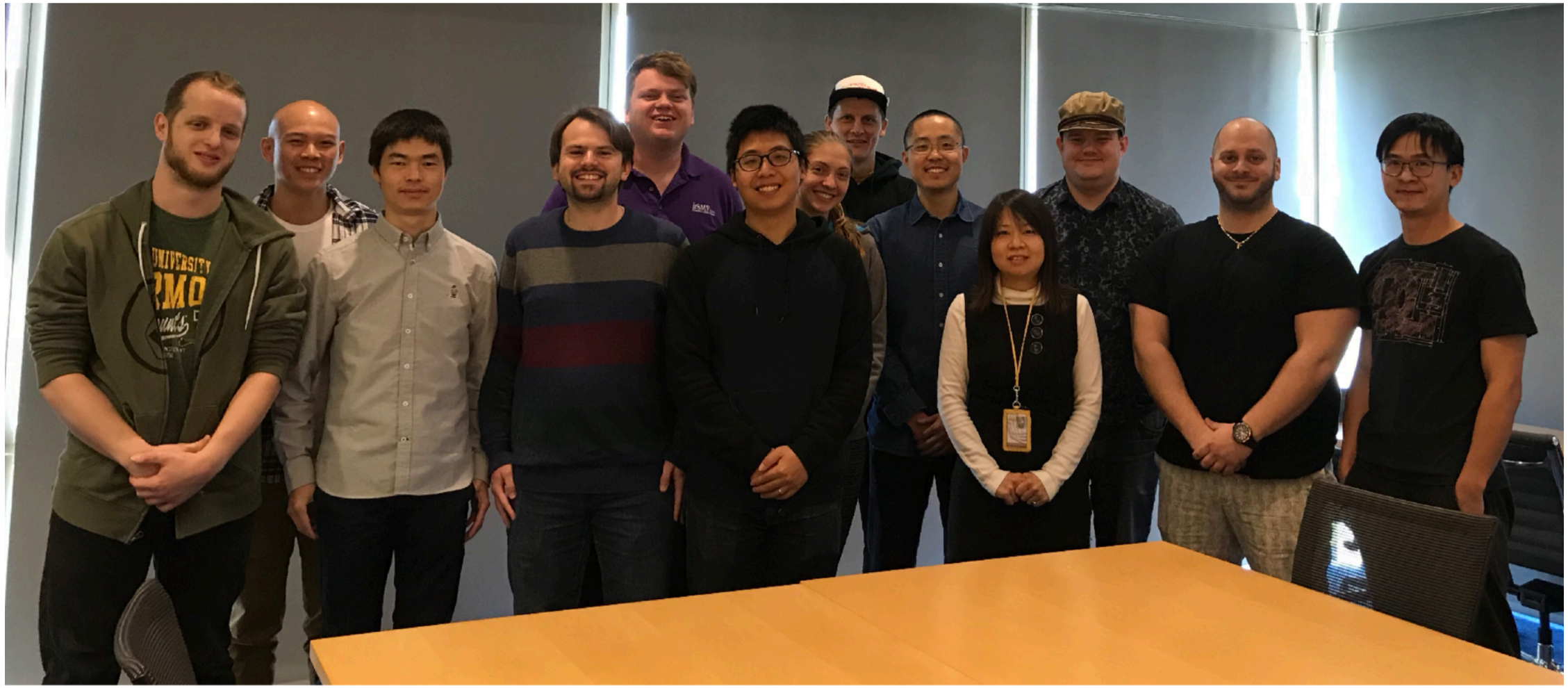
Schafmeister Group

Abiotic **B**iological **C**hemistry - ABC

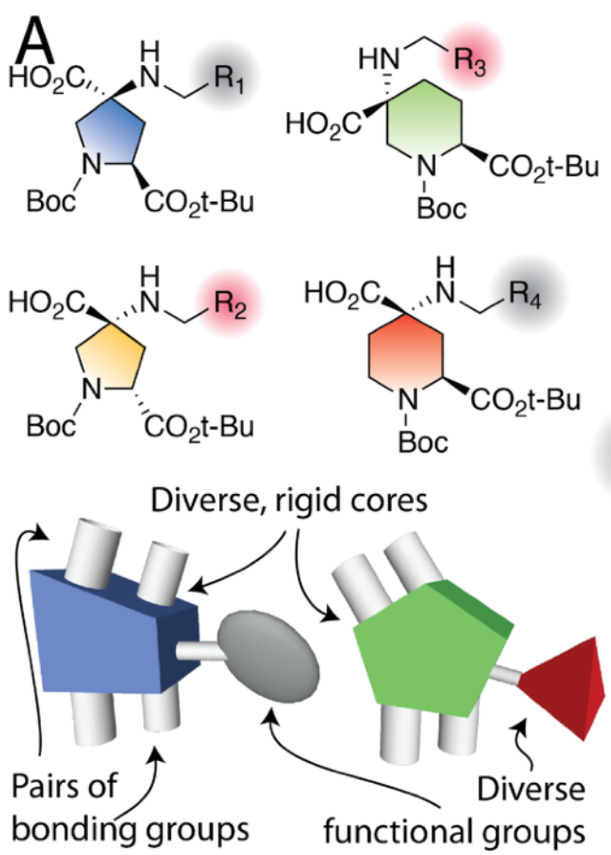
Nanometer scale - Angstrom precision

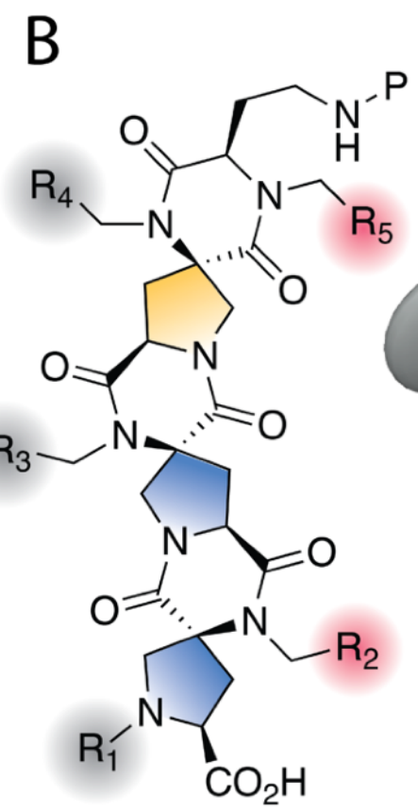


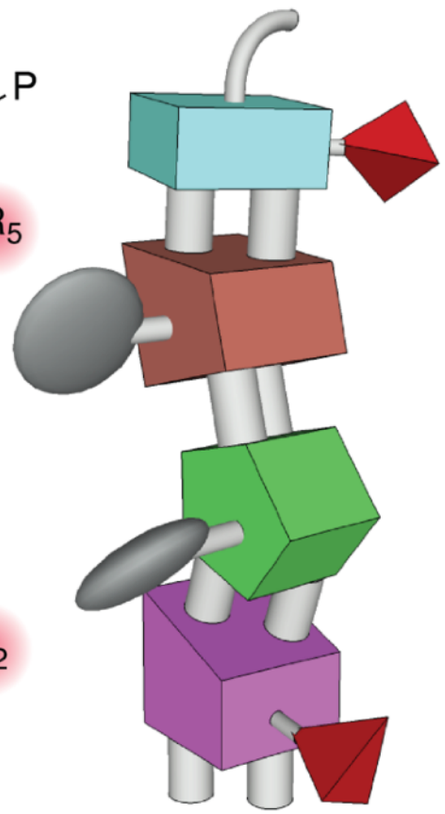
FUNDING

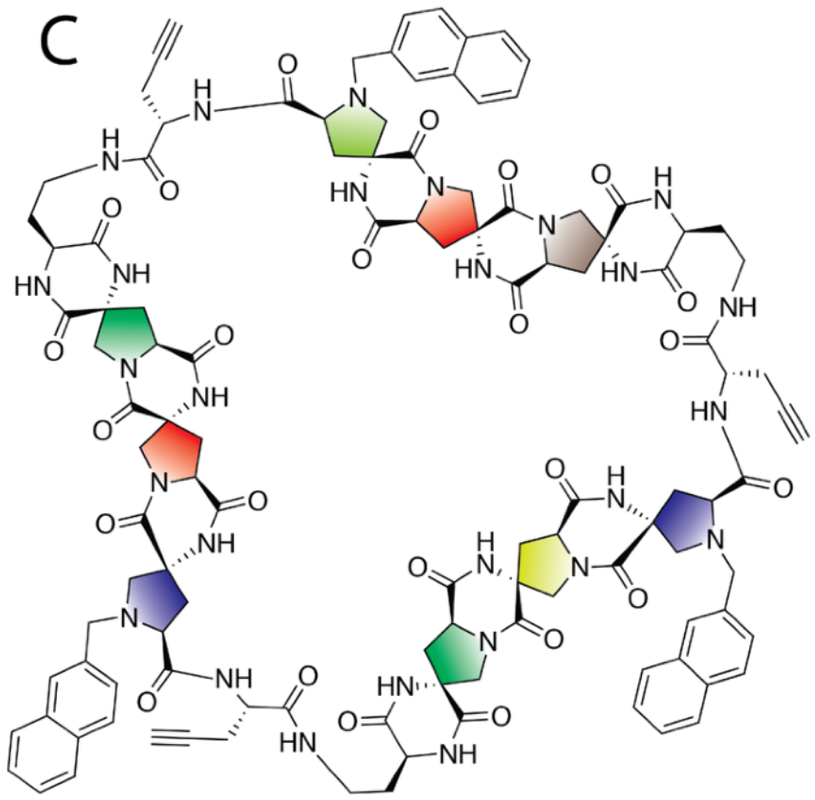
National Institute of Health/NIGMS
National Science Foundation,
Department of Energy,
Army Corp of Engineers,
Defense Threat Reduction Agency (DOD-DTRA),
Temple University

