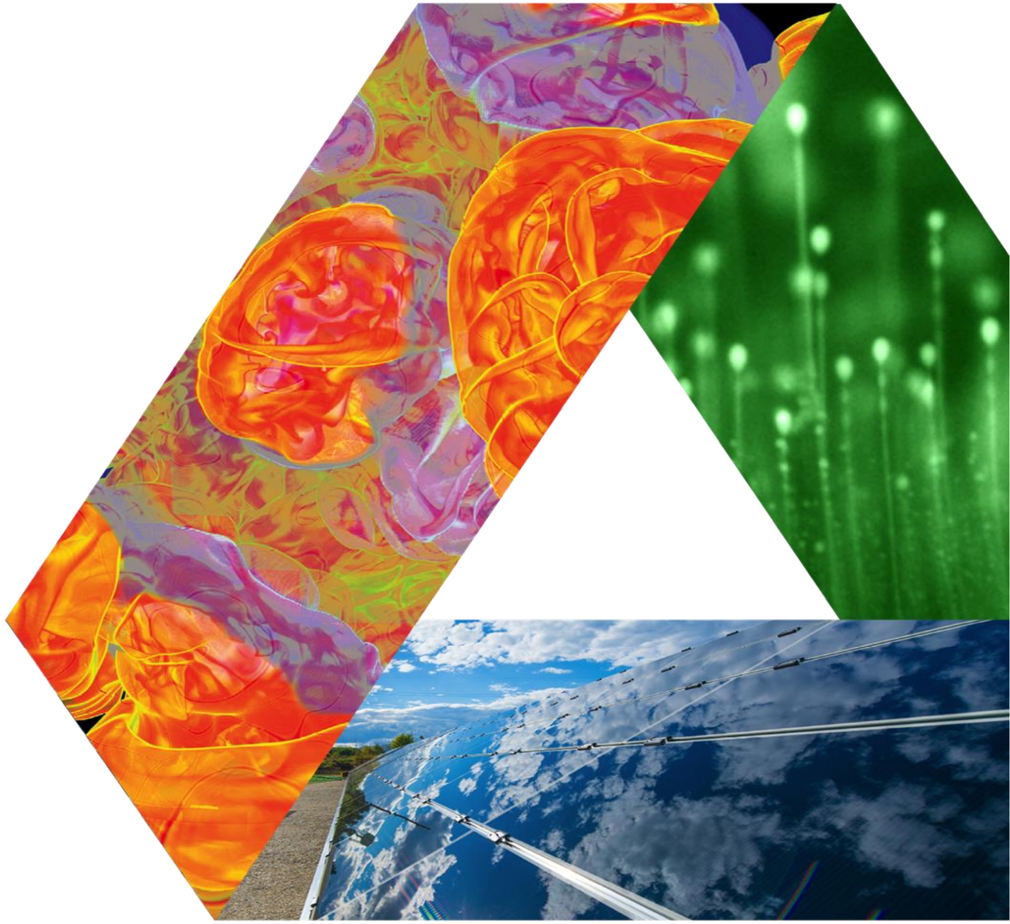
IDEAS productivity

Performance Portability Without Relying on C++ Based Abstractions

PERMAVOST Workshop Orlando 2023

Anshu Dubey

Contributors: Tom Klosterman, Jared O’Neal, Johann Rudi
Mohamed Wahib, , Klaus Weide



