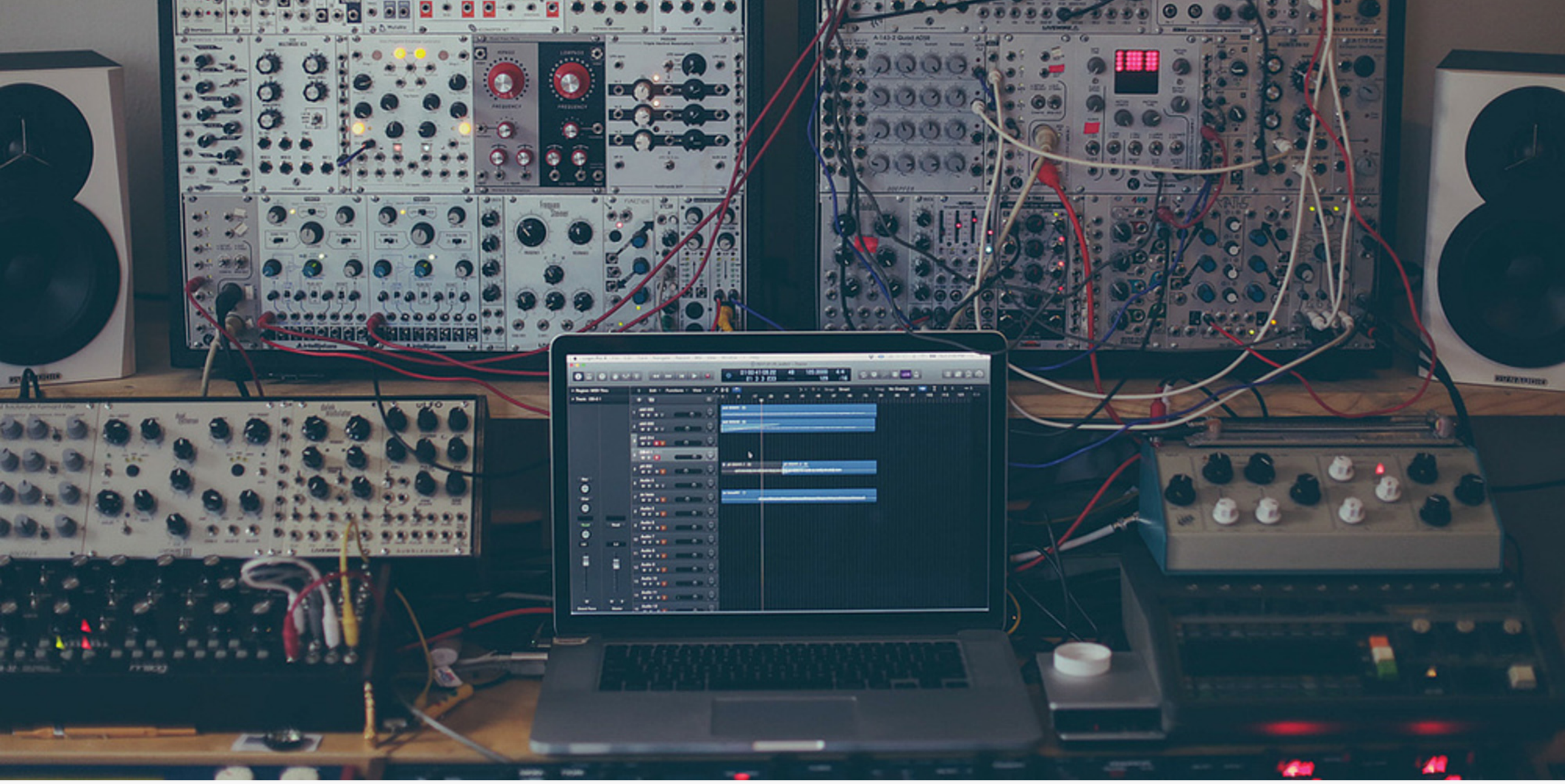


Hadrons: a Grid-powered measurement workflow management system

Antonin Portelli

19th of June 2019

Lattice 2019 - Wuhan, China

erc

Centre for Theoretical Physics · Higgs

The University of Edinburgh

THE UNIVERSITY *of* EDINBURGH

Photo: Alex Kacha for Sound on Sound

- Developers: Boyle, Cossu, Filaci, Portelli, Yamaguchi
- + Richtman, O'Haigan, Kettle, Erben, Marshall, Murphy, Jung, Lehner

