Leibniz Supercomputing Centre

of the Bavarian Academy of Sciences and Humanities

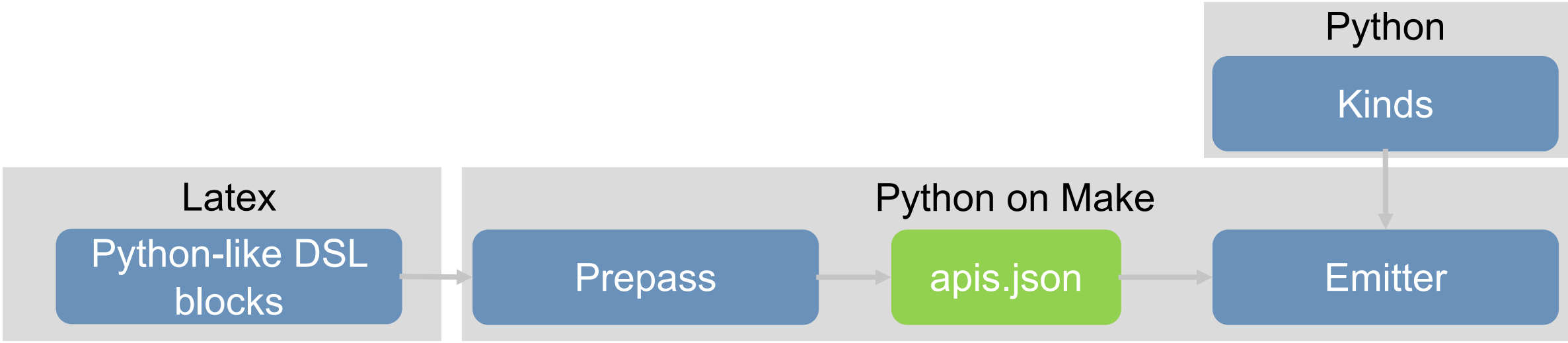
Pythonization and It’s Future

Leibniz Supercomputing Centre | Martin Ruefenacht

Review

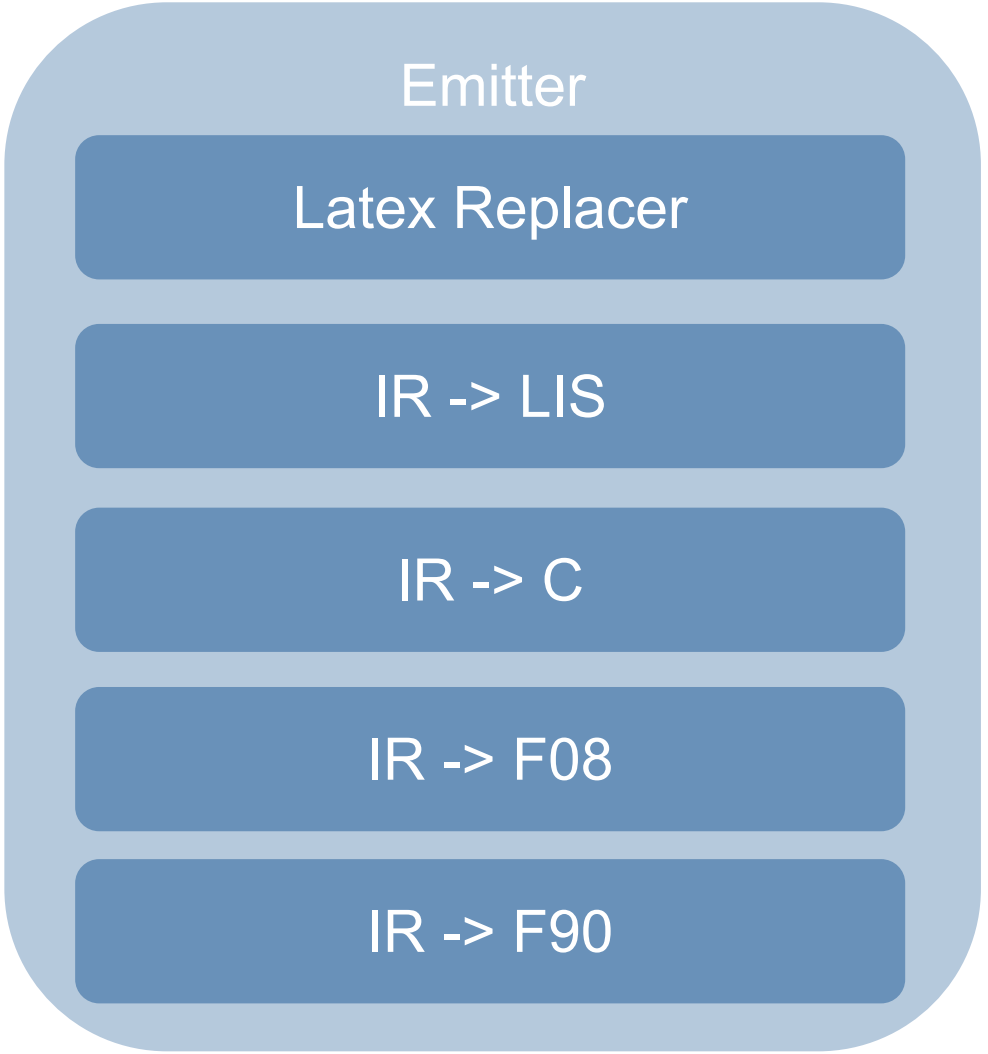
- Only ever intended for the Embiggenment
- DSL is limited to minimal information needed
- Somewhat convenient expression of bindings
- Intermediary data available instead of having to parse latex or PDF
- Unintended inclusion of other information