Argonne National Laboratory

EXASCALE COMPUTING PROJECT

U.S. DEPARTMENT OF ENERGY | Office of Science

acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357

Starting Point in Extensible Software Architecture

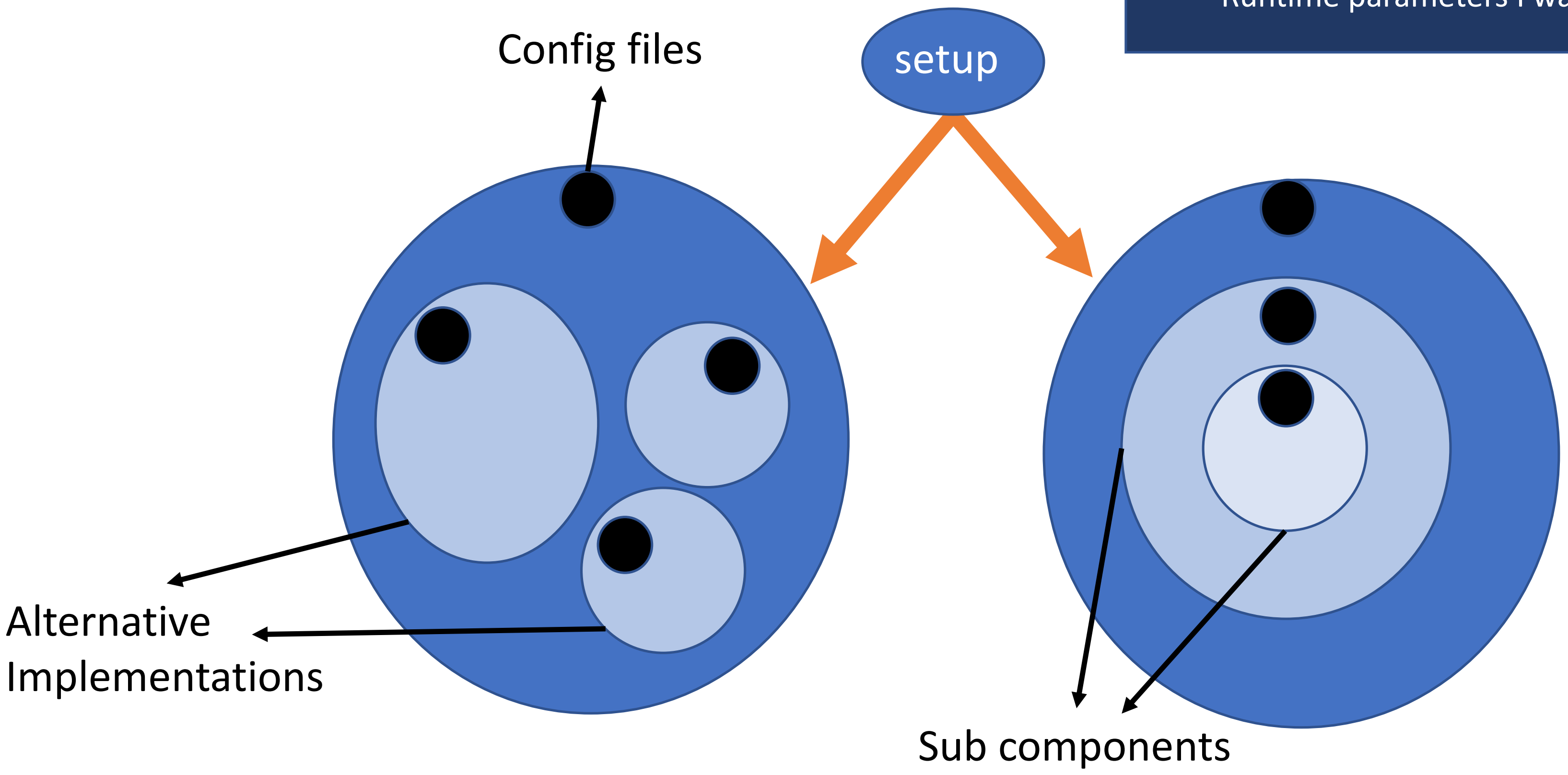
- Building blocks of code
  - Hierarchy of granularity
  - Units, subunits, components
- Multiple alternative implementations
  - Null implementations of API
  - High degree of composability
  - High degree of customizability
- A tool that can arbitrate on what to include when
  - Self describing code components

```
#        Config file for the gravity module.  Available sub-modules:

#          Constant     Spatially/temporally constant gravitational field
#          PlanePar     1/r^2 field for a distant point source
#          PointMass  1/r^2 field for an arbitrarily placed point source
#          Poisson       Field for a self-gravitating matter distribution
#   UserDefined A user-defined field

REQUIRES Driver
DEFAULT Constant
PPDEFINE GRAVITY
EXCLUSIVE Constant PlanePar PointMass Poisson UserDefined
PARAMETER useGravity BOOLEAN TRUE
```

Argonne NATIONAL LABORATORY

The Key to Composability

Encoded metadata
Units I need
Units I don't work with
State variables I need
AMR specific needs
Runtime parameters I want …



Platform Heterogeneity

Computation
- CPU
- GPU
- Other acceler-ators
- Other devices

Memory
- Cache hierar-chy
- Device memory
- NVram
- Other types

Network
- Between nodes
- Within node
- With I/O
- Other types

Mechanisms Needed by the Code

Mechanisms to unify expression of computation

- Minimize maintained variants of source suitable for all computational devices
- Reconcile differences in data structures

Mechanisms to move work and data to computational targets

- Moving between devices
  - Launching work at the destination
  - Hiding latency of movement
- Moving data offnode

Mechanisms to map work to computational targets

- Figuring out the map
  - Expression of dependencies
  - Cost models
- Expressing the map

So what do we need?