Molecular Lego

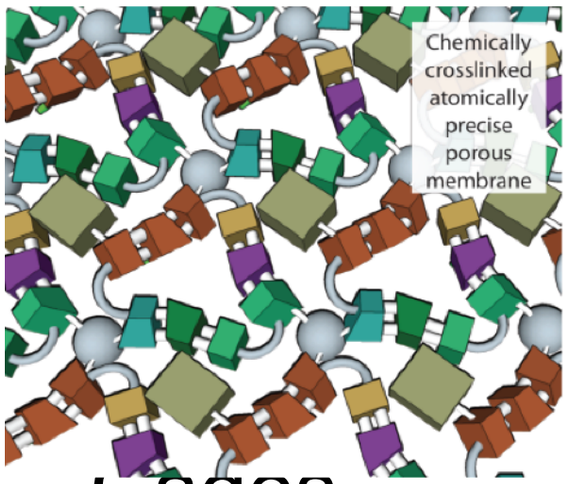




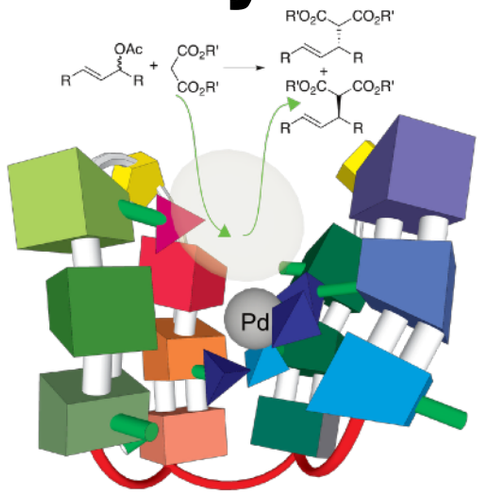




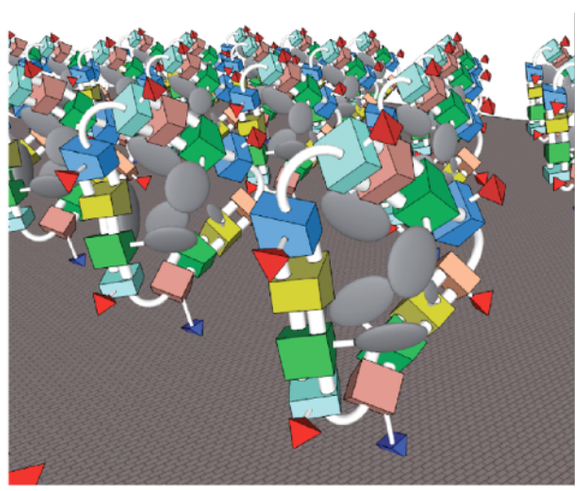
Filters



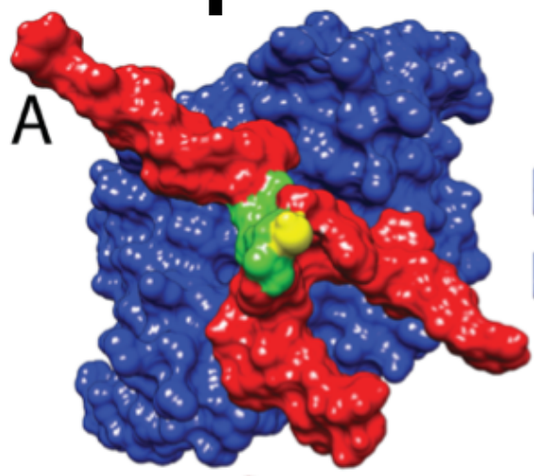
Catalysts



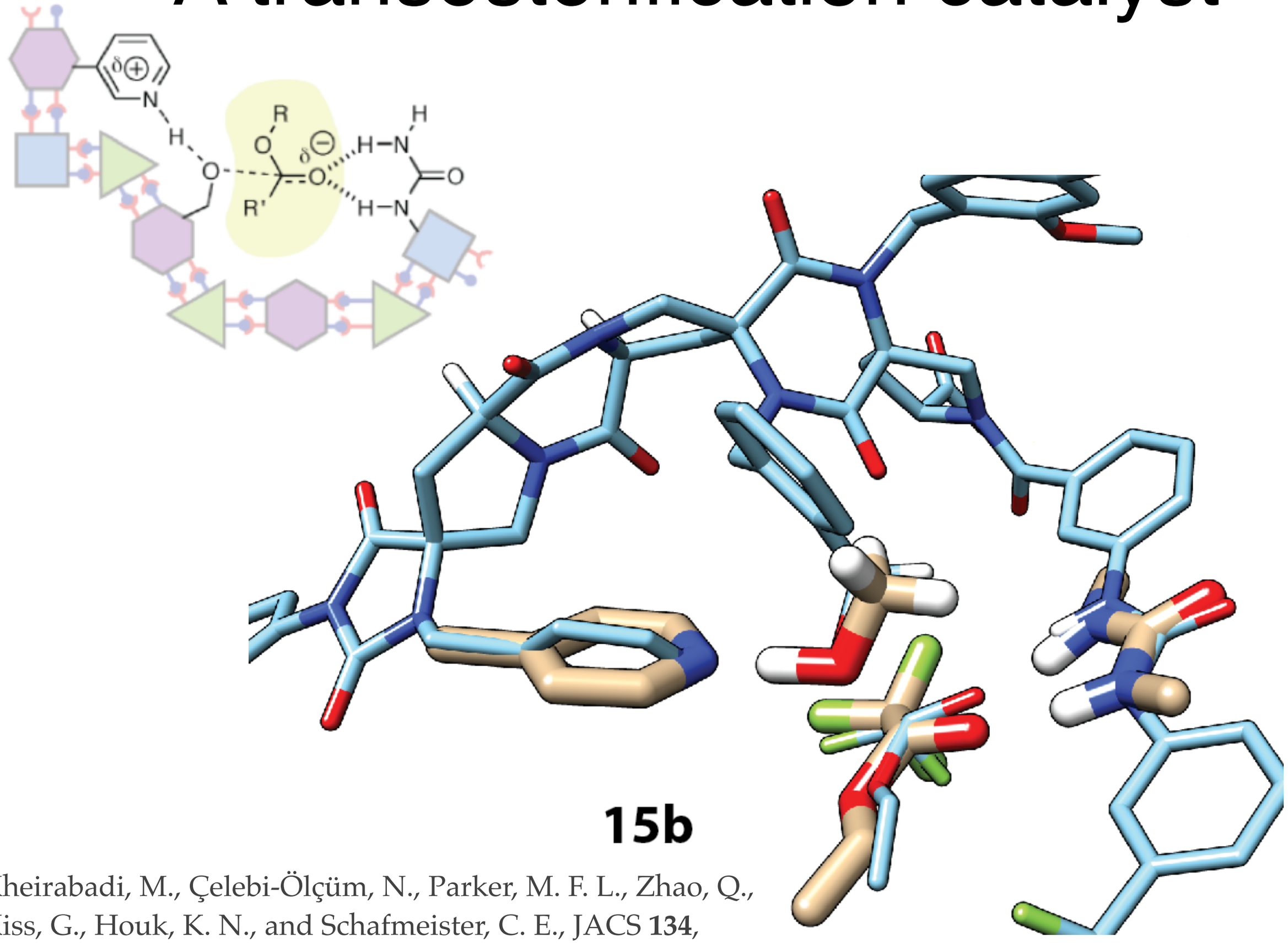
Sensors



Therapeutics

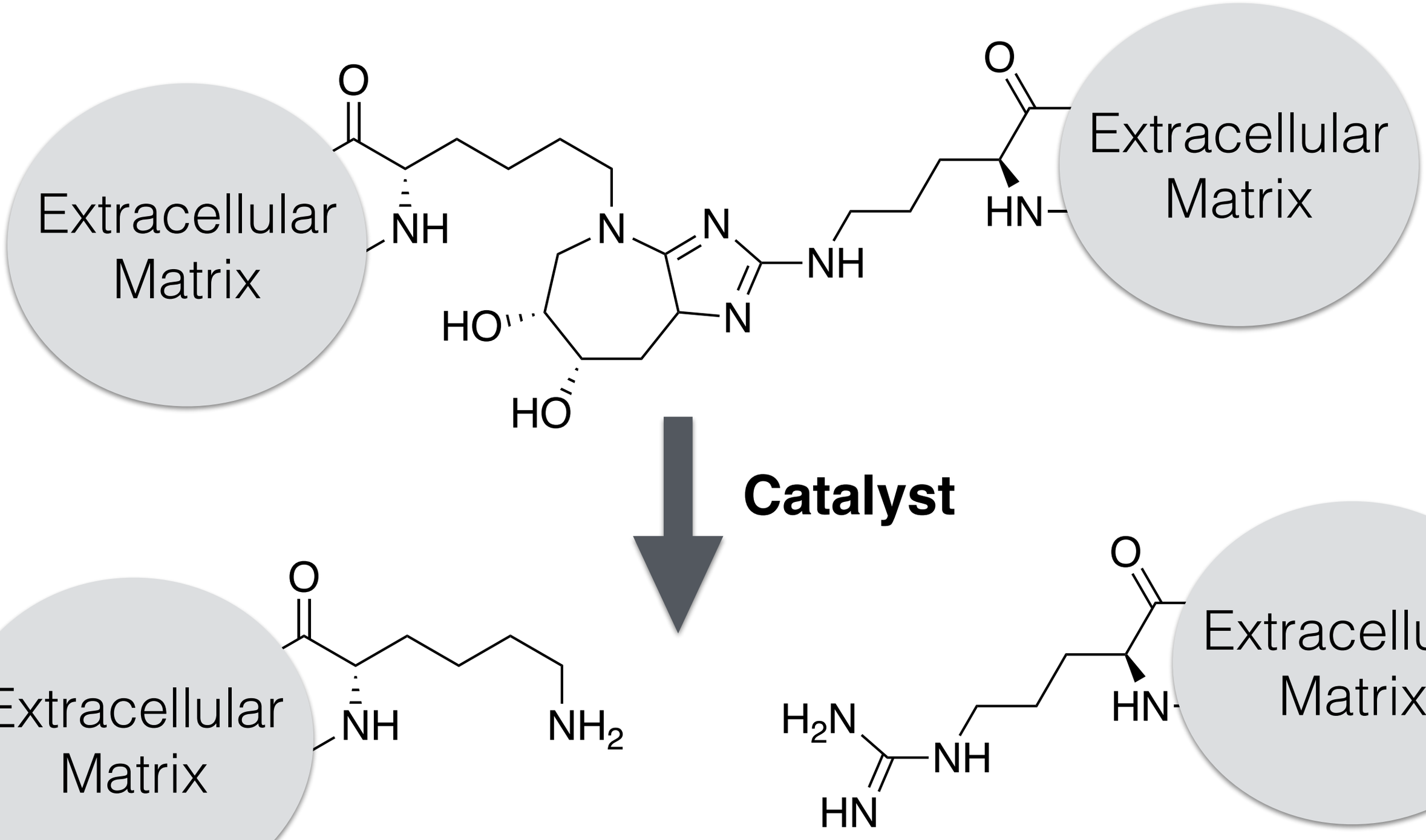


A transesterification catalyst



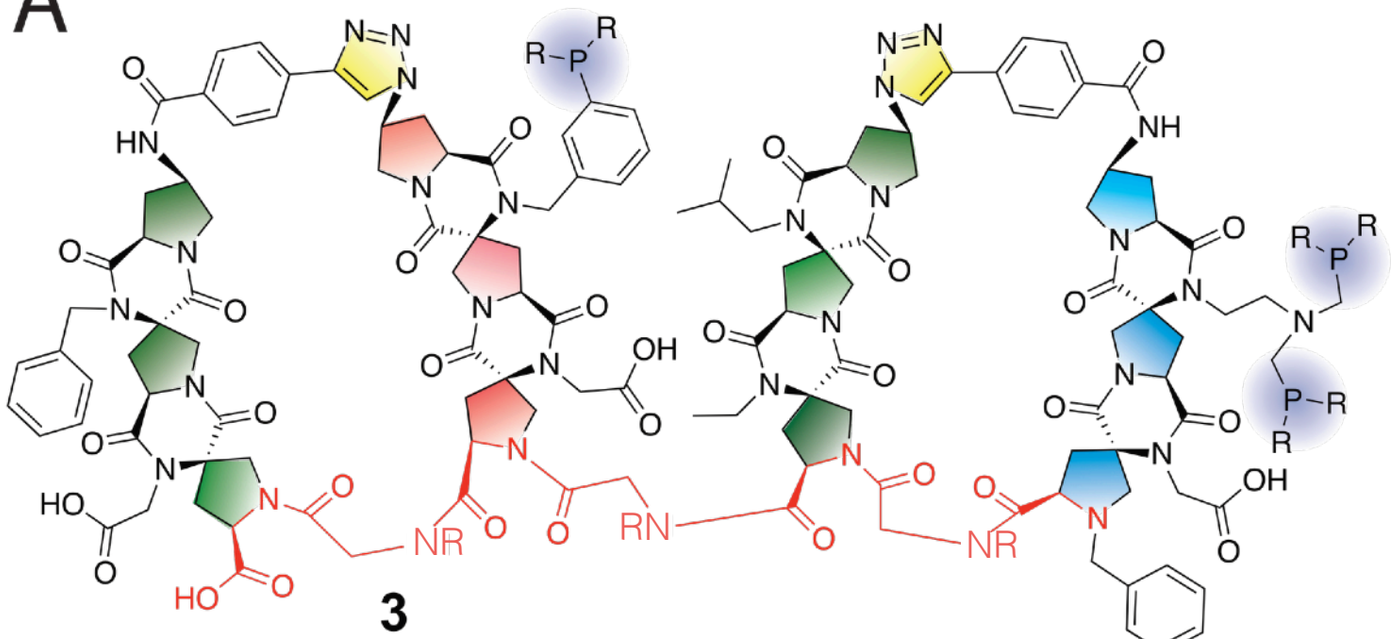
Kheirabadi, M., Çelebi-Ölçüm, N., Parker, M. F. L., Zhao, Q., Kiss, G., Houk, K. N., and Schafmeister, C. E., JACS **134**, (2012): 18345–18353. doi:10.1021/ja3069648,

