Towards OpenMP Support in LLVM

Alexey Bataev, Andrey Bokhanko, James Cownie
Intel

Software and Services Group

Agenda

- What is the OpenMP* language?
- Who Can Benefit from the OpenMP language?
- OpenMP Language Support
  - Early / Late Outlining
  - History
  - OpenMP Runtime
- OpenMP support in Clang*

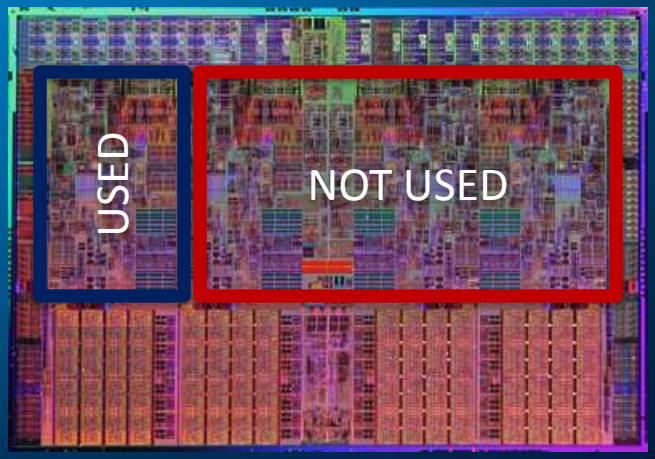
What is the OpenMP Language?

- **Industry-wide standard** for shared memory multiprocessing programming