- Abstractions layers
- Code transformation tools
- Data movement orchestrators

Philosophy of Design

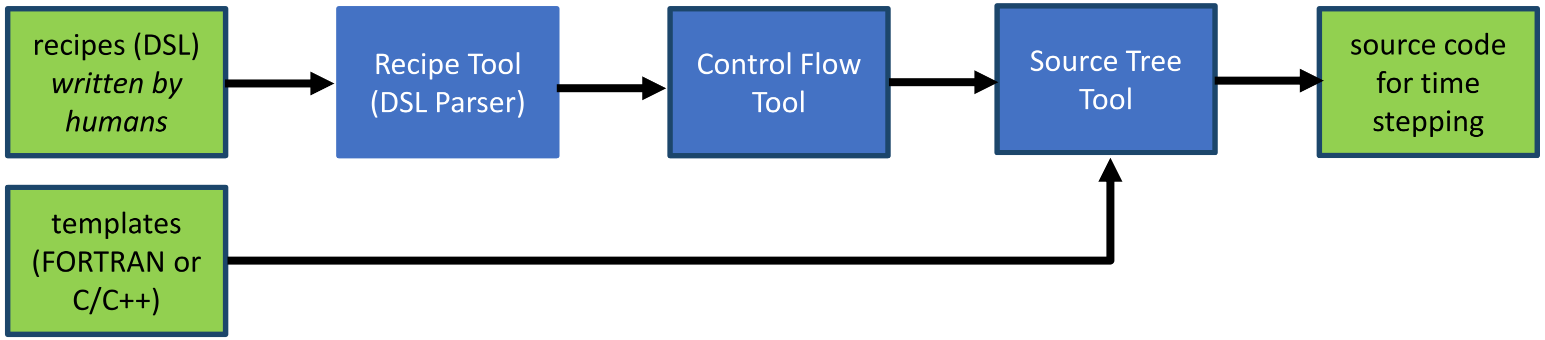
- Let the code developer decide what should be done for optimization on a platform
  - Make it easy to have that happen without coding to metal

- Have a set of tools, each with limited functionality
  - Tools remain simple and easy to maintain by non-experts
  - Combination of tools provides a powerful solution

- Tools can permute and combine building blocks, do some code translation and compose a full application
- As far as possible tools also have building blocks