Present


Python
Kinds
Latex
Python-like DSL blocks
Python on Make
Prepass
apis.json
Emitter

Emitter

- Complex
  - Beyond just code synthesis
  - MPI style has many exceptions
- Fulfils what is currently required
- Doesn’t capture a lot of MPI information present in the Standard



KINDs

- Mix of syntactic and behavioural types
- Value-space is not included at all
- We do not know how many kinds we have in MPI (188+)
- SoA vs AoS
  - Not particularly human friendly

```
# Various types of integers
# NNI is non-negative integer
'ARRAY_LENGTH'        : 'integer',
'ARRAY_LENGTH_NNI'    : 'non-negative integer',
'ARRAY_LENGTH_PI'     : 'positive integer',
'ATTRIBUTE_VAL_10'    : None, # From MPI-1.0
'ATTRIBUTE_VAL'       : None, # Current version of MPI
'BLOCKLENGTH'         : 'non-negative integer',
'COLOR'               : 'integer',
'COORDINATE'          : 'integer',
'COORDINATE_NNI'      : 'non-negative integer',
'DEGREE'              : 'non-negative integer',
'DIMENSION'           : 'integer',
'ENUM'                : 'integer',
'FILE_DESCRIPTOR'     : 'integer',
'KEY'                 : 'integer',
'KEYVAL'              : 'integer',
'INDEX'               : 'integer',
'LOGICAL'             : 'logical',
'LOGICAL_OPTIONAL'    : 'integer',
'LOGICAL_BOOLEAN'     : 'boolean',
'MATH'                : 'integer',
'NUM_DIMS'            : 'integer',
'RANK'                : 'integer',
'RANK_NNI'            : 'non-negative integer',
'COMM_SIZE'           : 'integer',
'COMM_SIZE_PI'        : 'positive integer',
'STRING_LENGTH'       : 'integer',
'STRIDE_BYTES'        : 'integer',
'STRIDE_ELEM'         : 'integer',
'TAG'                 : 'integer',
'VERSION'             : 'integer',
'WEIGHT'              : 'non-negative integer',
'OFFSET'              : 'integer',
'PROFILE_LEVEL'       : 'integer',
'ASSERT'              : 'integer',
'WINDOW_SIZE'         : 'non-negative integer',
'INFO_VALUE_LENGTH'   : 'integer',
'ACCESS_MODE'         : 'integer',
'KEY_INDEX'           : 'integer',
'TOOLENUM_INDEX'      : 'integer',
'TOOLENUM_SIZE'       : 'integer',
'TOOL_VAR_VERBOSITY'  : 'integer',
'TOOL_VAR_VALUE'      : 'integer',
'CVAR_INDEX'          : 'integer',
'CVAR_INDEX_SPECIAL'  : 'index',
'PVAR_INDEX'          : 'integer',
'PVAR_CLASS'          : 'integer',