Develop branch: excellent multicore CPU performance

- Contains functionality equivalent to QMP, QIO, QDP++, and much of Chroma
- SU(N), U(1), multiple fermion representations
- Gauge evolution: HMC, RHMC, EOFA
- Wilson, Clover, Domain wall, Mobius, + various 5D overlap
  - Improved staggered valence action
- Multiple solvers
  - Mixed precision Krylov solvers (CG, GMRES, BiCGstab …)
  - Block solvers
  - Polynomial preconditioned Lanczos & deflation
  - Experimental multigrid
- Split Grids for valence solves - avoid communication
    - NB: Not the same as trivial parallelism:
      - Eigenvectors held globally, deflate many RHS
      - Solves are local. 3x gain on Cori

feature/gpu-port

- Aim to target multiple accelerators (Nvidia, AMD, Intel Aurora system)

- Presently compile all C++ files through NVCC (no .cu files)
  - Minimise vendor specific code to a few macros
    - Study HIP, OpenMP 5.0, SyCL options for other targets
  - Assume host and device have a unified memory space (UVM, SVM)

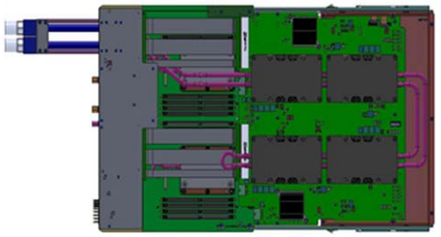
- (recent) Single source high performance kernels:
  - same kernel gives good performance on GPU *and* CPU

- All Fermion operators now accelerated
  - Wilson, Wilson5D, Improved Staggered
  - 5D Mobius terms also

- Entire CG runs on GPU. Full (R)HMC next target.

- Intra node GPU-GPU communication uses NVlink ; multi node under test

HPE ICE XA780i Quad Volta blade
24^4 5D single precision Wilson operator
1GPU 1.45 TF/s
4GPU 4.5 TF/s



Lattice measurements

- In QCD basically:
  Solver - Propagators - Contractions

- More and more involved:
  Deflation, AMA, 4-pt functions…

Things I did not want to repeat

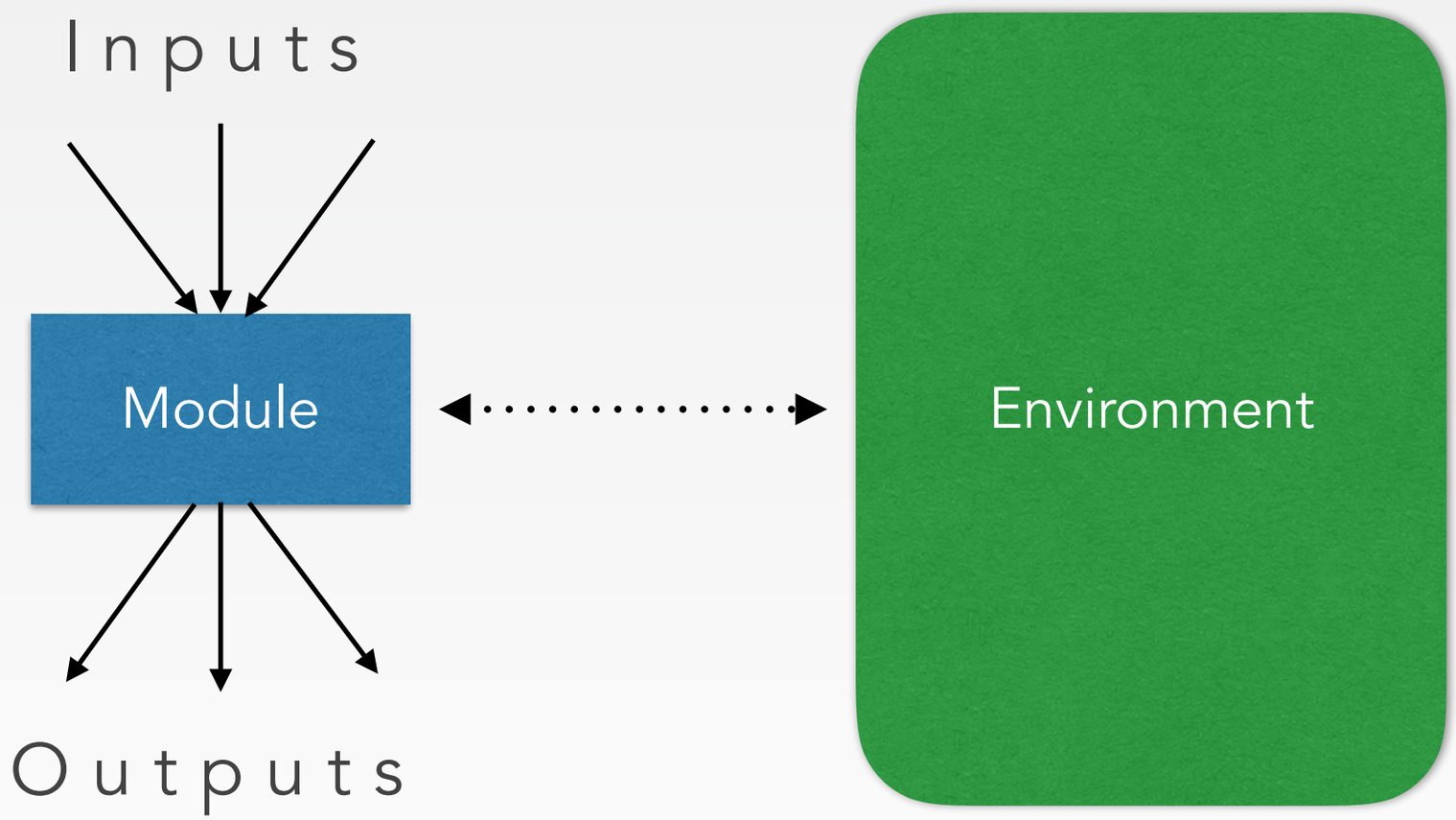
(no hard feelings, just trying to improve 🙂)