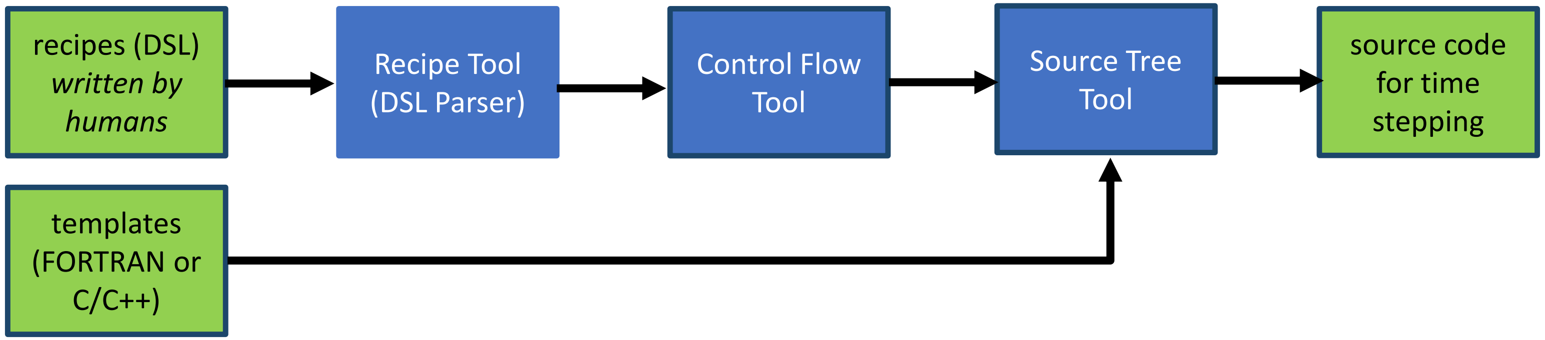
CGkit

- Generating Code from Recipes and code Templates



CGkit

- Generating Code from Recipes and code Templates



Orthogonal separation of concerns

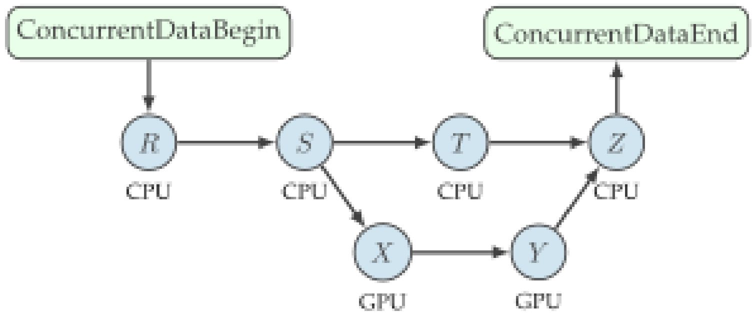
Example recipe

express dependencies

express hardware mapping

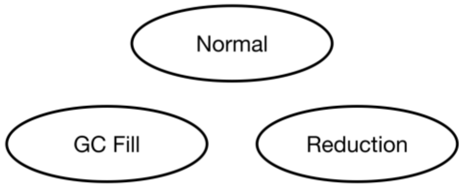
```
dIn   = ConcurrentDataBegin()()

aR    = Action(routine='function_R')(dIn)
aS    = Action(routine='function_S')(aR)
aT    = Action(routine='function_T')(aS)

aX    = Action(routine='function_X')(aS)
aY    = Action(routine='function_Y')(aX)
aZ    = Action(routine='function_Z')([aT,aY])

dOut  = ConcurrentDataEnd()(aZ)

ConcurrentHardware(CPU={'actions': [aR,aS,aT,aZ]},
                   GPU={'actions': [aX,aY]})
```

Resulting control flow graph



Milhoja – domain specific runtime

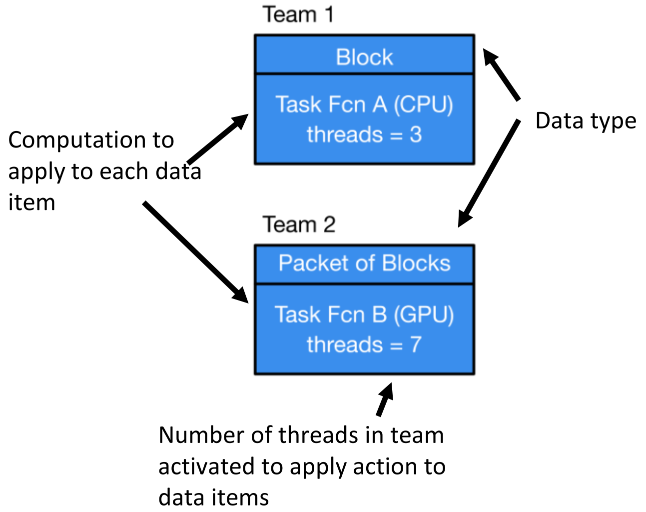
❑ A Toolkit for Building Pipelines



Distributors

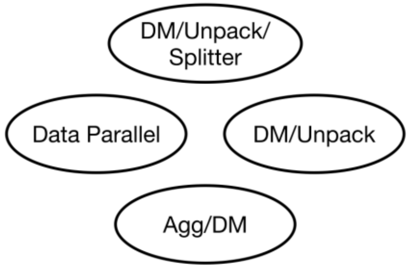
- Use block iterator
- Aggregate blocks if necessary
- Initiate asynchronous transfers if necessary
- Push blocks to other elements