- Vendor-neutral, platform-neutral, portable, managed by an independent consortium
- Supports C, C++ and Fortran
  - Implemented in GCC*, ICC, Open64*, Visual C++*, …
  - But not in Clang / LLVM*!
- Current version is 3.1
  - 4.0 under development
- **www.openmp.org**

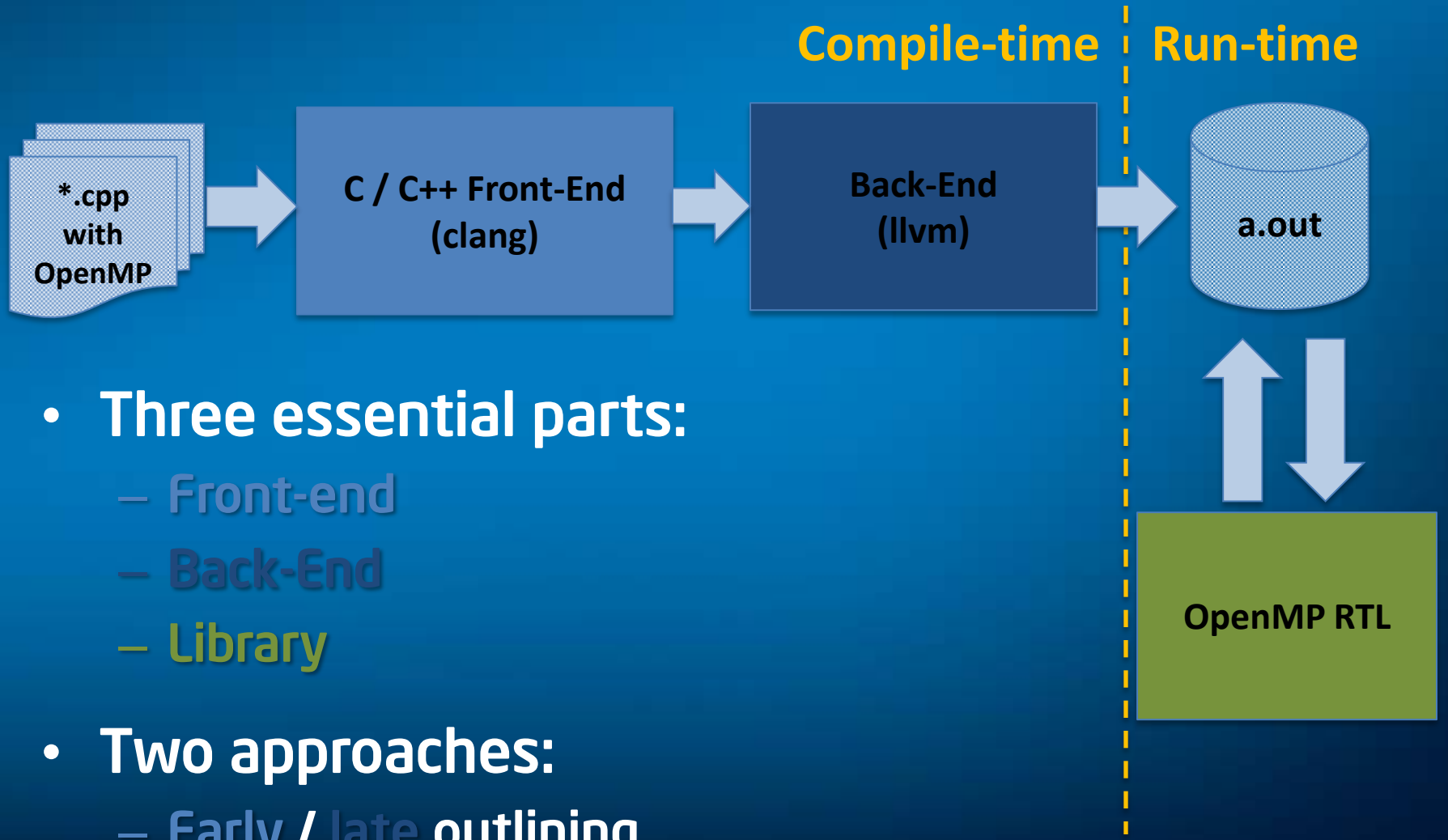
```
#pragma omp parallel for
for (i = 0; i < N; i++)
{
    ...
}
```