- **Very complicated inputs.**
  (100k lines XML files, machine generated inputs)
- **Very rigid programs.**
  (lots of global variables scattered in the program)
- **No safety net.**
  (dependency between steps, memory consumption)

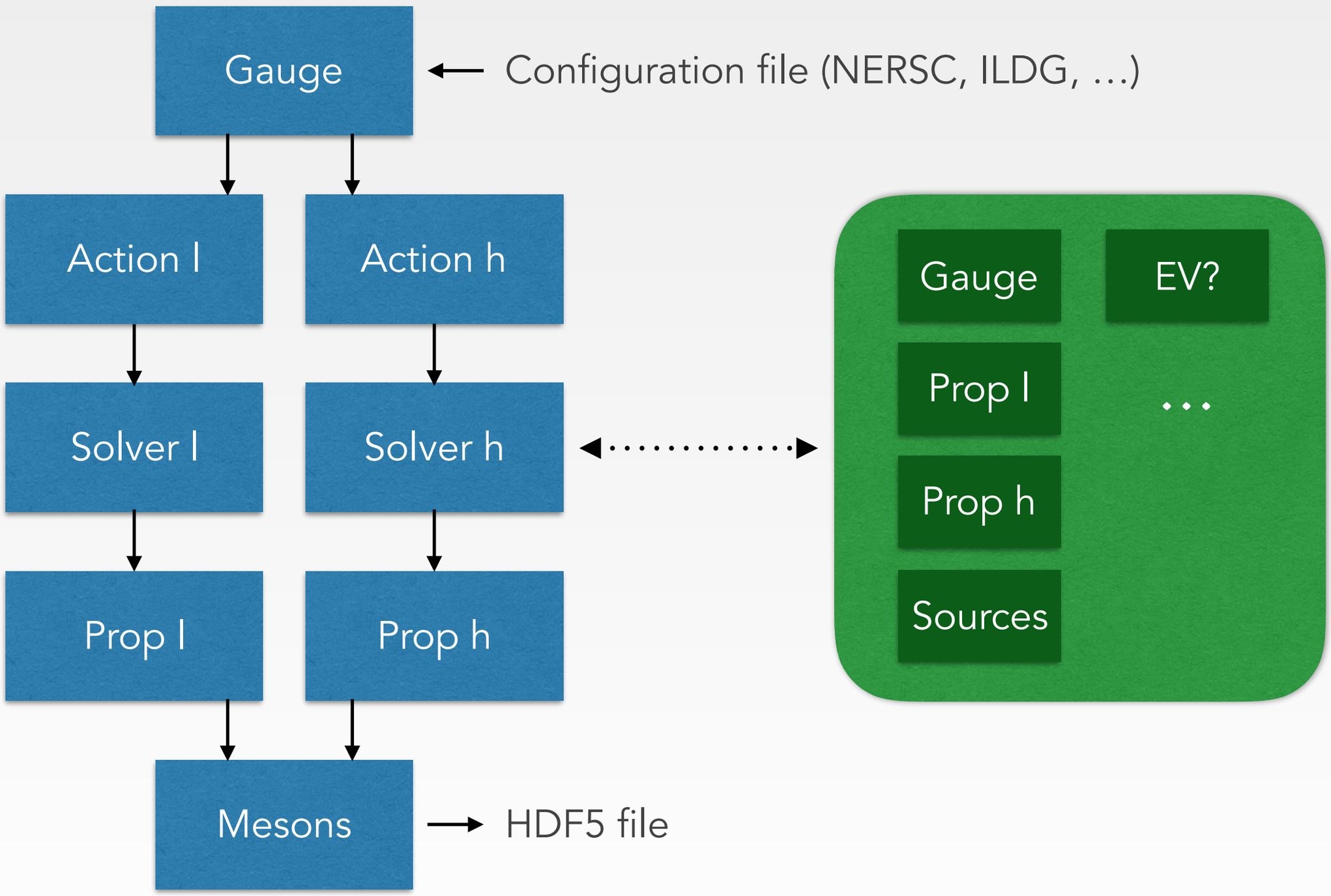
Directions for solutions

- **High modularity** — building a new project is easy.
- **Flexible I/O & control** — highly customisable input.
- **Automatic scheduling** — more self-consistency checks.

Measurement data flow



Measurement data flow



Scheduling

- Dataflow diagram: DAG.
- Dependency solving: DAG topological sort.
- Memory optimisation 1: garbage collection.
- Memory optimisation 2: constrained topological sort.
- Very likely NP-hard problem: need a heuristic solution.
- So far: genetic algorithm minimising high-water function on the space of topological sorts.
- Find a schedule in O(10 min) for big graphs, work in progress.

Productions workflows

- Heavy-light semi-leptonic decays using distillation
  O(1000) modules — Talk: Felix Erben
- Rare kaon decays
  O(1000) modules — Talk: Fionn Ó hÓgáin
- All-to-all IB corrections
  O(10) modules — Talk: James Richings
- IB corrections to light leptonic decays
  O(1000) modules — Talk: A.P.
- Holographic cosmology
  O(10) modules — Talk: Joseph Lee

Flexible control

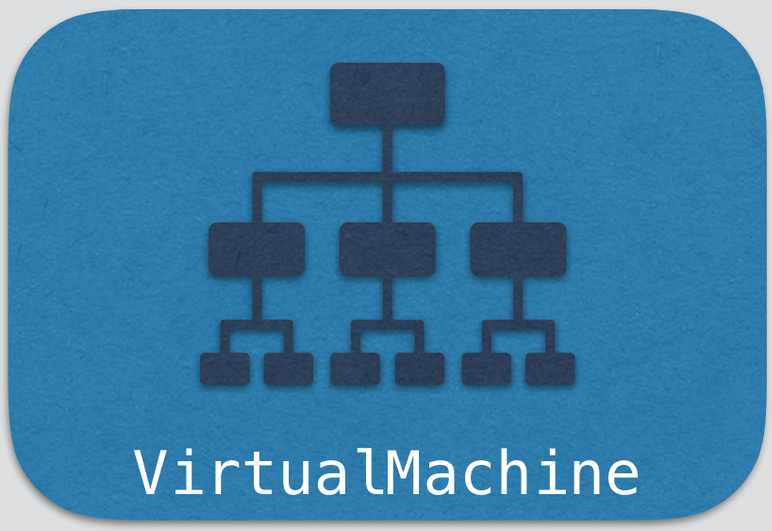
Hardcoded C++ ASCII input (*e.g.* XML)

- Hardcoded: risk of code (and bug) duplication.
- ASCII input: too general, complicated input.
- Matter of taste: user should be able to choose.
