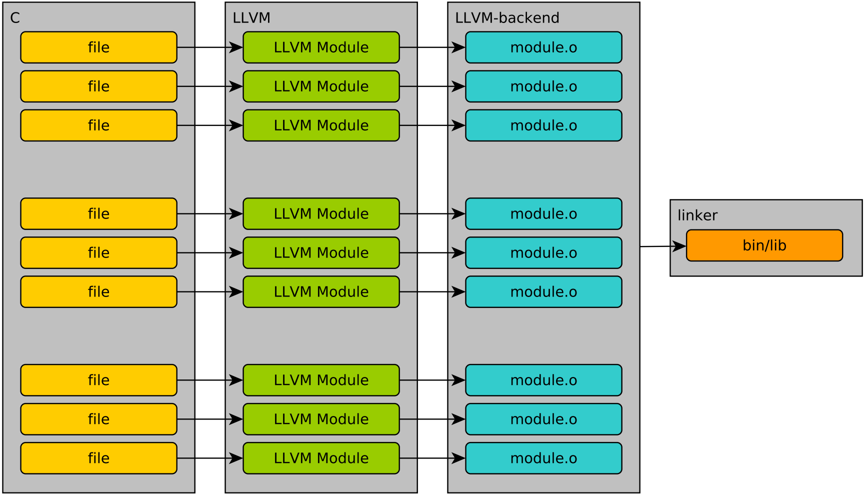
```
target example1
  $warnings no-unused
  example1/gui.c2
  example1/utils.c2
end

target mylib
  $config NO_DEBUG WITH_FEATURE1 FEATURE2
  example2/mylib1.c2
  example2/mylib2.c2
end
```

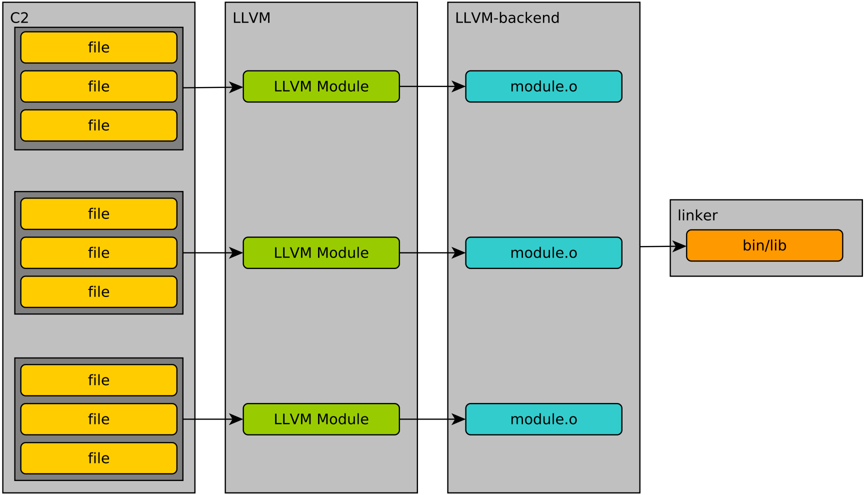
⟹ C2 compiler always knows all files in the project.

⟹ only the C2 compiler is needed to build (no buildsystem).

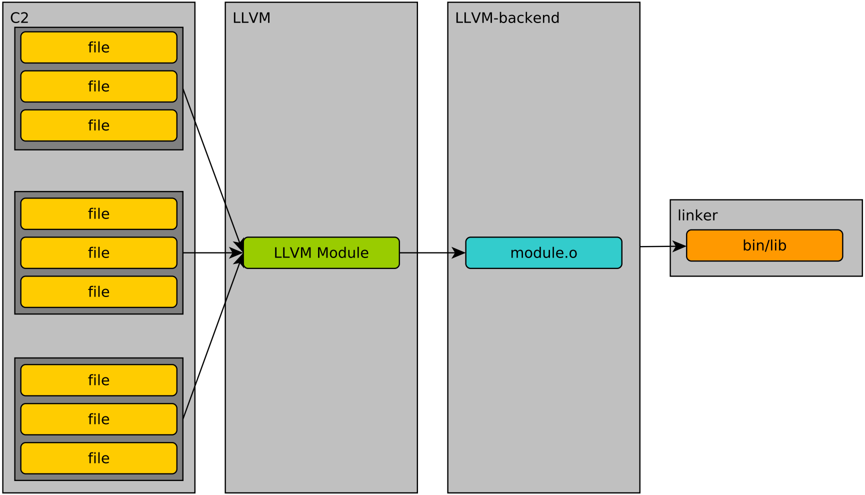
Feature: partial/full 'LTO'



