GAMBIT-light

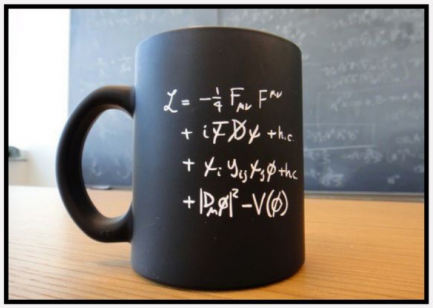
Anders Kvellestad

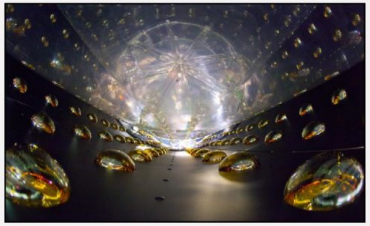
8 August 2023, N-PACT 2023

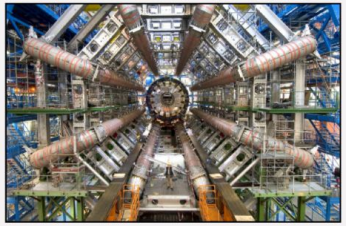
UNIVERSITETET
I OSLO

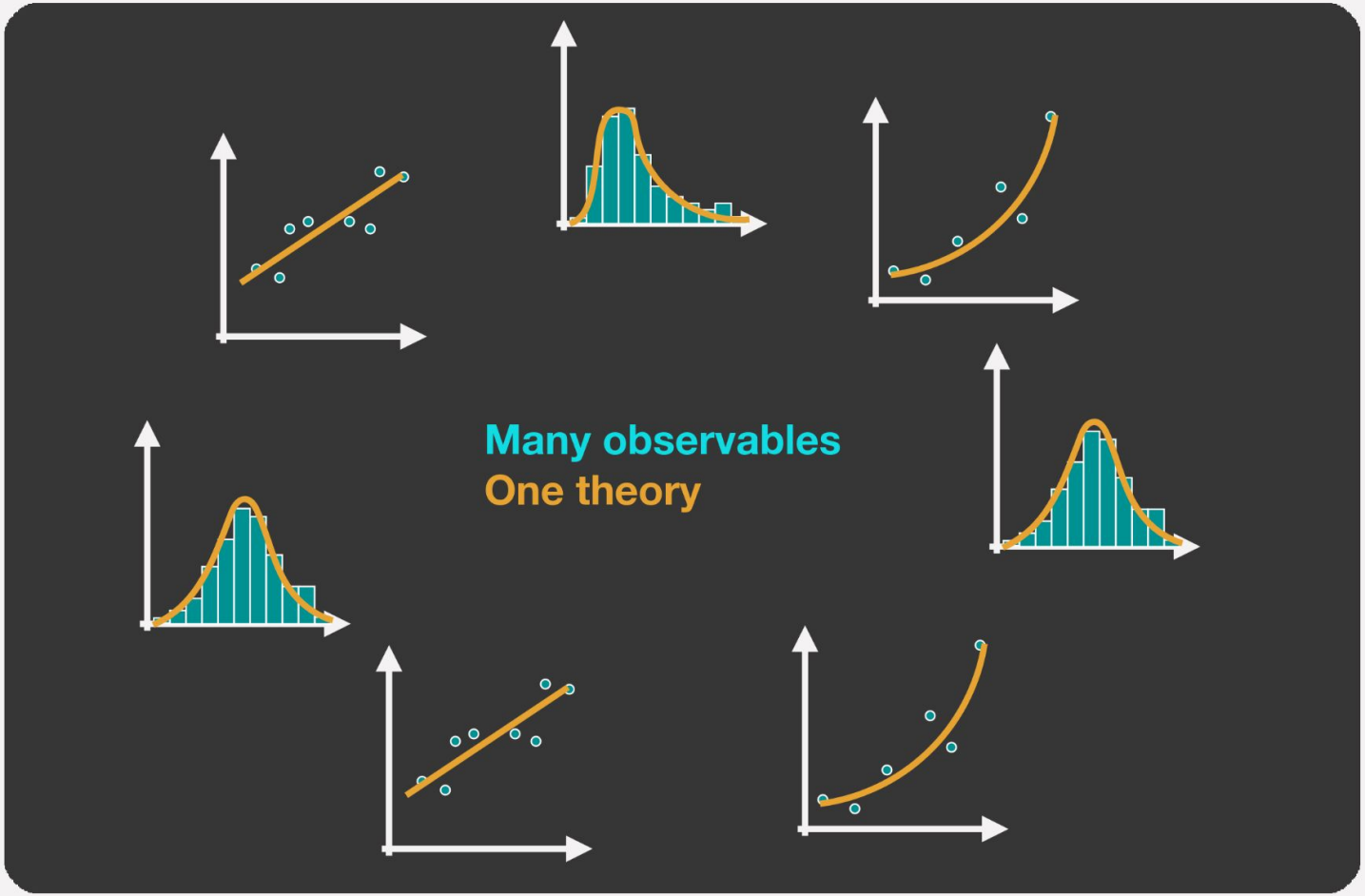
1. What is GAMBIT? (Once more, with feeling)
2. Quick GAMBIT news
3. **GAMBIT-light:** What? Why?
4. Particles, pandemics and parameter spaces