Who Can Benefit from the OpenMP Language?

- Anyone who uses a multi-core processor
  - Your phone almost certainly has more than 1 core!
- "Must have" for HPC
  - Without OpenMP support, LLVM is at a disadvantage in this area
- Becomes a "must have" for "power clients"
  - You can hardly find a desktop / notebook with a single core



OpenMP Support



- Three essential parts:
  - Front-end
  - Back-End
  - Library
- Two approaches:
  - Early / late outlining

Early / Late Outlining

- Parallel regions are put into separate routines
  - To be executed in separate threads
  - This can be done either in front-end or back-end

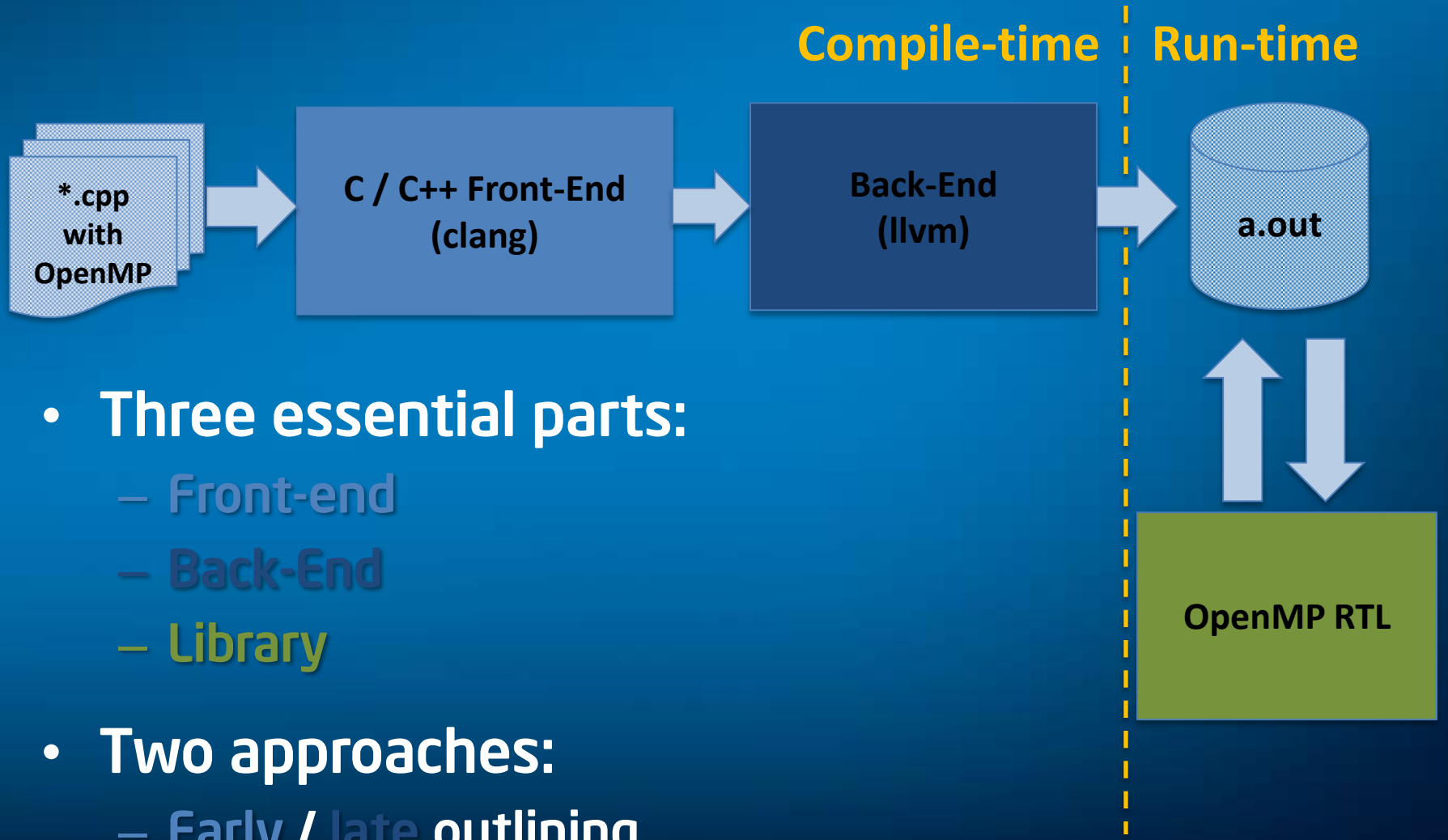
```
float a,x,y,z;
#pragma omp parallel for
for (i = 0; i < N; i++) {
  a[i] = x * y * z;
  ... // rest of loop
}
```

```
omp_parallel_for(0, N,
N/omp_get_num_threads(), forb)
...
void forb(int L, int U, R *r) {
  for (i = L; i < U; i++) {
    r->a[i] = r->x * r->y * r->z;
    ... // rest of loop
  }
}
```

OpenMP in LLVM: A Brief History

- 2H 2012: Several proposals with late outlining
  - From Intel, Hal Finkel, others
  - All of them involve changes to LLVM IR and thus, require modifications of LLVM phases
  - None of them got enough support in the community
- October 2012: OpenMP in Clang project
  - Started by AMD*, continued by Intel
  - Early outlining
  - OpenMP RTL calls generated in Clang
  - No changes to LLVM IR

OpenMP Support



- Three essential parts:
  - Front-end
  - Back-End
  - Library
- Two approaches:
  - Early / late outlining

OpenMP Runtime

- Fortunately, there is libgomp
  - Unfortunately, it is under GPLv3*
  - "Copyleft" license
- Clang / LLVM uses UoI / NCSA OSL*
  - Permissive (aka BSD-style) free software license
- Permissively licensed free runtime library is needed