Low MW Catalysts, *in situ* Cleavage, Advanced Glycation End products



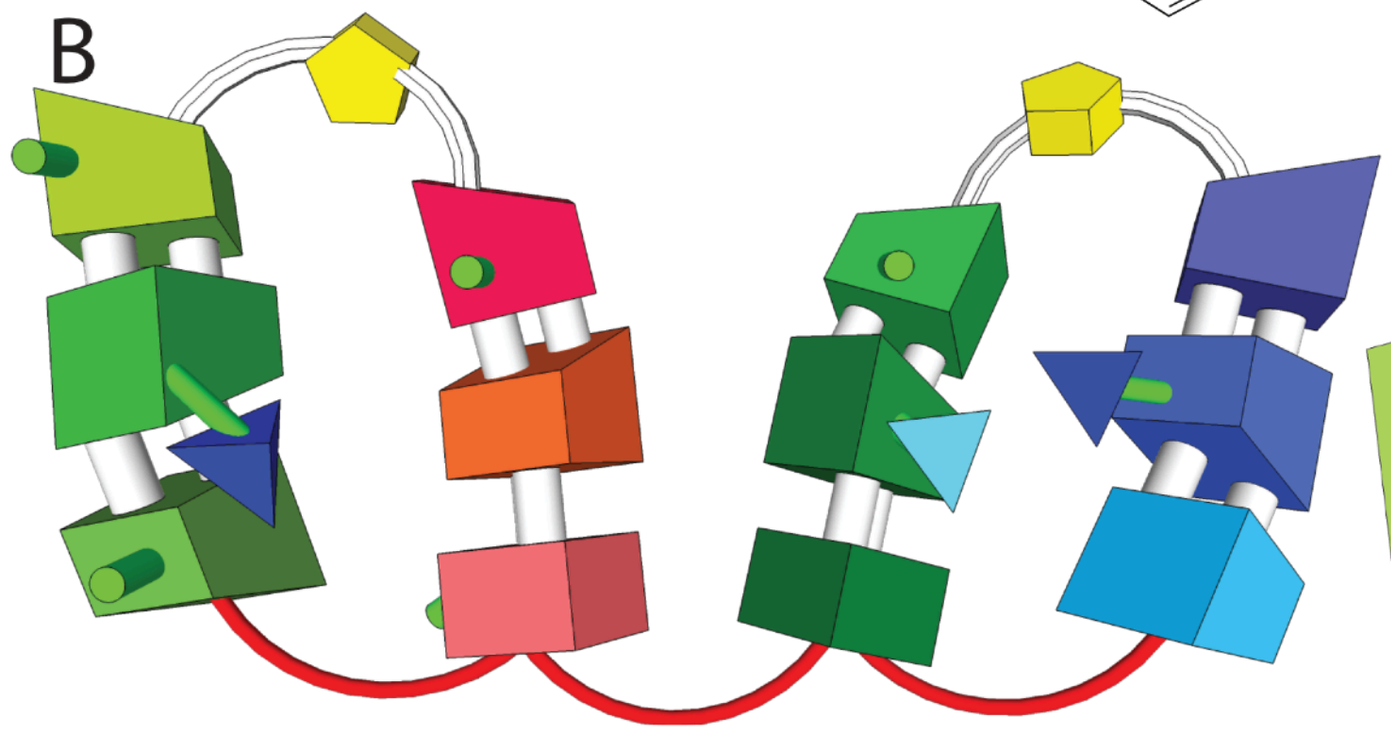
Atomically Precise Catalysts

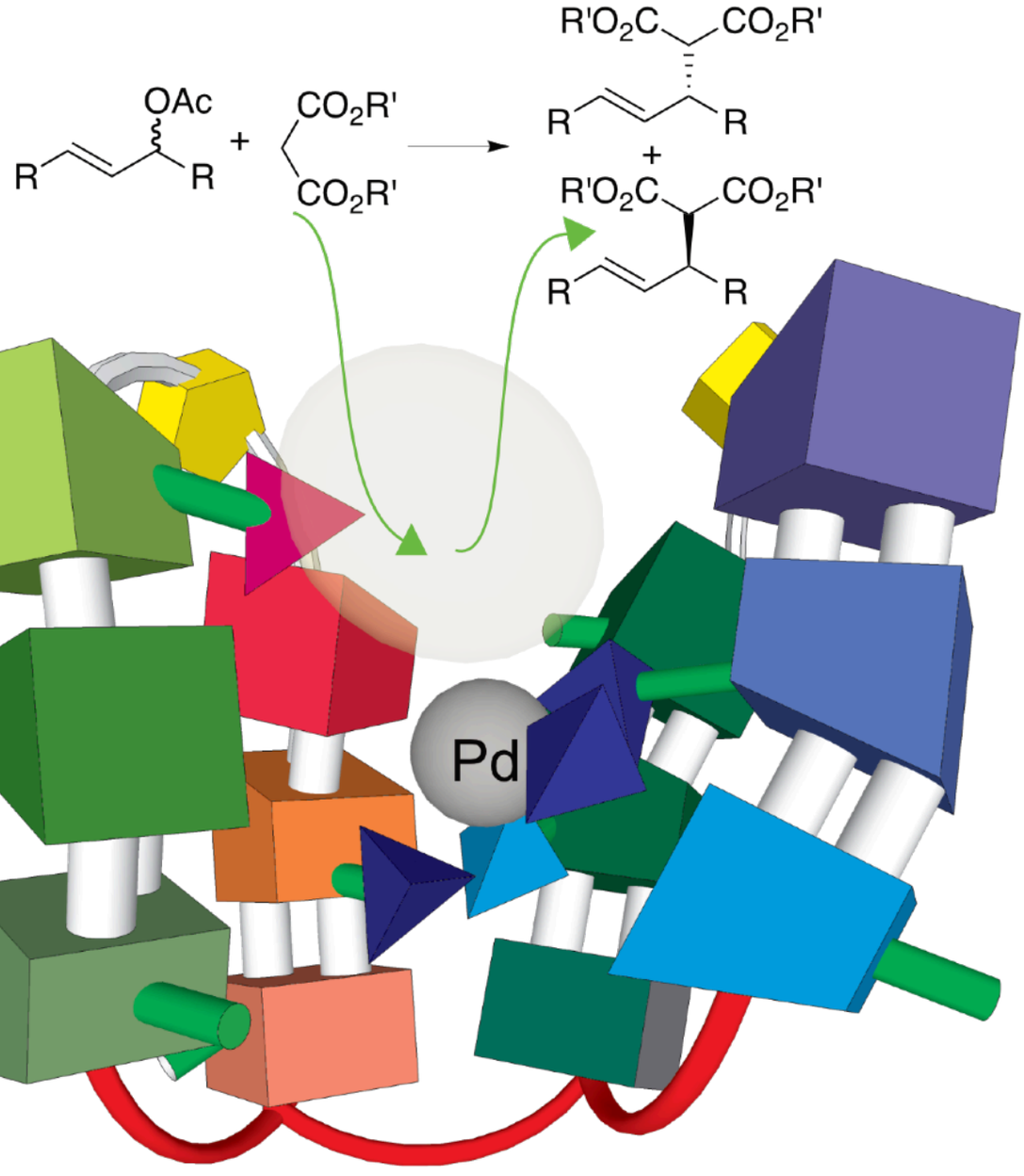
A



3

B





I need an **oracle** to help me design molecules

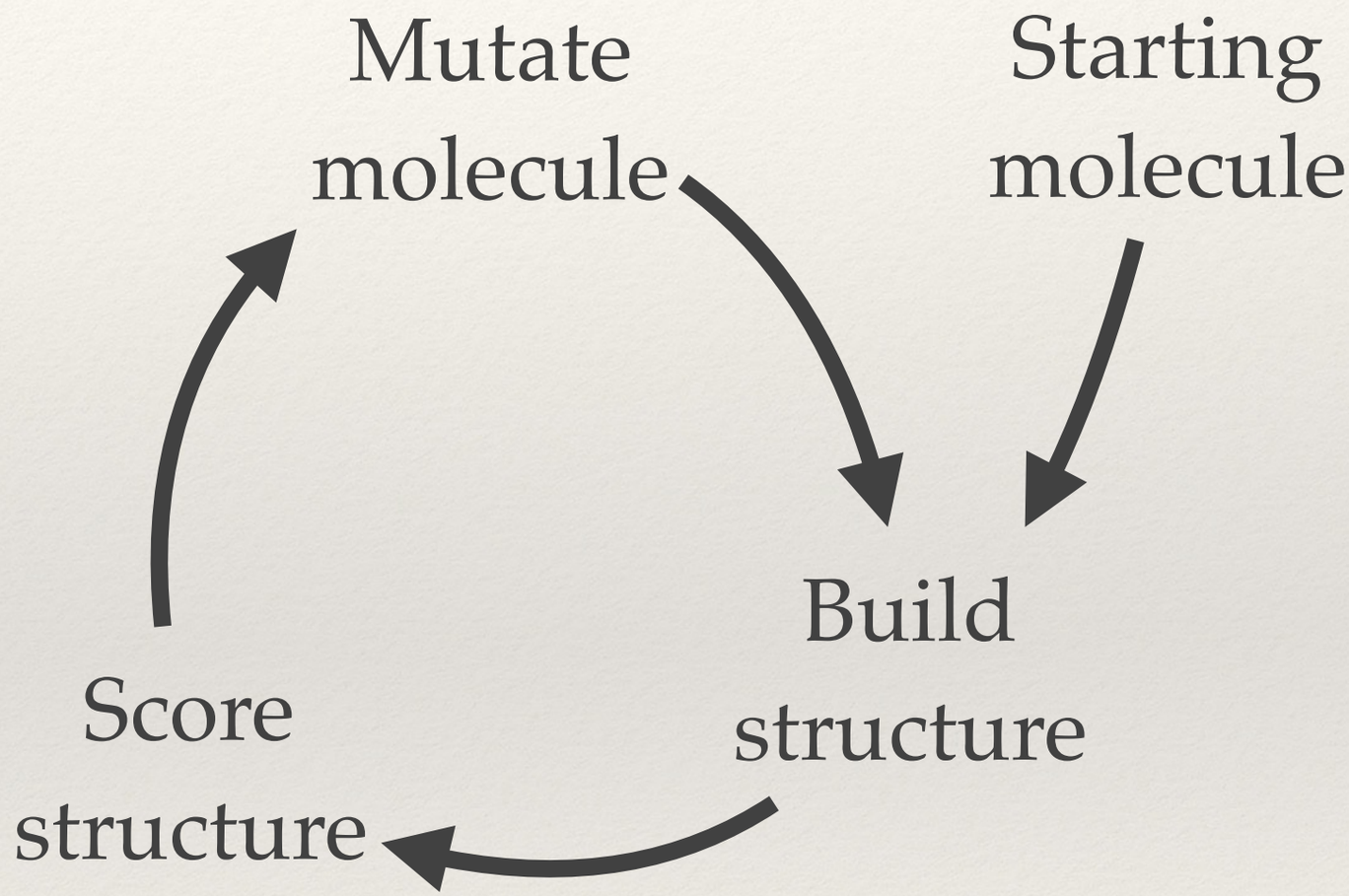
Requirements of a functional molecule

CANDO: Computer Aided Nanostructure Design and Optimization

Sequence of the molecule

Software to Design Molecules

Evolutionary molecular design



Specialized software with tightly coupled functions in the same processor

CANDO: A Molecular Design Environment

CANDO

- CANDO: Computer Aided Nanostructure Design and Optimization

Clasp: Common Lisp

- Extends Clasp with chemistry
- C++ interoperation - Use C++ Libraries

C++ and LLVM-IR

- LLVM backend - generate fast, cache-aware, native code

How Cando came to be

- Wrote 500,000 LOC C++ chemistry library
- Used boost::python to expose it to Python
- Keeping C++/Python working == Headache!
- Implemented a simple lisp* interpreter in C++
- Added “clbind” & exposed chemistry to lisp*

Expose C++ to Common Lisp

```
CL_LISPIFY_NAME(CreateLandingPad);
CL_EXTERN_DEFMETHOD(IRBuilder_O, &llvm::IRBuilder::CreateLandingPad);

CL_LISPIFY_NAME(addClause);
CL_EXTERN_DEFMETHOD(LandingPadInst_O,
&llvm::LandingPadInst::addClause);
```

```
core::externalClass_<IRBuilder_O>()
   .def(core::magic_name("LLVM-SYS:CreateLandingPad"),
                        &llvm::IRBuilder::CreateLandingPad,””,””);

core::externalClass_<LandingPadInst_O>()
   .def(core::magic_name("LLVM-SYS:addClause"),
                        &llvm::LandingPadInst::addClause,””,””);
```

```
(defun irc-generate-unwind-protect-landing-pad-code ()
     (let* ((landpad (llvm-sys:create-landing-pad *irbuilder* %exception-struct% 1)))