1. What is GAMBIT?


$\mathcal{L} = -\frac{1}{4} F_{\mu\nu} F^{\mu\nu}$
$+ i\bar{\psi} \not{D} \psi + h.c$
$+ \lambda_i y_{ijk} \chi_j \chi_k \phi + h.c$
$+ |D_\mu \phi|^2 - V(\phi)$






Many observables
One theory

How to test your model against data?
The **likelihood** is key!

$$p(\boldsymbol{D}_{\rm obs}|\boldsymbol{\theta}) \equiv L(\boldsymbol{\theta})$$

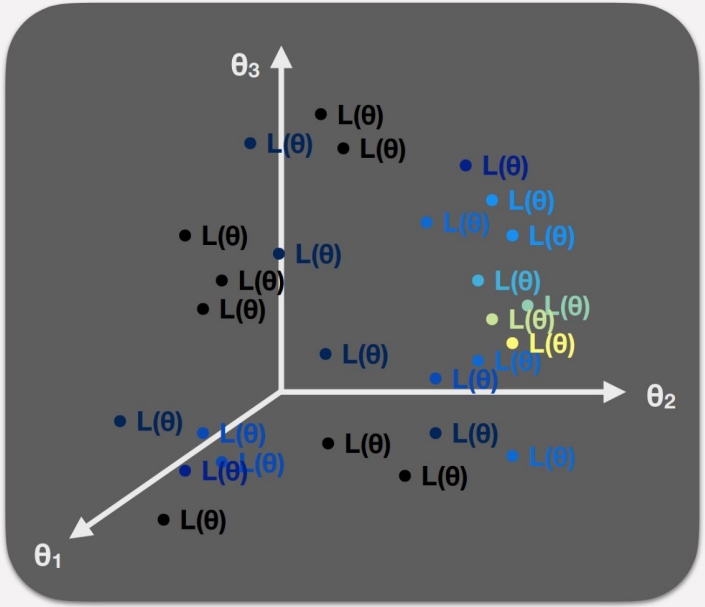
$$p(\boldsymbol{\theta}|\boldsymbol{D}_{\rm obs}) = \frac{p(\boldsymbol{D}_{\rm obs}|\boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\boldsymbol{D}_{\rm obs})}$$

$$p(\boldsymbol{\theta}|\boldsymbol{D}_{\rm obs}) = \frac{L(\boldsymbol{\theta})\, \pi(\boldsymbol{\theta})}{Z}$$