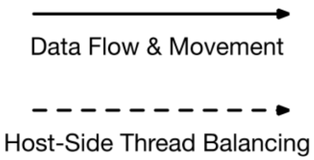
Thread Teams



Helpers

- Initiate asynchronous transfers if necessary
- Translate data types



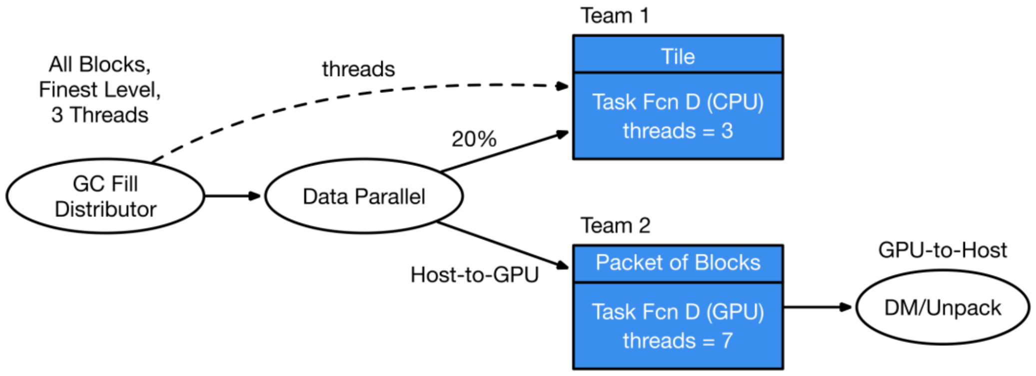


Argonne NATIONAL LABORATORY

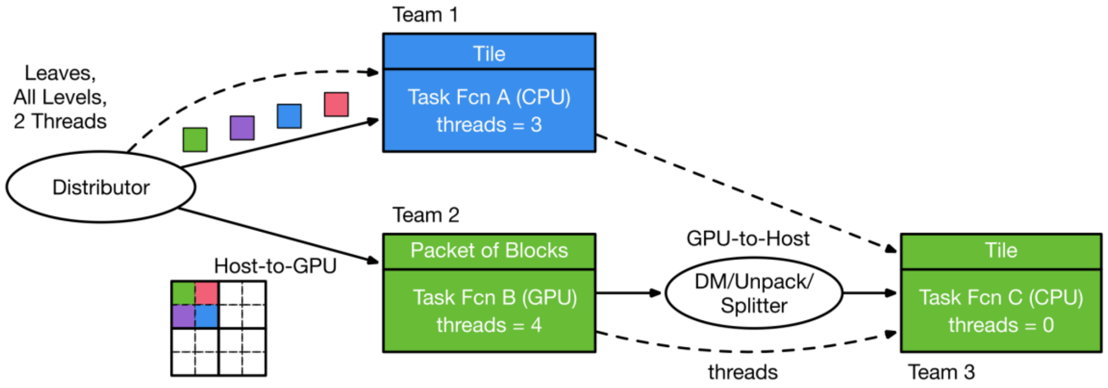
Thread Team Configurations

- ❑ Expose Hierarchy of Parallelism

Single data-parallel pipeline configuration
CPU/GPU Data Parallel



Two-pipeline configuration



Task functions applied to each block

- in order within a pipeline and
- concurrently across pipelines.

Allow for running concurrently independent actions from different physics units.

Macroprocessor – unify static code

- Mimic the functionality of template meta-programming
  - Single source code with specializations for variants
- Code in building blocks
  - That can be permuted and combined
  - Smaller building blocks can be fused into bigger ones for performance if needed