       (llvm-sys:add-clause landpad (llvm-sys:constant-pointer-null-get %i8*%))))
```

Why Common Lisp

- **Macros and compile-time computing.**

- Closures and first class functions, optional typing, correct implementation of lexical scope, dynamically scoped variables, a meta-object protocol, conditions & restarts, a compiler that is available for customization, multiple value returns, symbols, union types, control when functions are run at compile time, load time or run time, a programmable reader, bignums, rationals, complex numbers and bit vectors built in, specialized array types, automatic memory management, **community of programmers and libraries**.

- **Language standard** that is **timeless**.

Energy Efficiency across Programming Languages

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

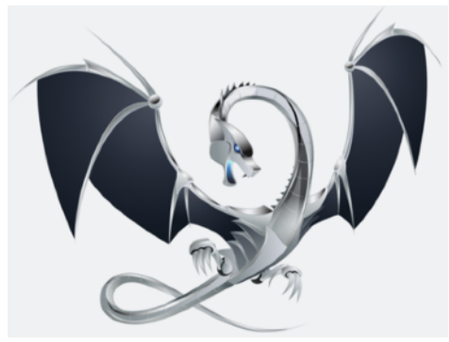
Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2017). Energy efficiency across programming languages: how do energy, time, and memory relate? (pp. 256–267). Presented at the the 10th ACM SIGPLAN International Conference, New York, New York, USA: ACM Press. http://doi.org/10.1145/3136014.3136031

Table 1: Calculating the 78th Fibonacci number 10^7 times

Language	Seconds	Factor	Stdev(sec)	Rel. cclasp
clang C++	0.76	1	0.016	0.3
sbcl 1.2.11	0.63	1	0.008	0.2
cclasp	2.9	4	0.022	1.0
Python 2.7	77.7	102	0.36	26.8
bclasp	639	839	n/c	221

Pass 21,265 of 21,734 (97.8%)
of ansi tests for Common Lisp

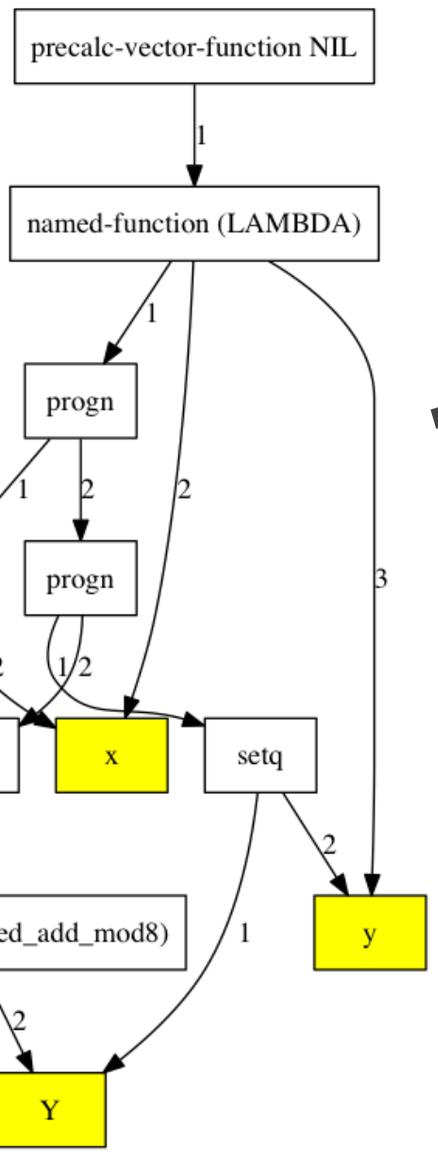
LLVM: A need for speed

- LLVM 3.4 -> 6.0 (chasing the )
  - Keep up to date - it's not hard and it's best.
- Kaleidoscope demo was very helpful.
- Exposed LLVM C++ API 907 symbols total,
  100 llvm classes, 423 llvm functions/methods.
- Common Lisp requires a new AST and HIR.
- Cleavir (Robert Strandh, Alex Wood)
  - cutting edge dynamic language compiler.