- Achieved with modules + Grid serialisable I/O.

Full structure

- Module DAG
- Scheduling & garbage collection

- Named object store
- Memory footprint aware



Application

High-level interface for VM setup

Hadrons::

```
using namespace Grid;
using namespace Hadrons;

int main(int argc, char *argv[])
{
    // initialization ////////////////////////////////////////////////////////
    Grid_init(&argc, &argv);
    HadronsLogError.Active(GridLogError.isActive());
    HadronsLogWarning.Active(GridLogWarning.isActive());
    HadronsLogMessage.Active(GridLogMessage.isActive());
    HadronsLogIterative.Active(GridLogIterative.isActive());
    HadronsLogDebug.Active(GridLogDebug.isActive());
    LOG(Message) << "Grid initialized" << std::endl;

    // run setup /////////////////////////////////////////////////////////////
    Application              application;
    std::vector<std::string> flavour = {"l", "s", "c1", "c2", "c3"};
    std::vector<double>      mass    = {.01, .04, .2  , .25 , .3  };

    // global parameters
    Application::GlobalPar globalPar;
    globalPar.trajCounter.start = 1500;
    globalPar.trajCounter.end   = 1520;
    globalPar.trajCounter.step  = 20;
    globalPar.seed              = "1 2 3 4";
    application.setPar(globalPar);

    // ... module enumeration

    // execution
    application.saveParameterFile("spectrum.xml");
    application.run();

    // epilogue
    LOG(Message) << "Grid is finalizing now" << std::endl;
    Grid_finalize();

    return EXIT_SUCCESS;
}
```