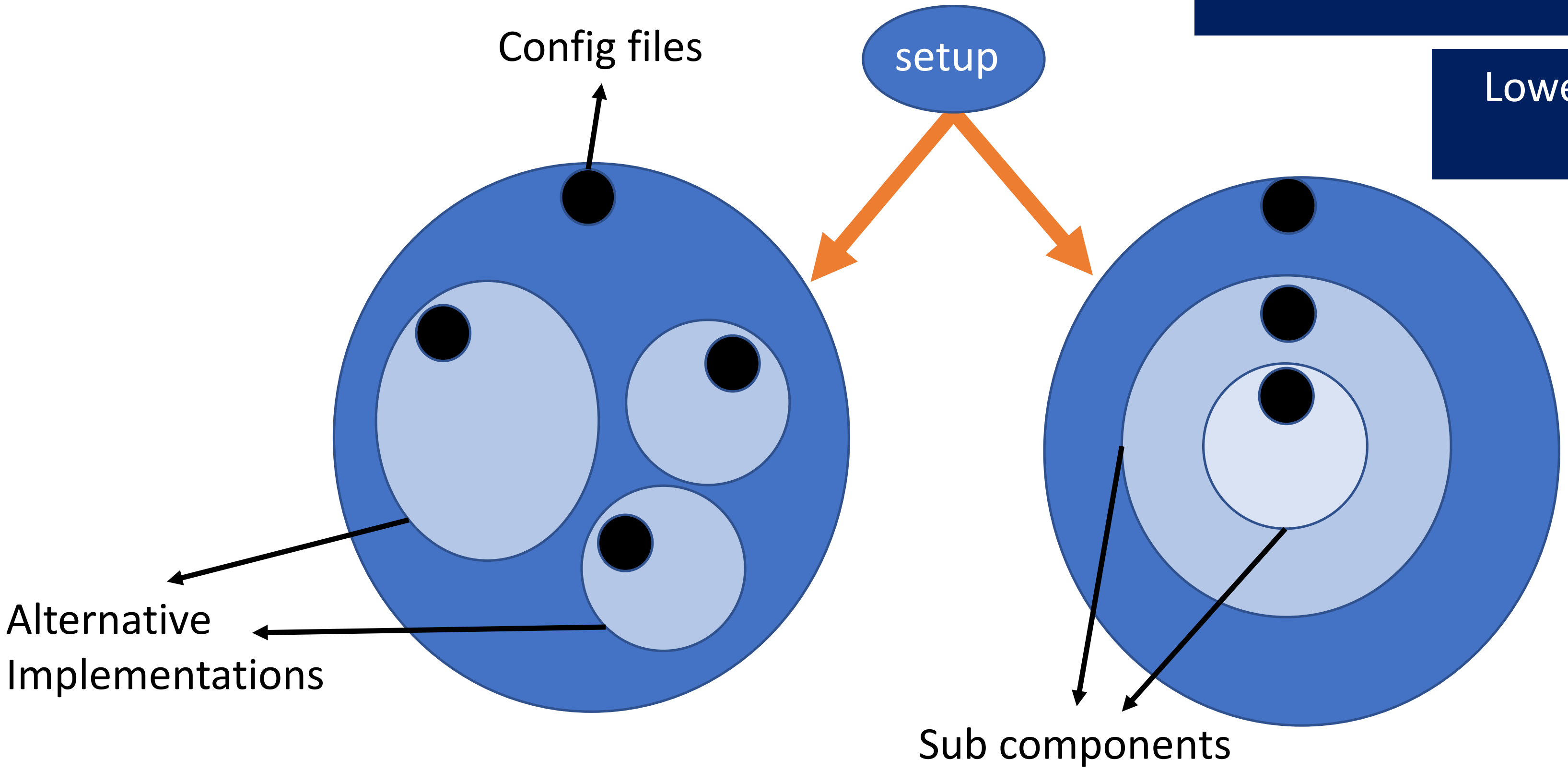
Modification in Configuration

Encoded metadata
Other compoments I need
Components I don't work with
State variables I need
Runtime parameters I want …

Lowest granularity -- subroutine

Express code with embedded macros

- Let macros have multiple alternative definitions
- Implement mechanism to select specific macro definition
- Implement mechanism to safely include more than one definition
- Allow inline, recursion and arguments in macros



Code Expressed with Keys

```
@M declare
@M directive1
@M loop_2d
      .... computation 1
      .... computation 2
@M endloop_2d_spl
@M directive2
@M loop_2d_spl
      .... computation 3
@M endloop_2d
```

Definitions for CPU

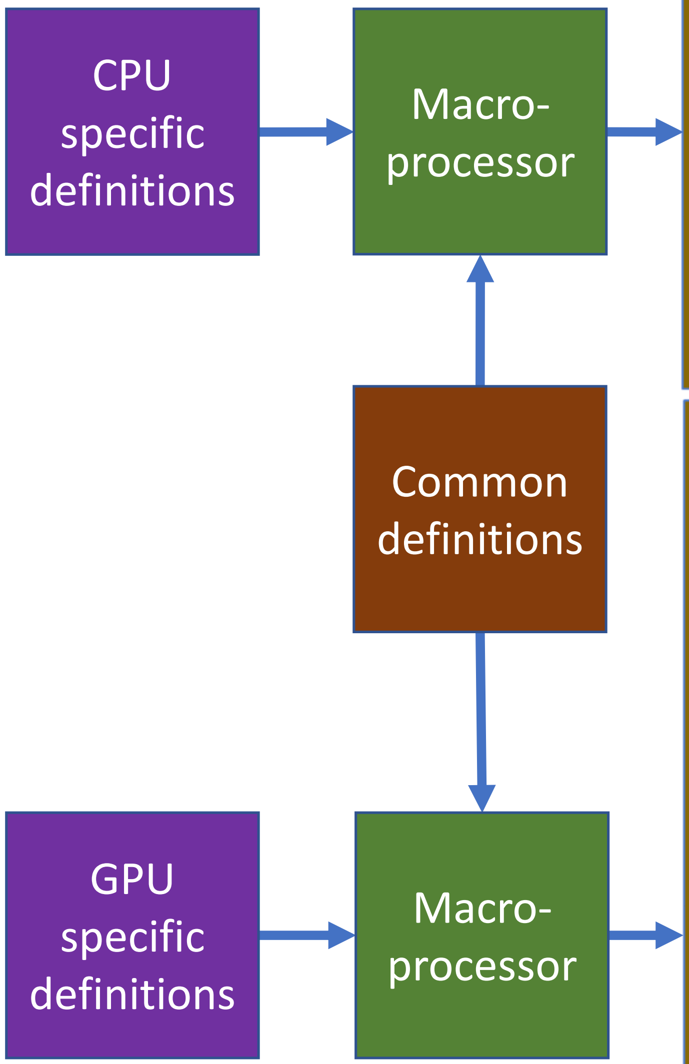
```
[declare]
   definition=
  cpu-specific declarations
[directive1]
  definition=
  !!omp directive for cpu
[endloop_2d_spl]
 definition=
[directive2]
 definition=
[loop_2d_spl]
 definition=
```