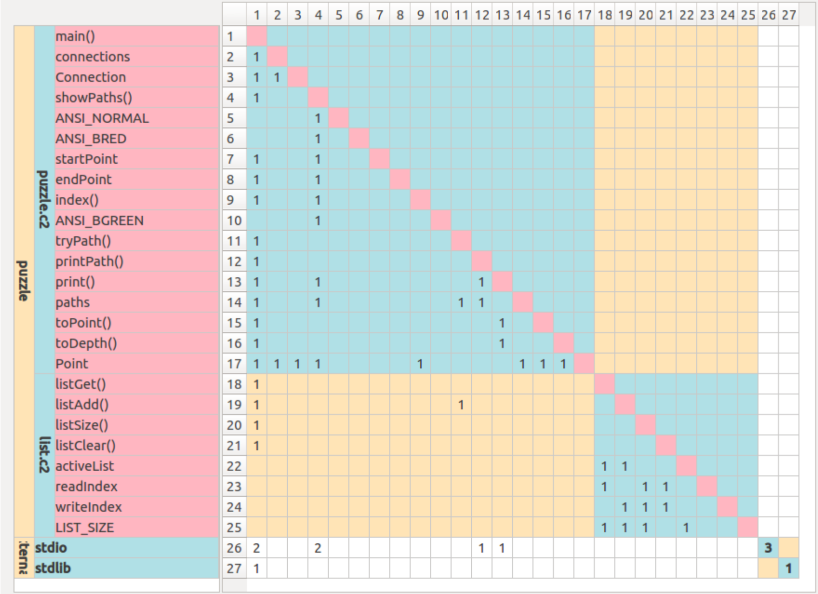
Feature: partial/full 'LTO'



Feature: partial/full 'LTO'



Feature: (DSM) dependency generation



				1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
puzzle	puzzle.c2	main()	1																											
		connections	2	1																										
		Connection	3	1	1																									
		showPaths()	4	1																										
		ANSI_NORMAL	5				1																							
		ANSI_BRED	6				1																							
		startPoint	7	1			1																							
		endPoint	8	1			1																							
		index()	9	1			1																							
		ANSI_BGREEN	10				1																							
		tryPath()	11	1																										
		printPath()	12	1																										
		print()	13	1			1								1															
		paths	14	1			1							1	1															
		toPoint()	15	1												1														
		toDepth()	16	1												1														
		Point	17	1	1	1	1					1					1	1	1											
	list.c2	listGet()	18	1																										
		listAdd()	19	1										1																
		listSize()	20	1																										
		listClear()	21	1																										
		activeList	22																		1	1								
		readIndex	23																		1		1	1						
		writeIndex	24																			1	1	1						
		LIST_SIZE	25																		1	1	1		1					
:term	stdio		26	2			2								1	1													3	
	stdlib		27	1																										1

Keyword changes

removed keywords:

- extern
- static
- typedef
- long
- short
- signed
- unsigned

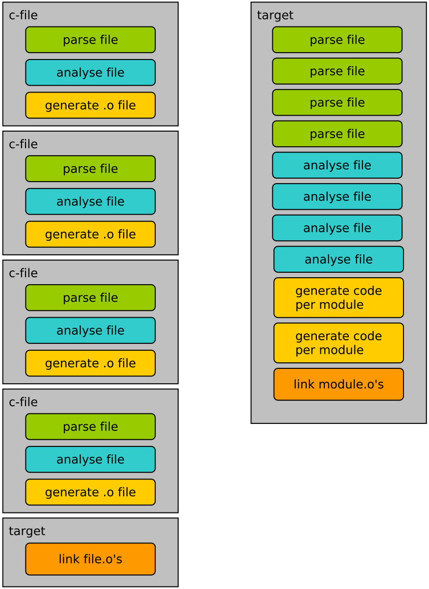
new keywords:

- module
- import
- as
- public
- local
- type
- func
- nil
- elemsof
- int8
- int16
- int32
- int64
- uint8
- uint16
- uint32
- uint64
- float32
- float64