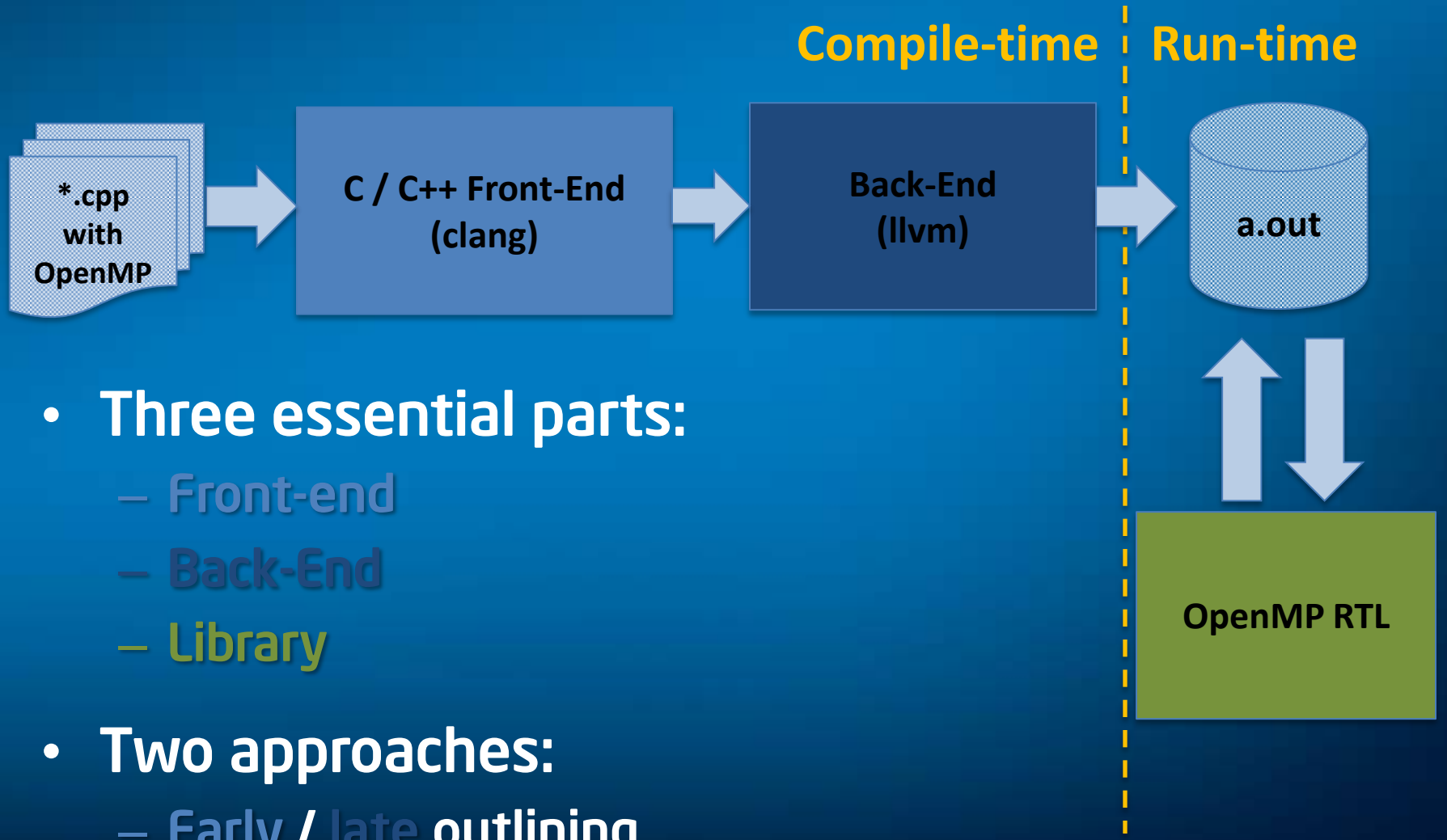
Intel® OpenMP Runtime

- Intel® OpenMP runtime was released in April with LLVM compatible 3-clause BSD license
- This is Intel's production runtime used by icc and ifort
- Continual development/tuning since before the OpenMP language existed (>15 years)
- Highly scalable (used on Intel® Xeon Phi™ coprocessor with 244 threads, large SGI* and Bull* ccNUMA SMP machines)