'SOURCE_INDEX'        : 'integer',
'EVENT_INDEX'         : 'integer',
'CAT_INDEX'           : 'integer',
'UPDATE_NUMBER'       : 'integer',
'DROPPED_COUNT'       : 'positive integer',
'TYPECLASS_SIZE'      : 'integer',
'GENERIC_DTYPE_INT'   : 'integer',
'GENERIC_DTYPE_COUNT' : 'integer',
'PROCESS_GRID_SIZE'   : 'positive integer',
'DTYPE_DISTRIBUTION'  : 'positive integer',
```

```
# Various types of integers
'ARRAY_LENGTH'        : 'int',
'ARRAY_LENGTH_NNI'    : 'int',
'ARRAY_LENGTH_PI'     : 'int',
'ATTRIBUTE_VAL_10'    : 'void', # From MPI-1.0
'ATTRIBUTE_VAL'       : 'void', # Current version of MPI
'BLOCKLENGTH'         : 'int',
'COLOR'               : 'int',
'COORDINATE'          : 'int',
'COORDINATE_NNI'      : 'int',
'DEGREE'              : 'int',
'DIMENSION'           : 'int',
'ENUM'                : 'int',
'FILE_DESCRIPTOR'     : 'int',
'KEY'                 : 'int',
'KEYVAL'              : 'int',
'INDEX'               : 'int',
'LOGICAL'             : 'int',
'LOGICAL_OPTIONAL'    : 'int',
'LOGICAL_BOOLEAN'     : 'int',
'MATH'                : 'int',
'NUM_DIMS'            : 'int',
'RANK'                : 'int',
'RANK_NNI'            : 'int',
'COMM_SIZE'           : 'int',
'COMM_SIZE_PI'        : 'int',
'STRING_LENGTH'       : 'int',
'STRIDE_BYTES'        : 'MPI_Aint',
'STRIDE_ELEM'         : 'int',
'TAG'                 : 'int',
'VERSION'             : 'int',
'WEIGHT'              : 'int',
'OFFSET'              : 'MPI_Offset',
'PROFILE_LEVEL'       : 'int',
'WINDOW_SIZE'         : 'MPI_Aint',
'INFO_VALUE_LENGTH'   : 'int',
'ACCESS_MODE'         : 'int',
'UPDATE_MODE'         : 'int',
'KEY_INDEX'           : 'int',
'TOOLENUM_INDEX'      : 'int',
'TOOLENUM_SIZE'       : 'int',
'TOOL_VAR_VERBOSITY'  : 'int',
'TOOL_VAR_VALUE'      : 'int',
'CVAR_INDEX'          : 'int',
'CVAR_INDEX_SPECIAL'  : 'int',
'PVAR_INDEX'          : 'int',
'PVAR_CLASS'          : 'int',
'SOURCE_INDEX'        : 'int',
'TOOLS_TICK_COUNT'    : 'MPI_Count',
'EVENT_INDEX'         : 'int',
'CAT_INDEX'           : 'int',
'UPDATE_NUMBER'       : 'int',
'DROPPED_COUNT'       : 'MPI_Count',
'TYPECLASS_SIZE'      : 'int',
'GENERIC_DTYPE_INT'   : 'int',
'GENERIC_DTYPE_COUNT' : 'MPI_Count',
'PROCESS_GRID_SIZE'   : 'int',
'DTYPE_DISTRIBUTION'  : 'int',
```