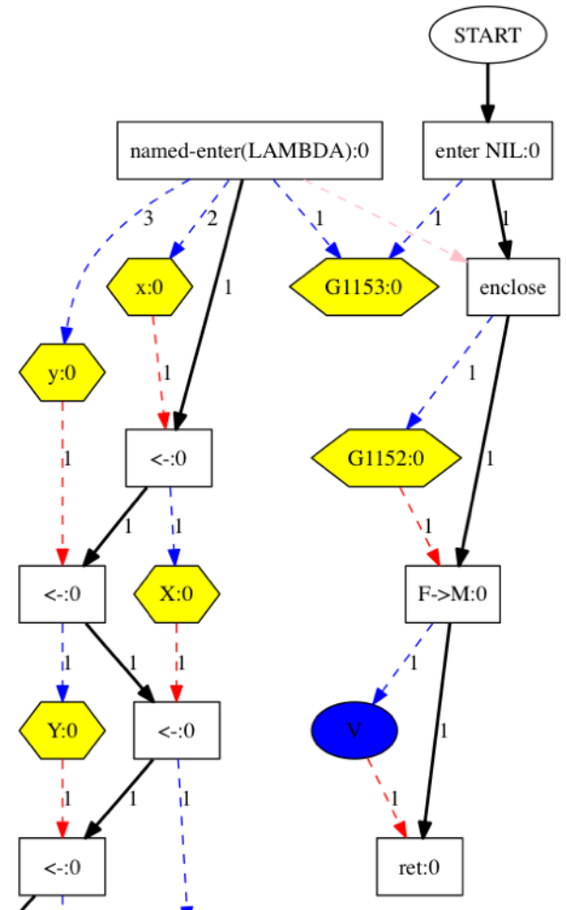
Cleavir: Sea of Nodes Compiler

```
(clasp-cleavir:cleavir-compile
 'foo
 '(lambda (x y)
    (intrinsic-call "wrapped_add_mod8" x y))
 :debug t)
```

AST



HIR



LLVM-IR

LLVM-IR from Generic Functions

```
;; Test for tagged character
(defmethod translate-branch-instruction
    (((instruction cc-mir:characterp-instruction) return-value
     inputs outputs successors (abi abi-x86-64) function-info)
  (let* ((value   (%load (first inputs)))
         (tag     (%and (%ptrtoint value cmp:%uintptr_t%)
                        (%uintptr_t cmp:+immediate-mask+) "character-tag-only"))
         (cmp (%icmp-eq tag (%uintptr_t cmp:+character-tag+))))
    (%cond-br cmp (first successors) (second successors) :likely-true t)))
```

(abi gpu)

```
;; Test for tagged single-float
(defmethod translate-branch-instruction
    (((instruction cc-mir:single-float-p-instruction) return-value
     inputs outputs successors (abi abi-x86-64) function-info)
  (let* ((value   (%load (first inputs)))
         (tag     (%and (%ptrtoint value cmp:%uintptr_t%)
                        (%uintptr_t cmp:+immediate-mask+) "single-float-tag-only"))
         (cmp     (%icmp-eq tag (%uintptr_t cmp:+single-float-tag+))))
    (%cond-br cmp (first successors) (second successors) :likely-true t)))
```

(abi gpu)

GPU Kernels in a subset of Common Lisp

$$V(r^N) = \sum_{\text{bonds}} k_b(l - l_0)^2 + \sum_{\text{angles}} k_a(\theta - \theta_0)^2$$

$$+ \sum_{\text{torsions}} \frac{1}{2} V_n[1 + \cos(n\omega - \gamma)]$$

$$+ \sum_{j=1}^{N-1} \sum_{i=j+1}^{N} \left\{ \epsilon_{i,j} \left[\left(\frac{r_{0ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{r_{0ij}}{r_{ij}} \right)^{6} \right] + \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}} \right\}$$

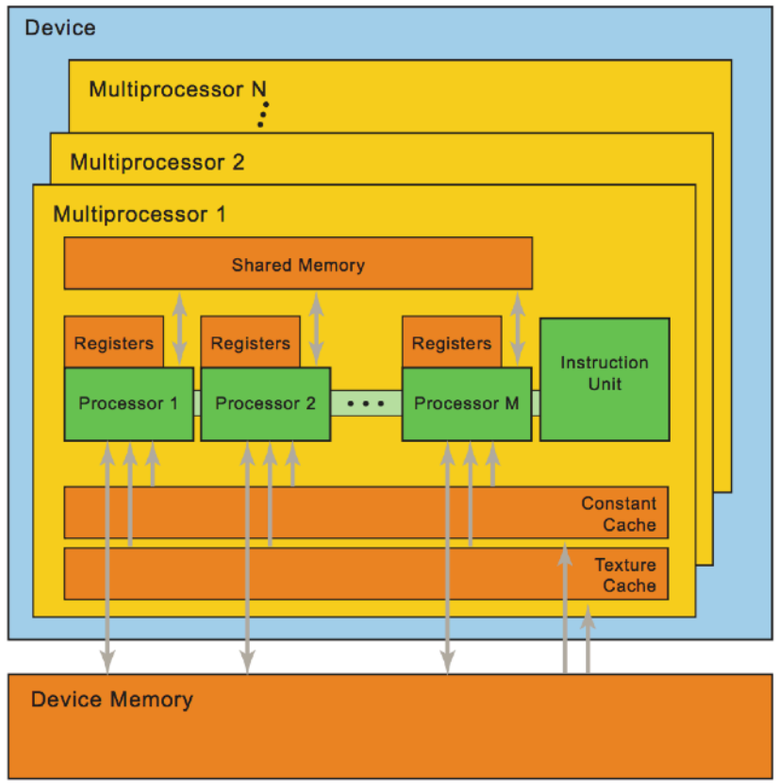
AMBER force field

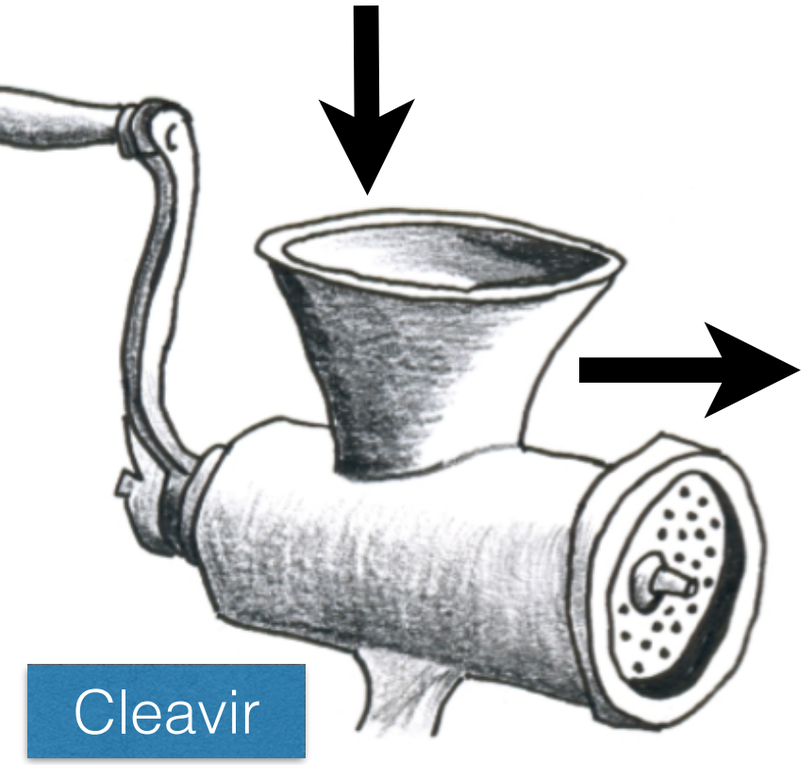
```
(%i5) diff(k*(sqrt((x1-x0)^2+(y1-y0)^2+(z1-z0)^2)-r0)^2,x1,1);
```

$$(\%o5) \quad \frac{2\,k\,(x1-x0)\left(\sqrt{(z1-z0)^2+(y1-y0)^2+(x1-x0)^2}-r0\right)}{\sqrt{(z1-z0)^2+(y1-y0)^2+(x1-x0)^2}}$$

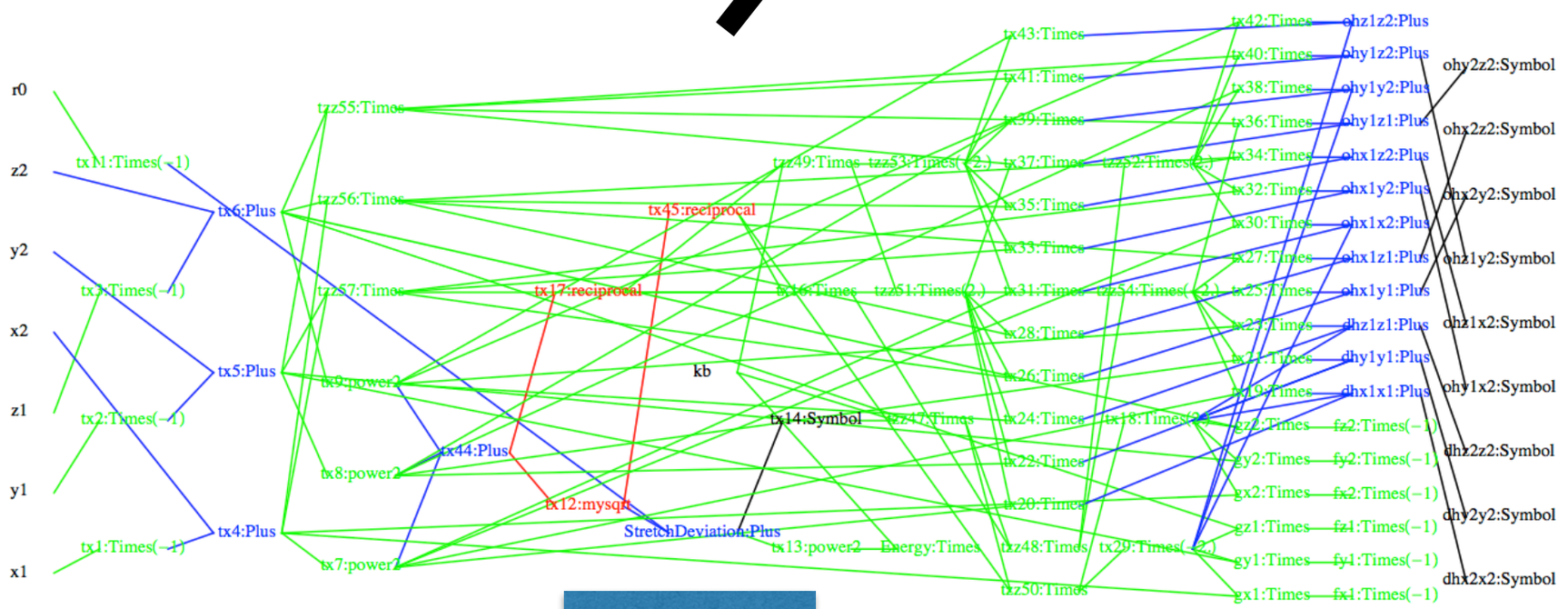
Maxima (Common Lisp)