tests/hadrons/test_hadrons_spectrum.cc
(https://github.com/paboyle/Grid GPL v2)

```
// sources
MSource::Z2::Par z2Par;
z2Par.tA = 0;
z2Par.tB = 0;
application.createModule<MSource::Z2>("z2", z2Par);
MSource::Point::Par ptPar;
ptPar.position = "0 0 0 0";
application.createModule<MSource::Point>("pt", ptPar);
// sink
MSink::Point::Par sinkPar;
sinkPar.mom = "0 0 0";
application.createModule<MSink::ScalarPoint>("sink", sinkPar);

// set fermion boundary conditions to be periodic space, antiperiodic time.
std::string boundary = "1 1 1 -1";

for (unsigned int i = 0; i < flavour.size(); ++i)
{
    // actions
    MAction::DWF::Par actionPar;
    actionPar.gauge = "gauge";
    actionPar.Ls    = 12;
    actionPar.M5    = 1.8;
    actionPar.mass  = mass[i];
    actionPar.boundary = boundary;
    application.createModule<MAction::DWF>("DWF_" + flavour[i], actionPar);

    // solvers
    MSolver::RBPrecCG::Par solverPar;
    solverPar.action   = "DWF_" + flavour[i];
    solverPar.residual = 1.0e-8;
    application.createModule<MSolver::RBPrecCG>("CG_" + flavour[i],
                                                solverPar);

    // propagators
    MFermion::GaugeProp::Par quarkPar;
    quarkPar.solver = "CG_" + flavour[i];
    quarkPar.source = "pt";
    application.createModule<MFermion::GaugeProp>("Qpt_" + flavour[i], quarkPar);
    quarkPar.source = "z2";
    application.createModule<MFermion::GaugeProp>("QZ2_" + flavour[i], quarkPar);
}
```

`tests/hadrons/test_hadrons_spectrum.cc`
(https://github.com/paboyle/Grid GPL v2)

```cpp
// constructor ////////////////////////////////////////////////////////////////
template <typename FImpl>
TPoint<FImpl>::TPoint(const std::string name)
: Module<PointPar>(name)
{}

// dependencies/products //////////////////////////////////////////////////////
template <typename FImpl>
std::vector<std::string> TPoint<FImpl>::getInput(void)
{
    std::vector<std::string> in;

    return in;
}

template <typename FImpl>
std::vector<std::string> TPoint<FImpl>::getOutput(void)
{
    std::vector<std::string> out = {getName()};

    return out;
}

// setup //////////////////////////////////////////////////////////////////////
template <typename FImpl>
void TPoint<FImpl>::setup(void)
{
    envCreateLat(PropagatorField, getName());
}

// execution //////////////////////////////////////////////////////////////////
template <typename FImpl>
void TPoint<FImpl>::execute(void)
{
    LOG(Message) << "Creating point source at position [" << par().position
                 << "]" << std::endl;

    std::vector<int> position = strToVec<int>(par().position);
    auto             &src     = envGet(PropagatorField, getName());
    SitePropagator   id;

    id  = 1.;
    src = zero;
    pokeSite(id, src, position);
}
```

Hadrons/Modules/MSource/Point.hpp
(https://github.com/paboyle/Grid GPL v2)

Outlook

- Grid + Hadrons: cross-platform, high-performance lattice software.
- Grid: high-performance data parallel library.
- Hadrons: high-level interface focused on physics measurements.
- Modular structure, with automatic scheduling.
  Aimed at fast & future-proof project development.
- Already used in production for a wide variety of calculations.

Perspectives

- Still untested on GPUs, soon!
- More modules (Baryons, staggered, multi-grid, …).
- Scheduler optimisations.

Hadrons contributors

Grid dev. team
Peter Boyle
(Guido Cossu)
Antonin Portelli
Azusa Yamaguchi

U. of Edinburgh
Felix Erben
Vera Gülpers
Julia Kettle
Joseph Lee
Michael Marshall
Fionn Ó hÓgáin
Tobi Tsang

U. of Southampton
Andreas Jüttner
(Andrew Lawson)
James Richings

U. of Connecticut
Tom Blum
Michael Abramczyk

Thank you!

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 757646.