Intel® OpenMP Runtime

- Supports OpenMP 3.1 (and parts of OpenMP 4.0 [work in progress])
- ABI compatible with
  - Intel Compilers (icc, icpc, ifort)
  - GCC
    - so gcc compiled code can be linked in without libgomp to avoid issues if there are multiple OpenMP runtimes in the same process
- Doxygen* documentation in the source
- Available from www.openmprtl.org

OpenMP Support

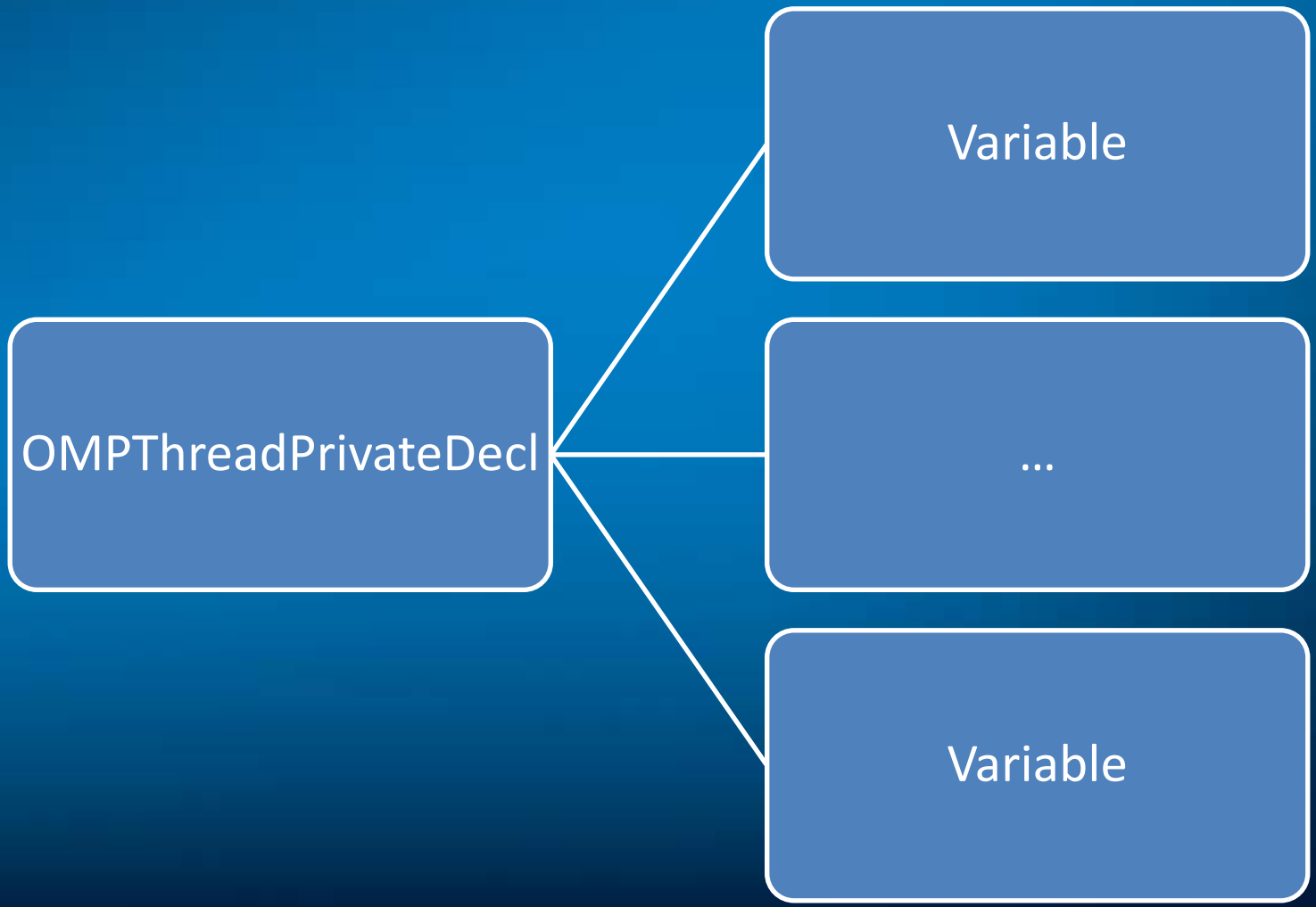


- Three essential parts:
  - Front-end
  - Back-End
  - Library
- Two approaches:
  - Early / late outlining