GPU





Cleavir



Math

ASTMatcher & Compacting GC

- Cando uses Memory Pool System GC or Boehm GC
- Inspired by Chandler Carruth's talk on ASTMatcher
- Exposed 273 Clang classes(AST) & ASTMatcher
- Static analyzer analyzes 381 C++ source files
- Identifies 1,167 pointers in 743 C++ classes

```
class Cons_O : public T_O {
    // ...
     T_sp _Car;
     T_sp _Cdr;
```

```
{ class_kind, STAMP_core__Cons_O, sizeof(core::Cons_O), 0, "core::Cons_O" },
  { fixed_field, SMART_PTR_OFFSET, sizeof(gctools::smart_ptr<core::T_O>),
      offsetof(SAFE_TYPE_MACRO(core::Cons_O),_Car), "_Car" },
{ fixed_field, SMART_PTR_OFFSET, sizeof(gctools::smart_ptr<core::T_O>),
      offsetof(SAFE_TYPE_MACRO(core::Cons_O),_Cdr), "_Cdr" },
```

```lisp
(defun reciprocal (x)
  (if (= x 0)
      (error "Divide by zero")
      (/ 1.0 x)))
(defun vector-sum-reciprocal (values)
    (loop for val across values
          sum (reciprocal val)))
(vector-sum-reciprocal #(5 0 6))
```

```
(load (compile-file "sys:tests;tb.lsp"))
; Compiling file: tb.lsp
;   (DEFUN RECIPROCAL ...)
;   (DEFUN VECTOR-SUM-RECIPROCAL ...)
;   (VECTOR-SUM-RECIPROCAL #(5 0 ...))
Writing temporary bitcode file to: #P"SYS:TESTS;TB.BC.NEWEST"
Writing fasl file to: #P"SYS:TESTS;TB.FASL.NEWEST"
Compile-file seconds real(0.8) run(0.8)
                     llvm(0.0) link(0.0)
                     (llvm+link)/(real+link)(1%)

Condition of type: SIMPLE-ERROR
Divide by zero
Available restarts:
(use :r1 to invoke restart 1)

1. (RESTART-TOPLEVEL) Go back to Top-Level REPL.

Broken at frame[142] LAMBDA. In: #<PROCESS TOP-LEVEL @0x10b950ef9>.
 File: #P"top.lsp" (Position #21213)
COMMON-LISP-USER>> :b
0: (UNIVERSAL-ERROR-HANDLER NIL "Divide by zero" NIL)
1: (ERROR "Divide by zero")
2: (RECIPROCAL 0)
3: (VECTOR-SUM-RECIPROCAL #(5 0 6))
4: (TB.LSP^8^TOP-COMPILE-FILE)
5: (LOAD-BINARY #P"tb.fasl" NIL NIL :DEFAULT)
6: (LOAD #P"SYS:TESTS;TB.FASL.NEWEST")
```

Really want access to DWARF from Cando runtime

Expression-level Debug Info

C++

```
frame #12: 0x00000001063c612c iclasp-boehm`core::cl__peek_char(peek_type=<unavailable>, strm=(theObject = 0x000000010b1e
stream.cc:5581 [opt]
   5578   if (!clasp_input_stream_p(strm))
   5579     SIMPLE_ERROR(BF("Not input-stream"));
   5580   if (peek_type.nilp()) {
-> 5581     int c = clasp_peek_char(strm);
   5582     if (c == EOF)
   5583       goto HANDLE_EOF;
   5584     return clasp_make_character(clasp_peek_char(strm));
(lldb)
```

C++

```
frame #13: 0x0000000105d88729 iclasp-boehm`::wrapped_cl__peek_char_T_spT_spT_spT_spT_sp(in0=<unavailable>, in1=<unavaila
   3612                       translate::from_object<core::T_sp> a3 (core::T_sp((gc::Tagged)(in3)));
   3613                       translate::from_object<core::T_sp> a4 (core::T_sp((gc::Tagged)(in4)));
   3614
-> 3615     core::T_sp ret = core ::cl__peek_char(a0._v,a1._v,a2._v,a3._v,a4._v);
                            ^
   3616      return translate::to_object<core::T_sp>::convert(ret).as_return_type();
   3617     }
   3618 // Generating code for core::cl__read_byte
(lldb)
```

CL

```
frame #14: 0x000000010c082a30 cclasp-boehm-image.fasl`::PEEK-CHAR^COMMON-LISP^FN^^() at cl-wrappers.lisp:893
   890  ;;; Generating code for core::cl__read_char
   891  (wrap-c++-function (core:magic-intern "cl__read_char") () (&optional strm (eof-error-p t) eof-value recursive-p)
   892  ;;; Generating code for core::cl__peek_char
-> 893  (wrap-c++-function (core:magic-intern "cl__peek_char") () (&optional peek-type strm (eof-errorp t) eof-value rec
        ^
   894  ;;; Generating code for core::cl__read_byte
   895  (wrap-c++-function (core:magic-intern "cl__read_byte") () (&optional strm (eof-error-p t) eof-value) "wrapped_cl
   896  ;;; Generating code for core::core__input_stream_source_pos_info
(lldb)
```

CL

```
frame #15: 0x000000010f1b5029 cclasp-boehm-image.fasl`::LAMBDA^COMMON-LISP^FN^^() at top.lsp:633
   630  (defun tpl-read (&aux (*read-suppress* nil))
   631    (finish-output)
   632    (loop
-> 633      (case (peek-char nil *standard-input* nil :EOF)
                 ^
   634        ((#\))
   635         (warn "Ignoring an unmatched right parenthesis.")
   636         (read-char))
(lldb)
```

Thanks
Eric Christopher

Link Time Optimization and LLVM toolchain compatible

C++ code

```
int add( int x, int y)
{ return x + y; };
```

Common Lisp code

```
(defun mul-add (x y z)
    (+ (* x y) z))
```

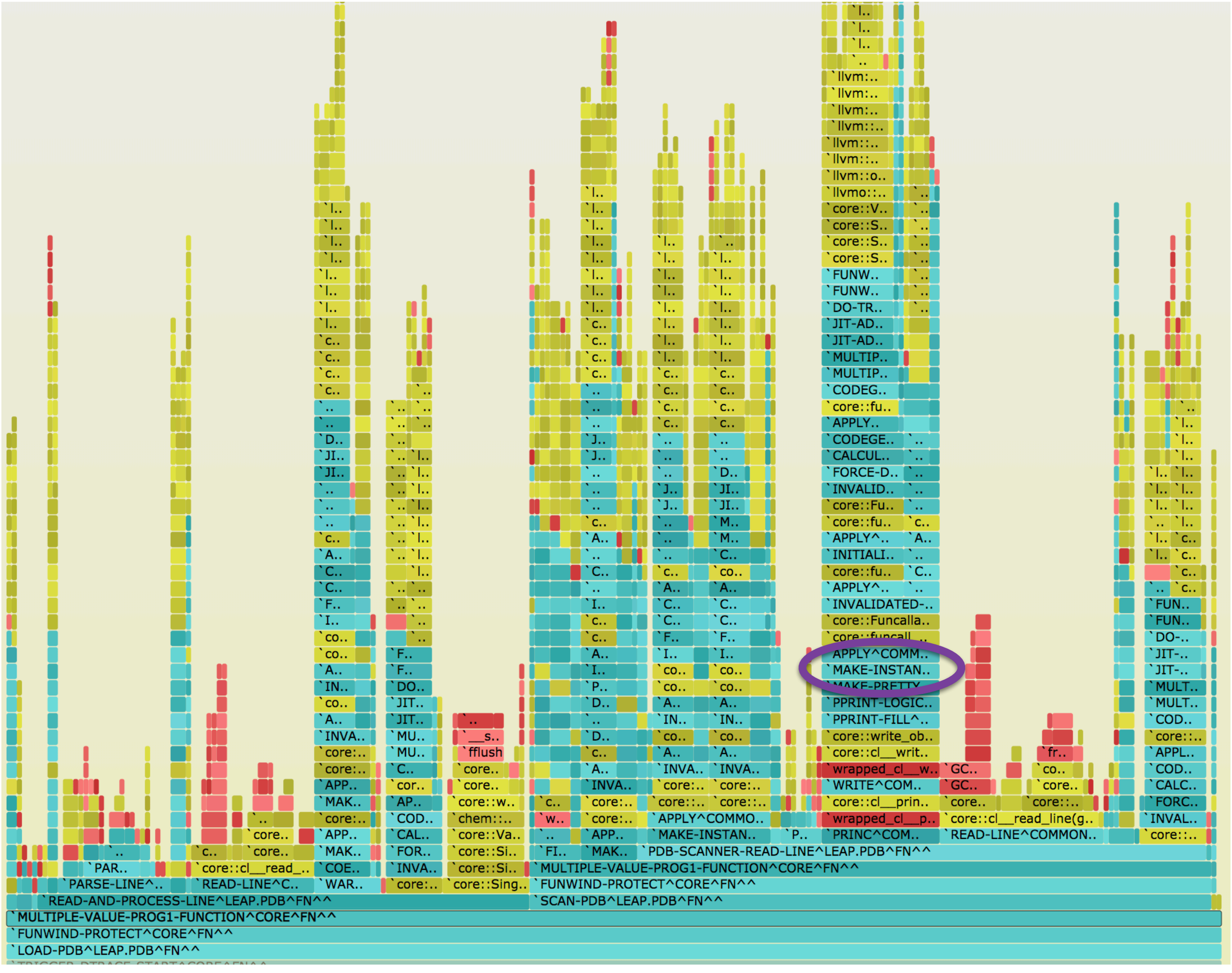
Thanks
Mehdi Amini!