Common definitions

```
[loop_2d]                [endloop_2d]
  definition=              definition=
  do j=1,n                    end do
   do i=1,m                 end do
```

Definitions for GPU

```
[declare]
 definition=
  gpu-specific declarations
[directive1]
 definition=
  !!omp directive1 for gpu
[endloop_2d_spl]
 definition=
   @M endloop_2d
[directive2]
 definition=
!!omp directive2 for gpu
[loop_2d_spl]
 definition=
   @M loop_2d
```




```
cpu-specific declaration
!!omp directive for CPU
do j=1,n
   do i=1,m
       .... computation 1
       .... computation 2
       .... computation 3
   end do
 end do
```

```
gpu-specific declarations
!!omp directive1 for GPU
do j=1,n
   do i=1,m
       .... computation 1
       .... computation 2
   end do
end do
!!omp directive2 for GPU
do j=1,n
   do i=1,m
       .... computation 3
   end do
 end do
```

NATIONAL LABORATORY

```
If(telescoping) then
  call gcfill
  @M iter_begin
       @M hy_save_state_1blk
       @M hy_prepare_stages
      do stage = 1,last_stage
         @M hy_set_limits
         @M hy_do_one_stage
         if(stage==last_stage)
            @M hy_update_state_1blk
         endif
      end do
   @M iter_end
else
   @M hy_save_global_state
   @M hy_prepare_stages
  do stage = 1,last_stage
     Call Gcfill
     @M iter_begin
        @M hy_do_one_stage
        @M hy_update_global_state
     @M iter_end
   end do
end if
```

```
[hy_do_one_stage]
definition =
  call hy_grav (@M hy_grav_args)
  call hy_getFaceFlux (@M hy_ff_args)
  call hy_addFluxes(@M hy_af_args)
  call hy_updateSolution(@M hy_us_args)
  call Eos
```