$$p(\boldsymbol{D}_{\rm obs}|\boldsymbol{\theta})$$

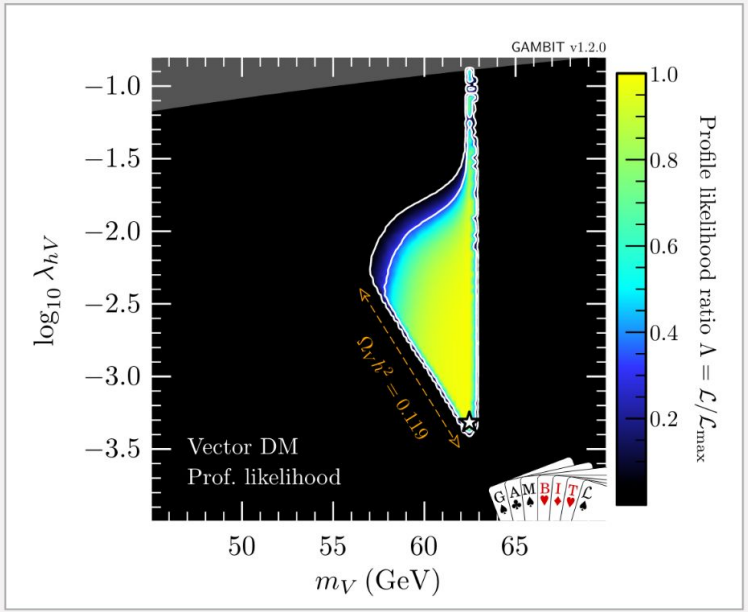
+ assumptions/simulations
of hypothetical data

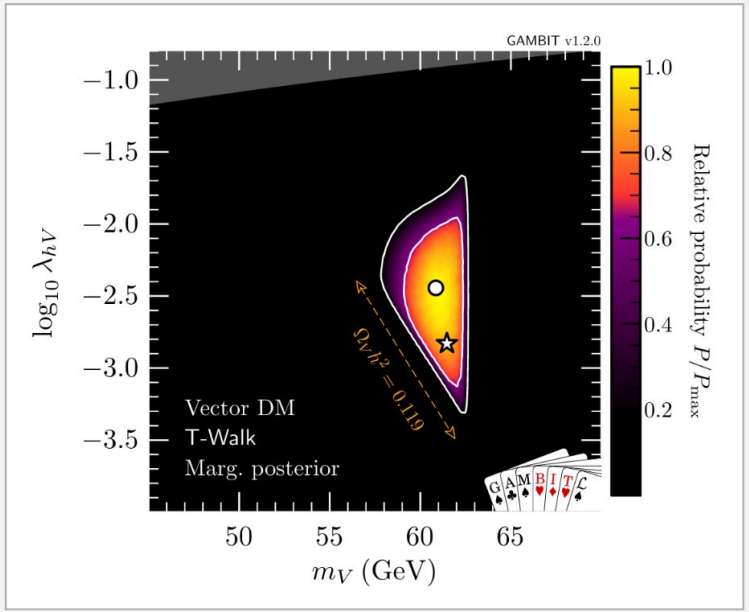
- **Explore the model parameter space** (θ_1, θ_2, θ_3, …)
- At every point θ: **compute all predictions(θ) → evaluate likelihood L(θ)**



- Region of highest L(θ) or lnL(θ): **model's best simultaneous fit to all data**
(but not necessarily a *good* fit, or the most probable θ…)

Typical result:
Parameter estimation, presented as **profile likelihood** and/or **posterior density** plots

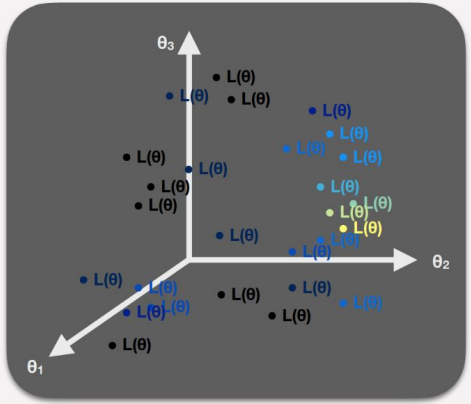




[arxiv:1808.10465]

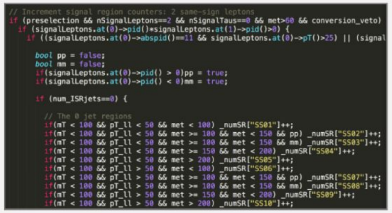
Computational challenges:

- Need **smart exploration** of parameter space
- Need **fast theory calculations**
- Need **fast simulations of experiments** (e.g. LHC)
- Need **sufficiently detailed likelihoods** or **full statistical models**



Search for electroweak production of charginos and neutralinos in multilepton final states in proton-proton collisions at $\sqrt{s} = 13$ TeV