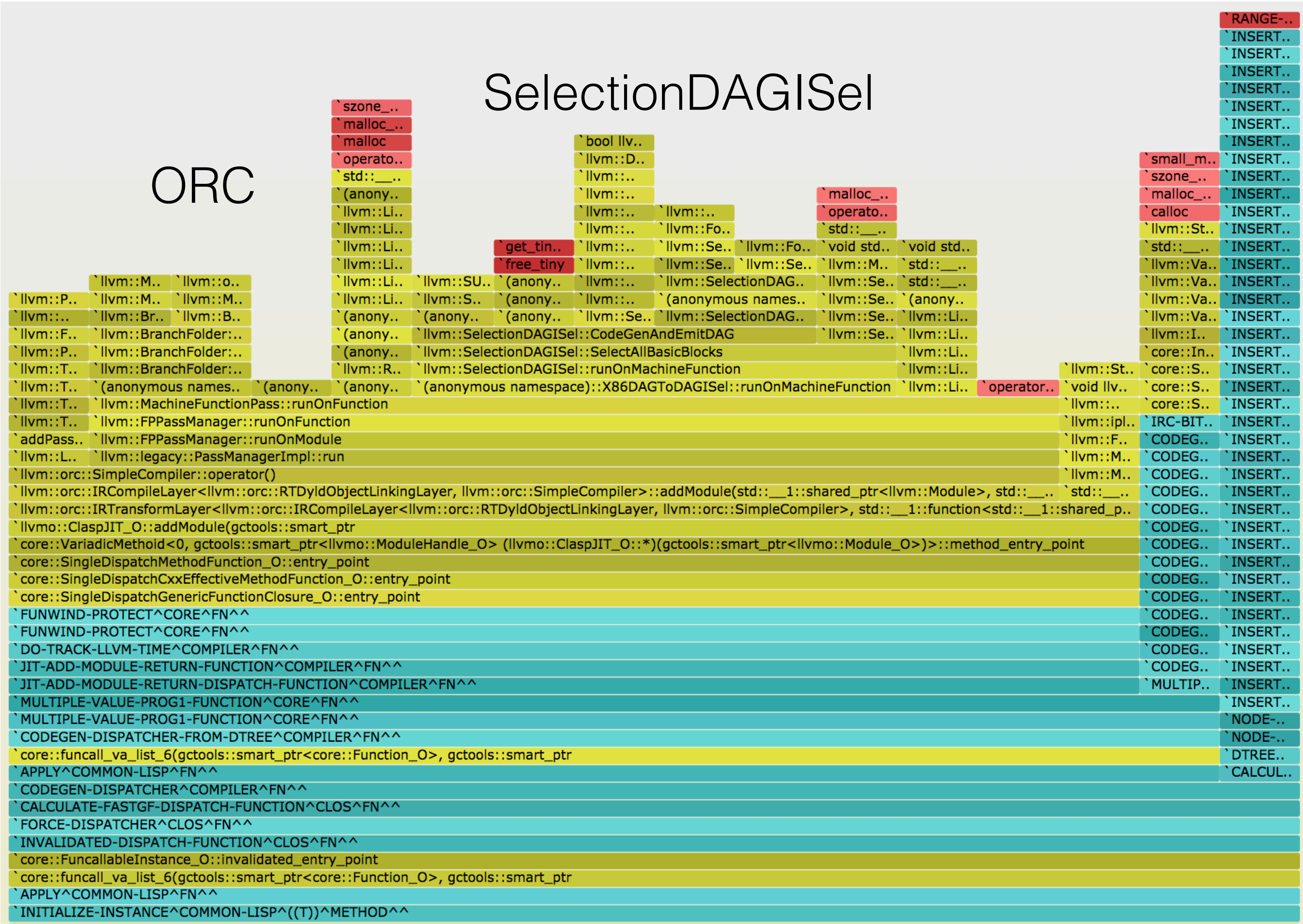
Optimized machine code

Other Features

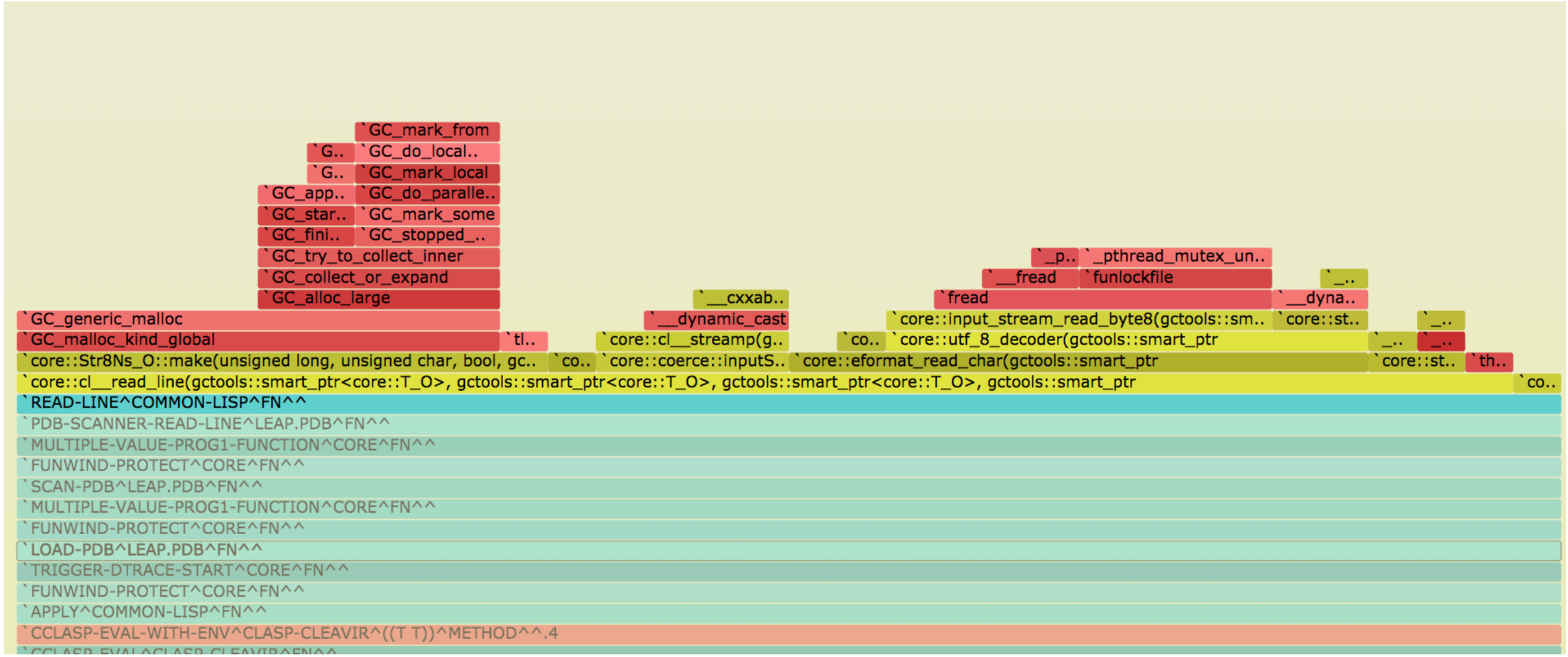
- Clasp's JIT uses ORC (thanks Lang Hames)
  - expressions
  - generic function dispatchers
- Multithreaded using pthreads
- Unwinding uses C++ exceptions (thanks Eric Christopher)
- Intrinsics written in C++, inlined into CL code
- Profile Common Lisp, C++ and C together

(core:with-dtrace-trigger (:= *pdb* (load-pdb "2zff_fixed.pdb")))


`READ-AND-PROCESS-LINE^LEAP.PDB^FN^^
`SCAN-PDB^LEAP.PDB^FN^^
`MULTIPLE-VALUE-PROG1-FUNCTION^CORE^FN^^
`FUNWIND-PROTECT^CORE^FN^^
`LOAD-PDB^LEAP.PDB^FN^^
`PARSE-LINE^..
`READ-LINE^C..
`core::cl__read_..
`PDB-SCANNER-READ-LINE^LEAP.PDB^FN^^
`MAKE-INSTAN..
`APPLY^COMM..
`core::cl__prin..
`wrapped_cl__p..
`PRINC^COM..
`core::cl__read_line(g..
`READ-LINE^COMMON..



SelectionDAGISel
ORC
`RANGE-..
`INSERT..
`szone_..
`malloc_..
`malloc
`operato..
`std::__..
`(anony..
`bool llv..
`llvm::D..
`small_m..
`szone_..
`malloc_..
`calloc
`operato..
`get_tin..
`free_tiny
`llvm::SelectionDAG..
`(anonymous names..
`llvm::SelectionDAGISel::CodeGenAndEmitDAG
`llvm::SelectionDAGISel::SelectAllBasicBlocks
`llvm::SelectionDAGISel::runOnMachineFunction
`(anonymous namespace)::X86DAGToDAGISel::runOnMachineFunction
`operator..
`llvm::BranchFolder:..
`llvm::MachineFunctionPass::runOnFunction
`llvm::FPPassManager::runOnFunction
`llvm::FPPassManager::runOnModule
`llvm::legacy::PassManagerImpl::run
`addPass..
`llvm::orc::SimpleCompiler::operator()
`llvm::orc::IRCompileLayer<llvm::orc::RTDyldObjectLinkingLayer, llvm::orc::SimpleCompiler>::addModule(std::__1::shared_ptr<llvm::Module>, std::__..
`llvm::orc::IRTransformLayer<llvm::orc::IRCompileLayer<llvm::orc::RTDyldObjectLinkingLayer, llvm::orc::SimpleCompiler>, std::__1::function<std::__1::shared_p..
`llvmo::ClaspJIT_O::addModule(gctools::smart_ptr
`core::VariadicMethoid<0, gctools::smart_ptr<llvmo::ModuleHandle_O> (llvmo::ClaspJIT_O::*)(gctools::smart_ptr<llvmo::Module_O>)>::method_entry_point
`core::SingleDispatchMethodFunction_O::entry_point
`core::SingleDispatchCxxEffectiveMethodFunction_O::entry_point
`core::SingleDispatchGenericFunctionClosure_O::entry_point
`IRC-BIT..
`CODEG..
`MULTIP..
`NODE-..
`DTREE..
`CALCUL..
`FUNWIND-PROTECT^CORE^FN^^
`FUNWIND-PROTECT^CORE^FN^^
`DO-TRACK-LLVM-TIME^COMPILER^FN^^
`JIT-ADD-MODULE-RETURN-FUNCTION^COMPILER^FN^^
`JIT-ADD-MODULE-RETURN-DISPATCH-FUNCTION^COMPILER^FN^^
`MULTIPLE-VALUE-PROG1-FUNCTION^CORE^FN^^
`MULTIPLE-VALUE-PROG1-FUNCTION^CORE^FN^^
`CODEGEN-DISPATCHER-FROM-DTREE^COMPILER^FN^^
`core::funcall_va_list_6(gctools::smart_ptr<core::Function_O>, gctools::smart_ptr
`APPLY^COMMON-LISP^FN^^
`CODEGEN-DISPATCHER^COMPILER^FN^^
`CALCULATE-FASTGF-DISPATCH-FUNCTION^CLOS^FN^^
`FORCE-DISPATCHER^CLOS^FN^^
`INVALIDATED-DISPATCH-FUNCTION^CLOS^FN^^
`core::FuncallableInstance_O::invalidated_entry_point
`core::funcall_va_list_6(gctools::smart_ptr<core::Function_O>, gctools::smart_ptr
`APPLY^COMMON-LISP^FN^^
`INITIALIZE-INSTANCE^COMMON-LISP^((T))^METHOD^^