Revamp

- Make MPI Standard machine readable
  - Expose all information present
- Don’t expose apis.json as API
  - It’s ugly!
- Do the heavy lifting for everyone
- Expose much more information
  - Stages
  - Kind Value-space
  - Constants
  - Contracts
  - …

```python
@hook(priority=0)
def binding_iso_c_procedure(procedure: Union[ISOCProcedure, EmbiggenedISOCProcedure]) -> str:
    """Generate all ISO C latex bindings from the given Procedure."""

    assert isinstance(procedure, (ISOCProcedure, EmbiggenedISOCProcedure)), type(procedure)

    binding = []

    if isinstance(procedure, EmbiggenedISOCProcedure):
        binding.append(f"\\MPIbindindex{{{procedure.name}}}\n")

    if procedure.has_uppercase_index():
        binding.append(r"\mpiemptybindidx{")

    elif procedure.return_kind == KINDS.ERROR_CODE:
        binding.append(r"\mpibind{")

    else:
        binding.append(f"\\mpibindnotint{{{procedure.return_type} ")

    binding.append(f"{procedure.name}(")

    if procedure.parameters:
        binding.append(r"\gb{}")

    else:
        binding.append("void")

    parameters = map(lambda parameter: str(parameter).replace(" ", "~"), procedure.parameters)
    binding.append(f"{', '.join(parameters)}}}")

    if procedure.has_uppercase_index():
        binding.append(f"{{{procedure.return_type}}}")
        binding.append(f"{{{procedure.name.upper()}}}")

    return "".join(binding)
```

Proposed path

- Form a WG for editing/information/artefacts?
  - Central point of contact
  - Essentially office hours

- Suggestions for what is wanted

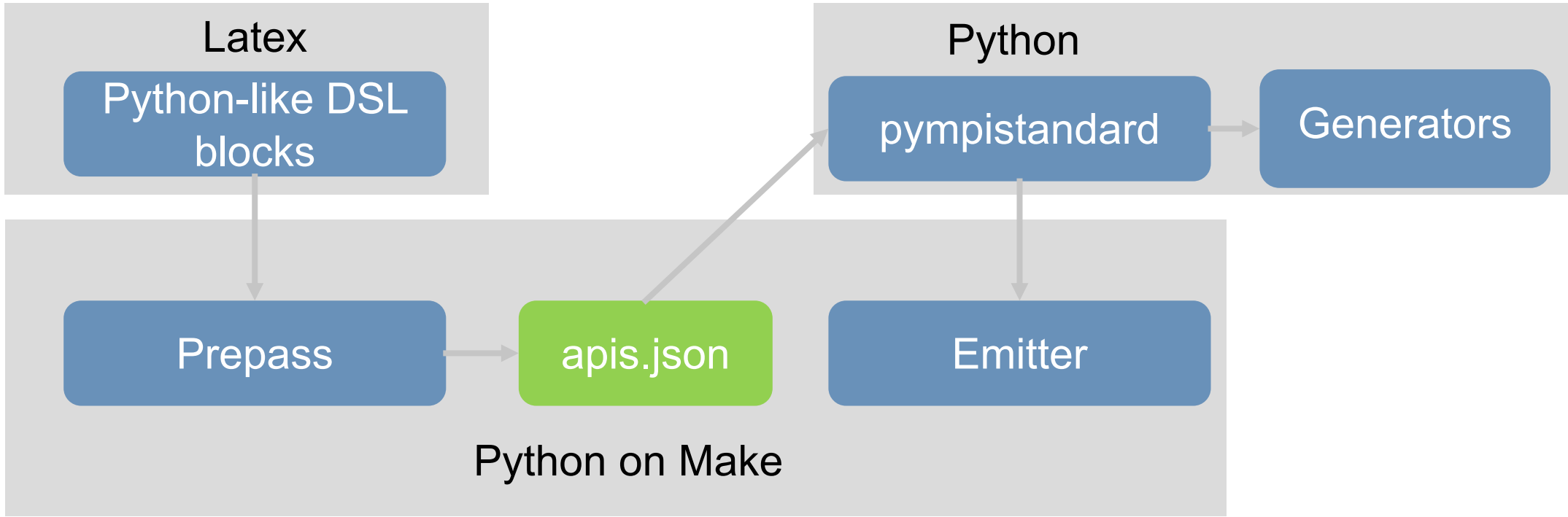
- Merging revamped API
  - pympistandard on pypi
  - binding_tool in tools/
    - plugins
  - Potentially other tools in MPI standard repo

```python
import pympistandard as std
std.use_api_version(1)


for name, procedure in std.PROCEDURES.items():
    if procedure.express.lis is None:
        continue

    for parameter in procedure.express.lis.parameters:
        if parameter.kind is std.KINDS.STATUS:
            print(name)
```

Future


Latex
Python-like DSL blocks
Python
pympistandard
Generators
Prepass
apis.json
Emitter
Python on Make