The CMS collaboration



```
// Increment signal region counters if event has passed
if (preselection && nSignalLeptons==2 && nSignalTaus==0 && met>50 && conversion_veto)
  if (signalLeptons.at(0)->pid()*signalLeptons.at(1)->pid()>0) {
    if ((signalLeptons.at(0)->absspid()==11 && signalLeptons.at(0)->pT()>25) || (signal

      bool pp = false;
      bool mm = false;
      if(signalLeptons.at(0)->pid() > 0)pp = true;
      if(signalLeptons.at(0)->pid() < 0)mm = true;

      if (num_ISRjets==0) {

        // The 0 jet regions
        if(mT < 100 && pT_ll < 50 && met < 100) _numSR["SS01"]++;
        if(mT < 100 && pT_ll < 50 && met >= 100 && met < 150 && pp) _numSR["SS02"]++;
        if(mT < 100 && pT_ll < 50 && met >= 100 && met < 150 && mm) _numSR["SS03"]++;
        if(mT < 100 && pT_ll < 50 && met >= 150 && met < 200) _numSR["SS04"]++;
        if(mT < 100 && pT_ll < 50 && met > 200) _numSR["SS05"]++;
        if(mT < 100 && pT_ll > 50 && met < 100) _numSR["SS06"]++;
        if(mT < 100 && pT_ll > 50 && met >= 100 && met < 150 && pp) _numSR["SS07"]++;
        if(mT < 100 && pT_ll > 50 && met >= 100 && met < 150 && mm) _numSR["SS08"]++;
        if(mT < 100 && pT_ll > 50 && met >= 150 && met < 200) _numSR["SS09"]++;
        if(mT < 100 && pT_ll > 50 && met > 200) _numSR["SS10"]++;
```

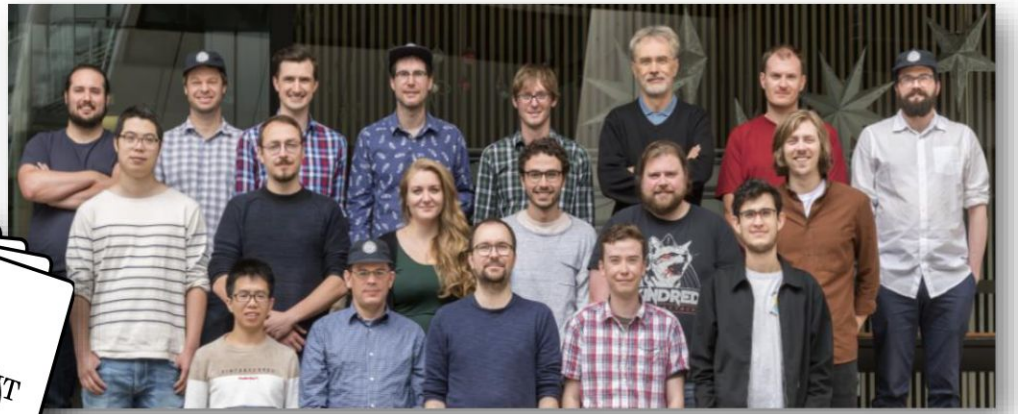
Some code infrastructure challenges:

- Need **different parameter scanning algorithms**
- Need **model-agnostic core framework**
- Need to interface ***many*** **external physics codes**
- Need **massive parallelisation**…
- …which implies a need for **diskless interfacing**
- …which implies a need to **stop external codes from calling STOP and kill your 10,000-CPU scan**… :)

GAMBIT: The Global And Modular BSM Inference Tool

gambit.hepforge.org github.com/GambitBSM EPJC 77 (2017) 784 arXiv:1705.07908

- Extensive model database, beyond SUSY
- Fast definition of new datasets, theories
- Extensive observable/data libraries
- Plug&play scanning/physics/likelihood packages
- Various statistical options (frequentist /Bayesian)
- Fast LHC likelihood calculator
- Massively parallel
- Fully open-source



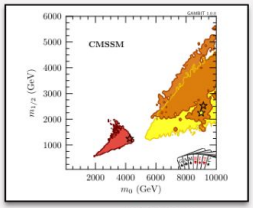
Members of: ATLAS, Belle-II, CLiC, CMS, CTA, Fermi-LAT, DARWIN, IceCube, LHCb, SHiP, XENON