`GC_mark_from
`G..
`GC_do_local..
`G..
`GC_mark_local
`GC_app..
`GC_do_paralle..
`GC_star..
`GC_mark_some
`GC_fini..
`GC_stopped_..
`GC_try_to_collect_inner
`GC_collect_or_expand
`GC_alloc_large
`GC_generic_malloc
`GC_malloc_kind_global
`tl..
`__cxxab..
`__dynamic_cast
`core::cl__streamp(g..
`co..
`_p..
`_pthread_mutex_un..
`__fread
`funlockfile
`_..
`fread
`__dyna..
`core::input_stream_read_byte8(gctools::sm..
`core::st..
`_..
`core::utf_8_decoder(gctools::smart_ptr
`_..
`_..
`core::Str8Ns_O::make(unsigned long, unsigned char, bool, gc..
`co..
`core::coerce::inputS..
`core::eformat_read_char(gctools::smart_ptr
`core::st..
`th..
`core::cl__read_line(gctools::smart_ptr<core::T_O>, gctools::smart_ptr<core::T_O>, gctools::smart_ptr<core::T_O>, gctools::smart_ptr
`co..
`READ-LINE^COMMON-LISP^FN^^
`PDB-SCANNER-READ-LINE^LEAP.PDB^FN^^
`MULTIPLE-VALUE-PROG1-FUNCTION^CORE^FN^^
`FUNWIND-PROTECT^CORE^FN^^
`SCAN-PDB^LEAP.PDB^FN^^
`MULTIPLE-VALUE-PROG1-FUNCTION^CORE^FN^^
`FUNWIND-PROTECT^CORE^FN^^
`LOAD-PDB^LEAP.PDB^FN^^
`TRIGGER-DTRACE-START^CORE^FN^^
`FUNWIND-PROTECT^CORE^FN^^
`APPLY^COMMON-LISP^FN^^
`CCLASP-EVAL-WITH-ENV^CLASP-CLEAVIR^((T T))^METHOD^^.4

CANDO

Computer Aided Nanostructure Design and Optimization

Jupyterlab

```
In [17]: (assign-atom-types *agg*)
         (energy:minimize *agg*
                          :restraints-on t
                          :max-sd-steps 1000
                          :sd-tolerance 5000.0
                          :max-cg-steps 1000)
```

```
======= Starting Steepest Descent Minimizer
---Stage-Seconds--Step-----Alpha---Dir------------Energy----------RMSforce
 minSDnP       0     1  0.000000   0.0              0.000          14309.953
 minSDnP       0     2  0.000241   0.0       15334412.708          10805.301
 minSDnP       0     3  0.000139   0.0       11855177.596           6309.419
 minSDnP       0     4  0.000258   0.0        9232272.319           5323.945
DONE absolute force test:
forceRmsMag(4852.389877).LT.forceTolerance(5000.000000)
======= Starting Conjugate Gradient Minimizer
 minCGnP       0     5  0.000000   0.0              0.000           4852.390
 minCGnP       0     6  0.000170  34.3        7778135.459           3304.155
 minCGnP       0     7  0.000168  53.2        7090916.013           4253.174
 minCGnP       0     8  0.000154  66.9        6267838.846           6140.031
 minCGnP       0     9  0.000038  43.4        6034328.183           3781.629
 minCGnP       0    10  0.000115  45.6        5585967.662           2999.755
---Stage-Seconds--Step-----Alpha---Dir------------Energy----------RMSforce
 minCGnP       0    11  0.000164  50.9        5145147.554           2937.863
 minCGnP       0    12  0.000112  45.5        4902050.564           2135.227
 minCGnP       0    13  0.000164  53.2        4712641.677           2373.039
 minCGnP       0    14  0.000143  44.4        4554811.644           1844.416
 minCGnP       0    15  0.000140  56.9        4418553.643           2014.536
```

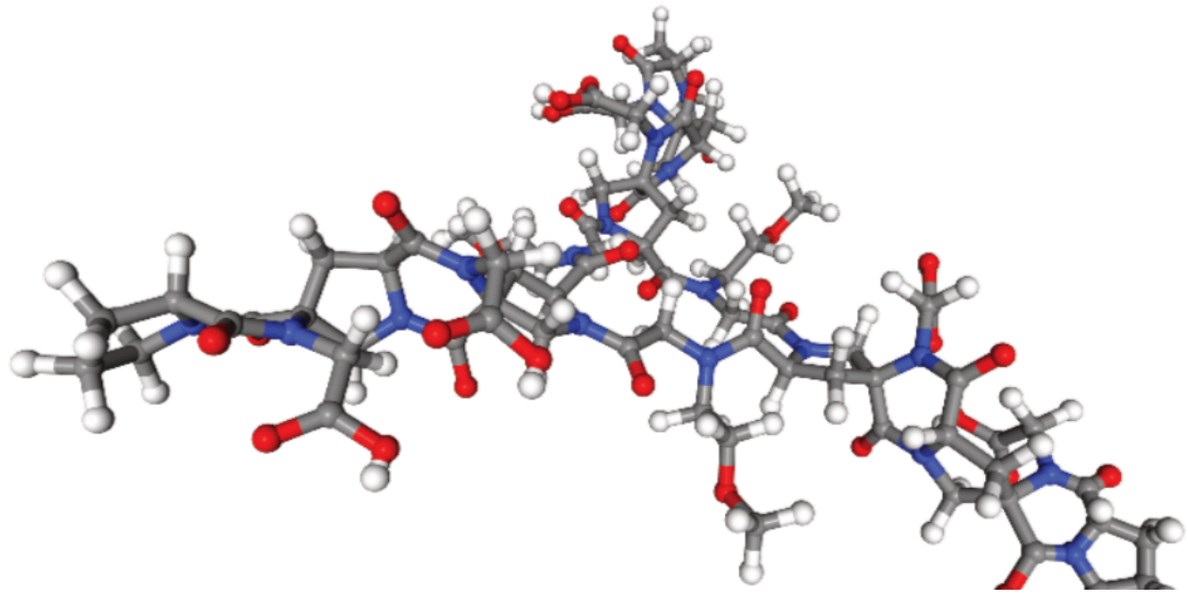
Check that the stereochemistry is maintained - they will all still be 'S' configuration.

```
In [18]: (calculate-all-stereochemistry *stereocenters*)
```

...

```
In [19]: (nglv:show-aggregate *agg*)
```

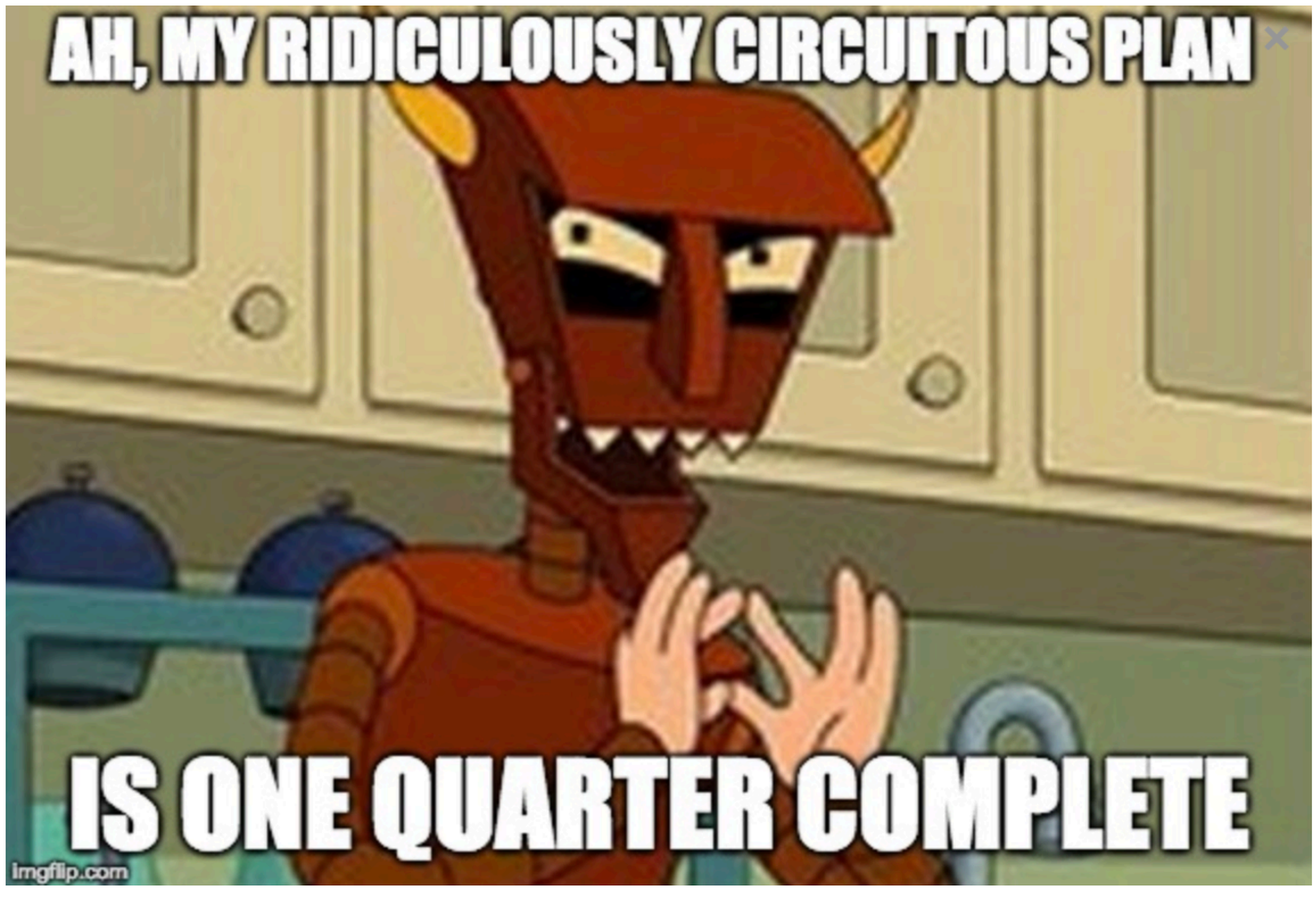
Out[19]:



LLVM community and #llvm members

Thank you!

<u>Cry for help!</u>
Calling conventions
DebugInfo access



Clasp & Cando: What, How, Where?

- macOS, Linux

github.com/clasp-developers/clasp

github.com/drmeister/cando