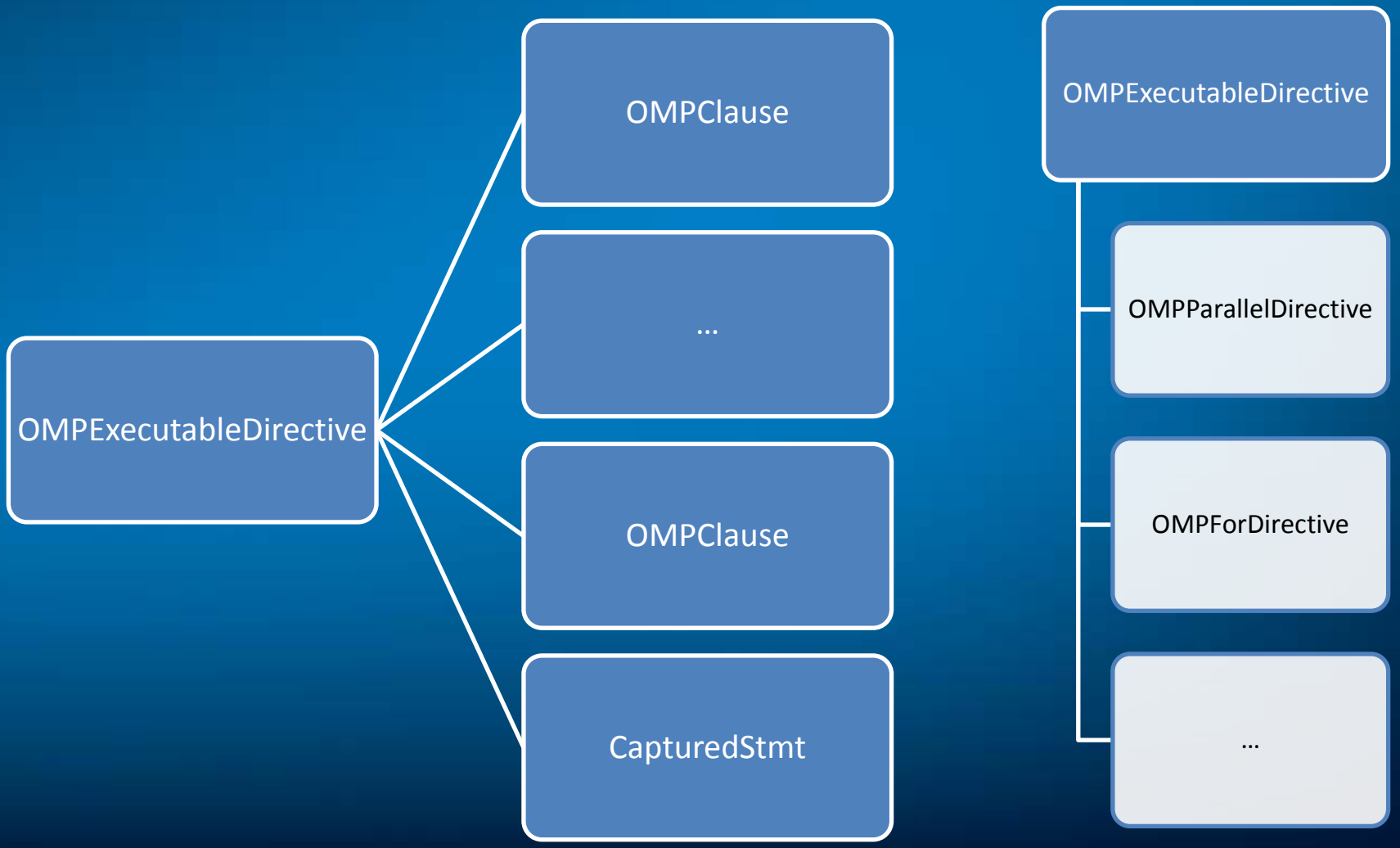
OpenMP Support in Clang

- First approach is to represent OpenMP directives as C++11 attributes (Olaf Krzikalla, Nov. 2012)
  - Currently may require two parsing passes
  - May need to change code generation for standard statements