Authors of: BubbleProfiler, Capt'n General, Contur, DarkAges, DarkSUSY, DDCalc, DirectDM, Diver, EasyScanHEP, ExoCLASS, FlexibleSUSY, gamLike, GM2Calc, HEPLike, IsaTools, MARTY, nuLike, PhaseTracer, PolyChord, Rivet, SOFTSUSY, Superlso, SUSY-AI, xsec, Vevacious, WIMPSim

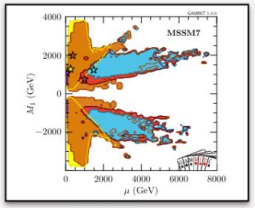
Recent collaborators: V Ananyev, P Athron, N Avis-Kozar, C Balázs, A Beniwal, S Bloor, LL Braseth, T Bringmann, A Buckley, J Butterworth, J-E Camargo-Molina, C Chang, M Chrzaszcz, J Conrad, J Cornell, M Danninger, J Edsjö, T Emken, A Fowlie, T Gonzalo, W Handley, J Harz, S Hoof, F Kahlhoefer, A Kvellestad, M Lecroq, P Jackson, D Jacob, C Lin, FN Mahmoudi, G Martinez, H Pacey, MT Prim, T Procter, F Rajec, A Raklev, JJ Renk, R Ruiz, A Scaffidi, P Scott, N Serra, P Stöcker, W. Su, J Van den Abeele, A Vincent, C Weniger, A Woodcock, M White, Y Zhang ++

80+ participants in many experiments and numerous major theory codes

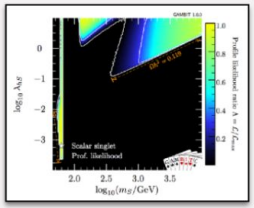
2. Quick GAMBIT news



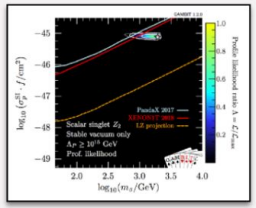
GUT-scale SUSY: 1705.07935



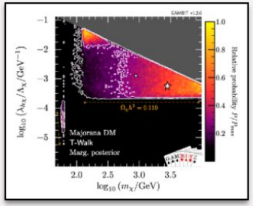
MSSM7: 1705.07917



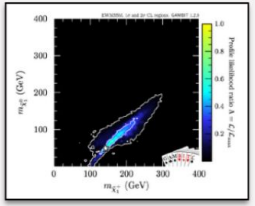
Scalar Higgs portal DM: 1705.07931



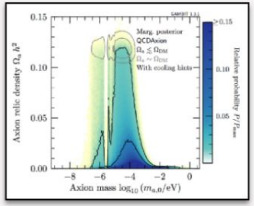
Scalar Higgs portal DM w/ vac. stability: 1806.11281



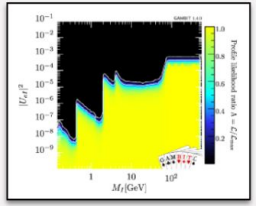
Vector and fermion Higgs portal DM: 1808.10465