C2 Compiler

the C2 compiler

C2 compiler: build process



c-file
parse file
analyse file
generate .o file

c-file
parse file
analyse file
generate .o file

c-file
parse file
analyse file
generate .o file

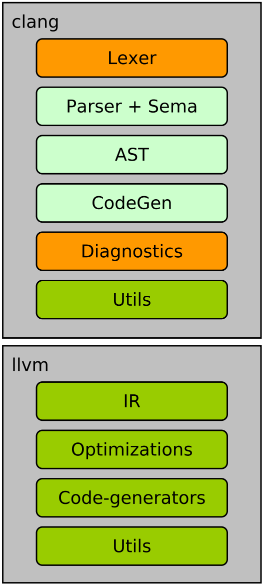
c-file
parse file
analyse file
generate .o file

target
link file.o's

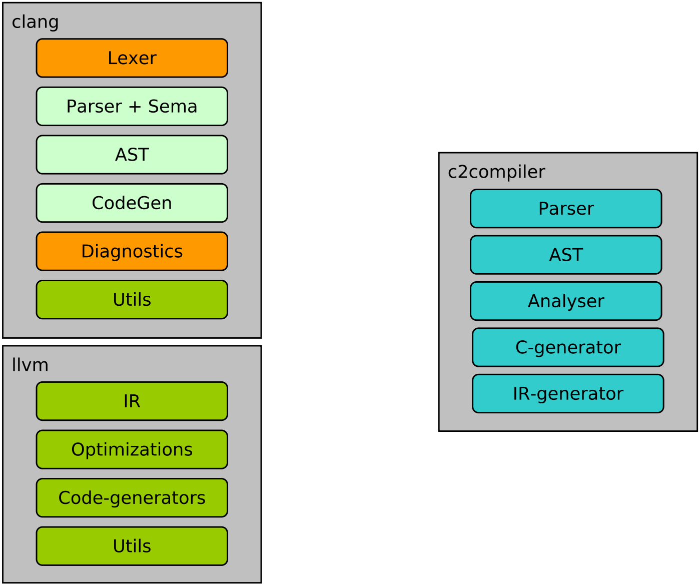
target
parse file
parse file
parse file
parse file
analyse file
analyse file
analyse file
analyse file
generate code per module
generate code per module
link module.o's

- C: a new compiler is started for each .c file
- C2 finds a compile error in file x much faster
- C2 generates code per module, not file
- The generation(+ optimization) step takes much longer then the parse/analyse step, so the yellow blocks are really much bigger

C2 compiler internals



C2 compiler internals



clang

Lexer

Parser + Sema

AST

CodeGen

Diagnostics

Utils

llvm

IR

Optimizations

Code-generators

Utils

c2compiler

Parser

AST

Analyser

C-generator

IR-generator

Experiences with LLVM/Clang

- it moves *fast*

Experiences with LLVM/Clang

- it moves *fast*
- mailing list is very friendly and helpful

Experiences with LLVM/Clang

- it moves *fast*
- mailing list is very friendly and helpful
- it achieves its design goal of having reusable components

Experiences with LLVM/Clang

- it moves *fast*
- mailing list is very friendly and helpful
- it achieves its design goal of having reusable components
- integration with build system tricky

Experiences with LLVM/Clang

- it moves *fast*
- mailing list is very friendly and helpful
- it achieves its design goal of having reusable components
- integration with build system tricky
- mapping your AST to LLVM IR is difficult

C2 current state

- Parser
- Analyser
- C generator
- IR codegen
- Building
- Tooling

C2 open issue: unified member access

foo.c2

```
type Point struct {
    uint32 x;
    uint32 y;
}

func void foo(Point* p) {
    p->x = 10;
    p.x = 10;

    a->child.member->name = "abc";
    a.child.member.name = "abc";
}
```

⟹ also see discussion on Forum

C2 open issue: foreign function interface (FFI)

Interface between C and C2

from/to	C	C2
C	working somewhat ;)	C2C generates C header file, no problem
C2	C2C needs to parse C headers and store in own interface format, TBD	C2C needs to parse interface format, TBD

⟹ Ideas/throught on interface format are welcome!

C2 open issue: solving the 32/64 bit issue

What is needed to 'solve' the 32/64-bit issue?

C2 open issue: solving the 32/64 bit issue

What is needed to 'solve' the 32/64-bit issue?

- printf formatters?
- size_t?
- ptrdiff_t?
- intptr_t?
- uintptr_t?

⟹ any other *issues* people run into?

C2 open issue: semantic macros

macro (idea)

```
macro max (x, y) {
    (x > y) x : y
}

func int foo() {
    int a = 2;
    int b = 3;
    int c = max!(a, b);
    return c;
}
```

⟹ must be correct C2 before expansion

⟹ do we need to distinguish between function calls and macros?

Future

Plans for 2015:

- rebase on LLVM/Clang 3.6 (and beyond)
- external libraries (C and C2)
- new recipe file format (toml?)
- c2reto
- semantic macros
- attribute syntax
- external tooling (vim syntax, bash completion, etc)
- more IR generation
- begin design of linker integration (lld)
- <your idea here>

Links

www.c2lang.org

http://github.com/c2lang/c2compiler

Let's create an even better C!