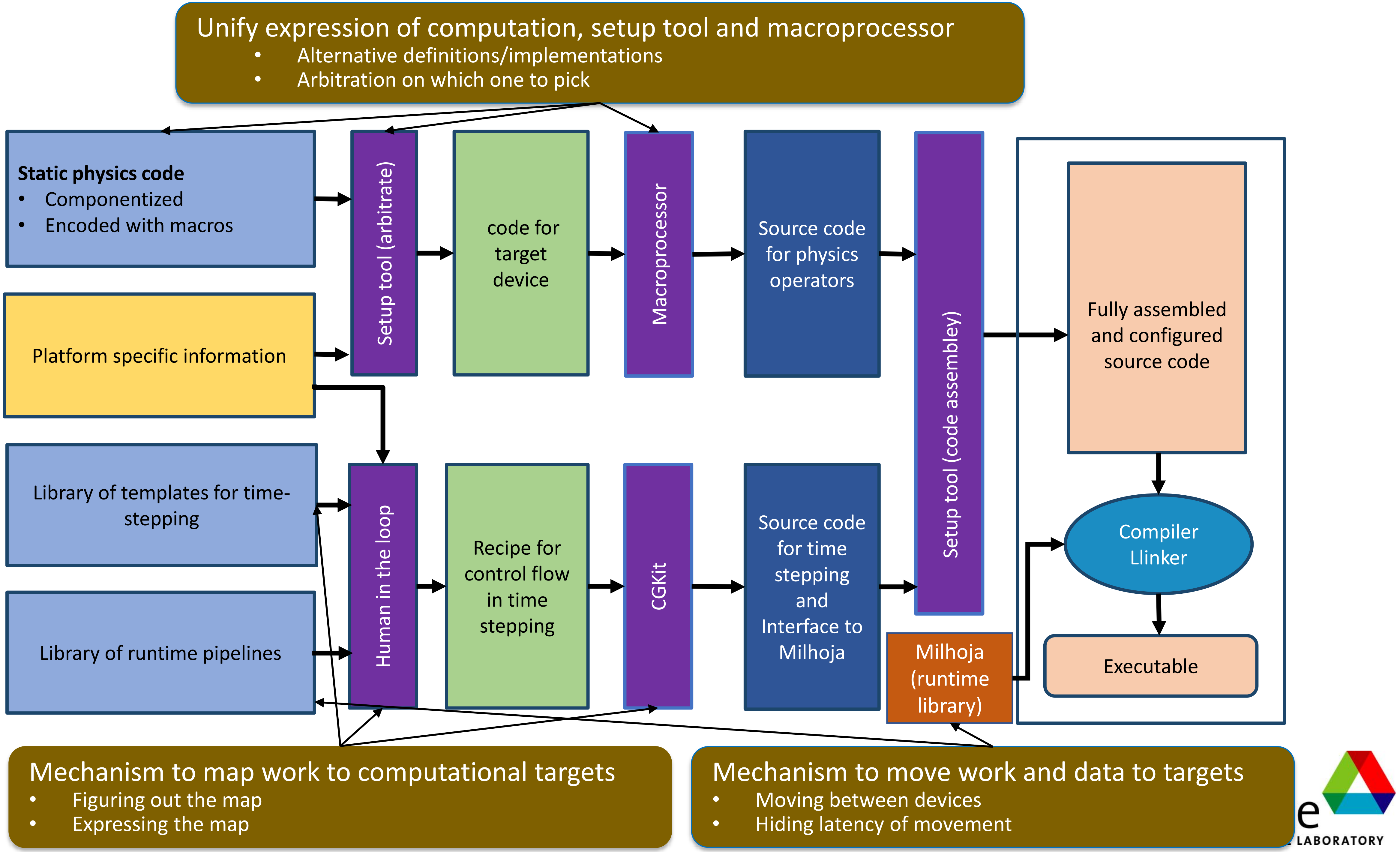
```
Subroutine hy_getFaceFlux(@M hy_ff_args)
  @M hy_ff_declare
  do dir=1,NDIM
    @M hy_set_loop
    @M hy_start_loop
      @M hy_fill_tmp_blk
      @M hy_reconstruct
      @M hy_riemann
      @M hy_save_fluxes
    @M hy_end_loop
end subroutine hy_getFaceFlux
```

Examples of CPU definitions

```
[hy_start_loop]
definition =
   @M loop_begin_2d(limits)

[hy_reconstruct]
definition =
   call reconstruct(@M hy_rec_args)
```

Examples of GPU definitions

```
[hy_set_loop]
definition =

[hy_reconstruct]
definition =
   @M loop_begin_3d(limits)
     call reconstruct(@M hy_rec_args)
   @M loop_end_3d
```



Unify expression of computation, setup tool and macroprocessor
Alternative definitions/implementations
Arbitration on which one to pick
Static physics code
Componentized
Encoded with macros
Platform specific information
Library of templates for time-stepping
Library of runtime pipelines
Setup tool (arbitrate)
Human in the loop
code for target device
Recipe for control flow in time stepping
Macroprocessor
CGKit
Source code for physics operators
Source code for time stepping and Interface to Milhoja
Setup tool (code assembley)
Milhoja (runtime library)
Fully assembled and configured source code
Compiler Llinker
Executable
Mechanism to map work to computational targets
Figuring out the map
Expressing the map
Mechanism to move work and data to targets
Moving between devices
Hiding latency of movement

The Toolchain

- Has been developed to minimize direct knowledge of Flash-X
- Some will be released as stand-alone tools
- Each one operates essentially independently
- Minimize the amount of recoding
  - In the code and in the tools
- A performance model to inform the optimizers

Porting to a new platform

- In an ideal world
  - Add to the library of runtime pipelines
  - Add to the library of recipes templates
  - Add to the knowledge base of the performance model
- In real world
  - Add variants for some solvers with alternative definitions of macros
- In the worst case
  - Develop new algorithms and add whole alternative implementation for some solvers