- Second approach is to use standard pragma parsing harness
  - Declarative directive is represented as a special kind of declaration
  - Executable directives and clauses are represented as a special kind of statements

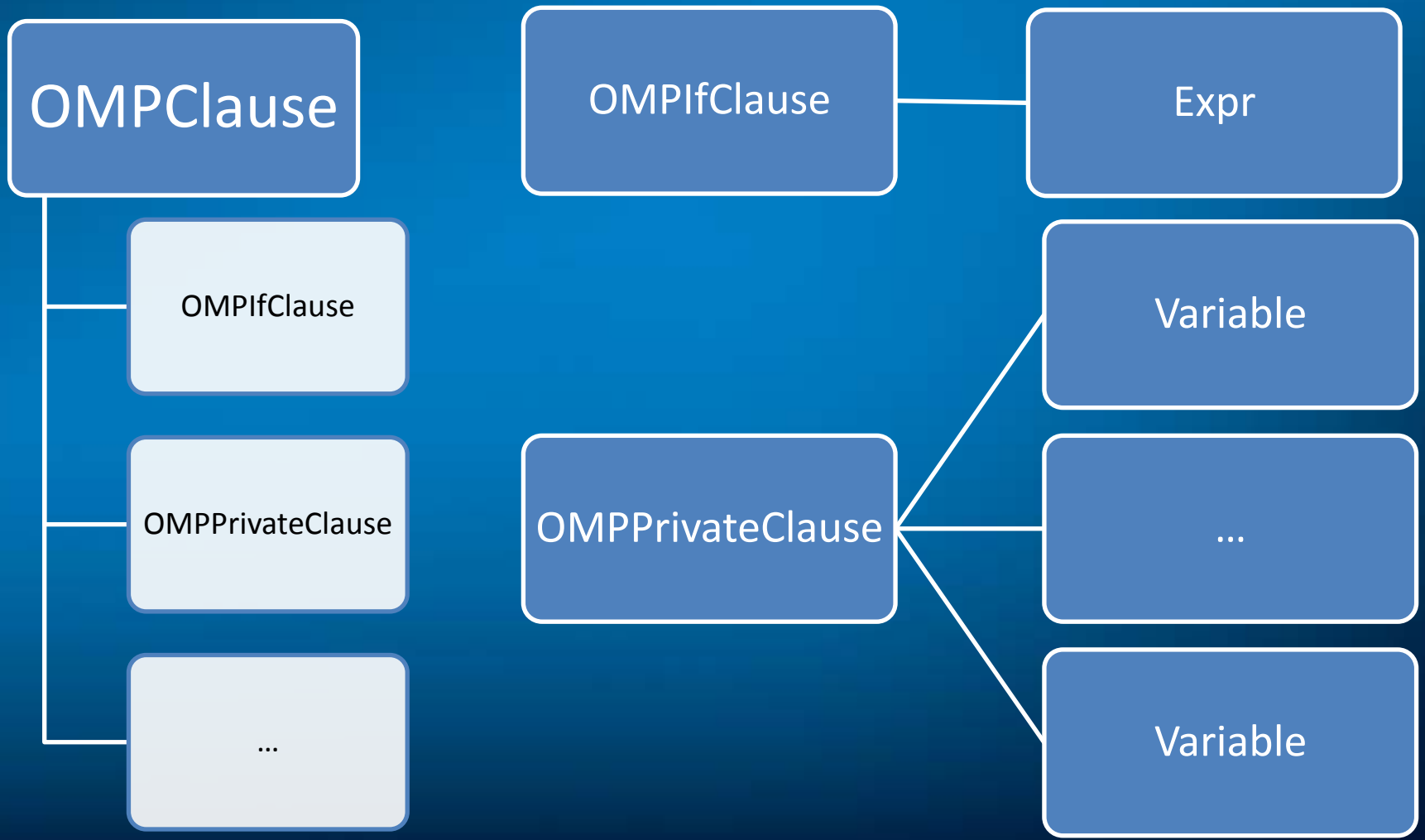
Representation in AST Declarative Directives



Representation in AST
Executable Directives



Representation in AST Clauses



Representation in AST Statements And Variables

- Statements are Structured Statements with protected regions
  - A single statement for #pragma omp parallel
  - One or more for-loops for #pragma omp for
- Statements are represented as CapturedStmt to capture local variables
  - Special processing for threadprivate variables
  - Private variables are constructed by default
  - Shared variables are captured by reference
  - Special processing for firstprivate, lastprivate, reduction variables

An Example

```
#pragma omp parallel if(a) private(argc,b)
foo();
```

```
-OMPParallelDirective <line:9:2, col:43>
  | |-OMPIfClause <col:22, col:27>
  | | `-ImplicitCastExpr <col:25> '_Bool' <IntegralToBoolean>
  | |   `-ImplicitCastExpr <col:25> 'int' <LValueToRValue>
  | |     `-DeclRefExpr <col:25> 'int' lvalue Var 'a' 'int'
  | |-OMPPrivateClause <col:28, col:43>
  | | |-DeclRefExpr <col:36> 'int' lvalue ParmVar 'argc' 'int'
  | | `-DeclRefExpr <col:41> 'int' lvalue Var 'b' 'int'
  | `-CapturedStmt <line:10:2, col:7>
  |   `-CallExpr <col:2, col:7> 'void'
  |     `-ImplicitCastExpr <col:2> 'void (*)(void)' <FunctionToPointerDecay>
  |       `-DeclRefExpr <col:2> 'void (void)' lvalue Function 'foo' 'void (void)'
```

Code Generation

- All variables are combined into an auto-generated record according to their data-sharing attributes (predetermined, explicit or implicit)
- OpenMP regions are outlined as functions with a single argument – pointer to the record
- LLVM IR code is generated to use captured variables instead of original ones

Current Status and Plans

- Implemented and committed:
  - -fopenmp option
  - #pragma omp threadprivate
  - Parsing and semantic analysis , AST representation
- Implemented, under code review:
  - All pragmas (parallel, for, sections, task etc.)
  - Parsing and semantic analysis, data-sharing attributes analysis, AST representation
- Under development
  - CodeGen for all OpenMP constructs

Acknowledgements

- Thank you to all code reviewers!
  - Especially to Dmitri Gribenko, Hal Finkel and Doug Gregor

- Your contribution is welcomed!

Legal Disclaimer & Optimization Notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804


intel
Software