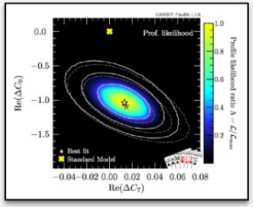
EW-MSSM: 1809.02097



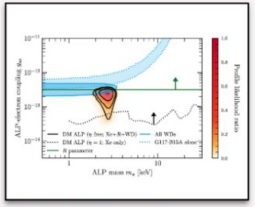
Axion-like particles: 1810.07192



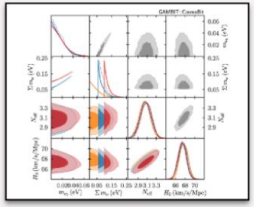
Right-handed neutrinos: 1908.02302



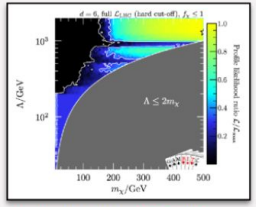
Flavour EFT: 2006.03489



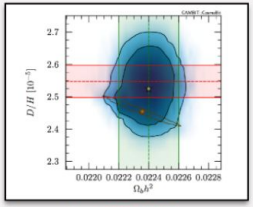
More axion-like particles: 2007.05517



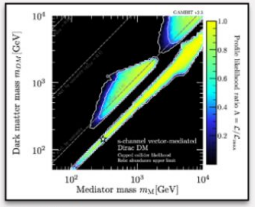
Neutrinos and cosmo: 2009.03287



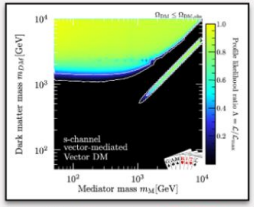
Dark matter EFTs: 2106.02056



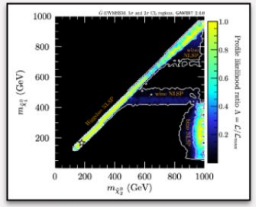
Cosmo ALPs: 2205.13549



Simplified DM, scalar/fermion: 2209.13266



Simplified DM, vector: 2303.08351

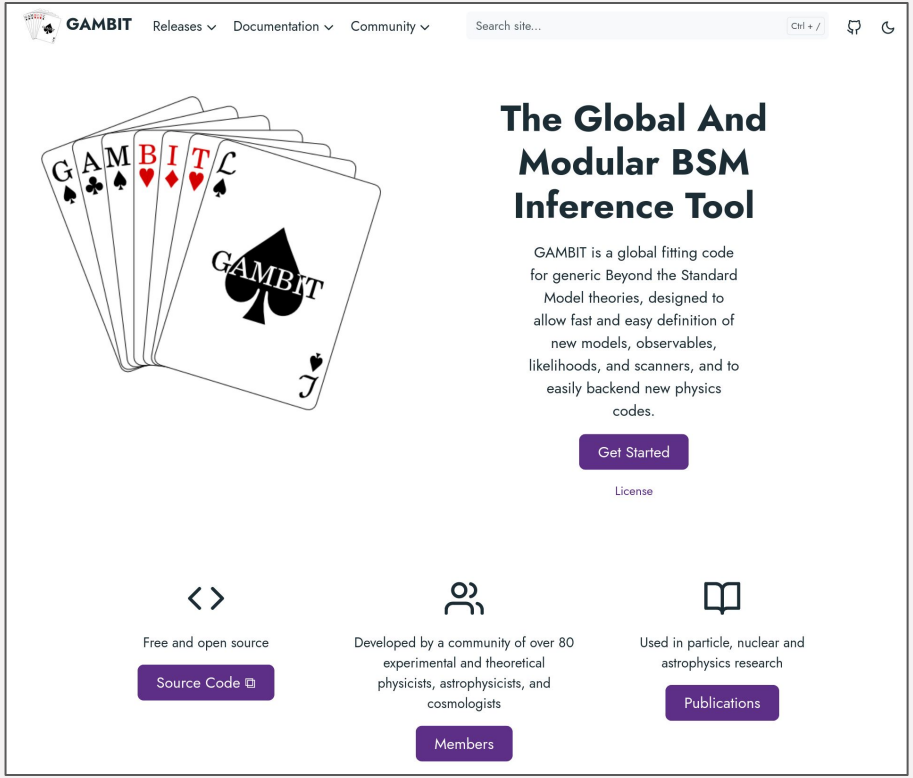


EW-MSSM w/ light gravitino: 2303.09082

- GAMBIT studies since 2017 have required **>200M CPU hours** in total
- Recently got a *EuroHPC Extreme Scale Access* computing grant of **2 x 80M CPU hours**
- On the new **LUMI supercomputer** (Finland)
- Grant for mapping out pheno implications of combined LHC results and prospects for future runs/colliders

EuroHPC
Joint Undertaking

- New web page! **gambitbsm.org**
- Soon ready #1: **GAMBIT + Python scanners**
- Soon ready #2: **GAMBIT-light**



3. GAMBIT-light

- GAMBIT-light: **GAMBIT without all the physics**
- A lightweight yet powerful tool for statistical fits and optimisation tasks
- *What GAMBIT-light is not:* A full-blown tool for global fits in <your discipline> — for that you'd want more of the full GAMBIT functionality
- Key design principles:
  - Users should never need to modify and rebuild any GAMBIT code
  - Minimise the extra maintenance work for GAMBIT developers

Maiken Pedersen (UiO)

Marcin Krotkiewski (UiO)

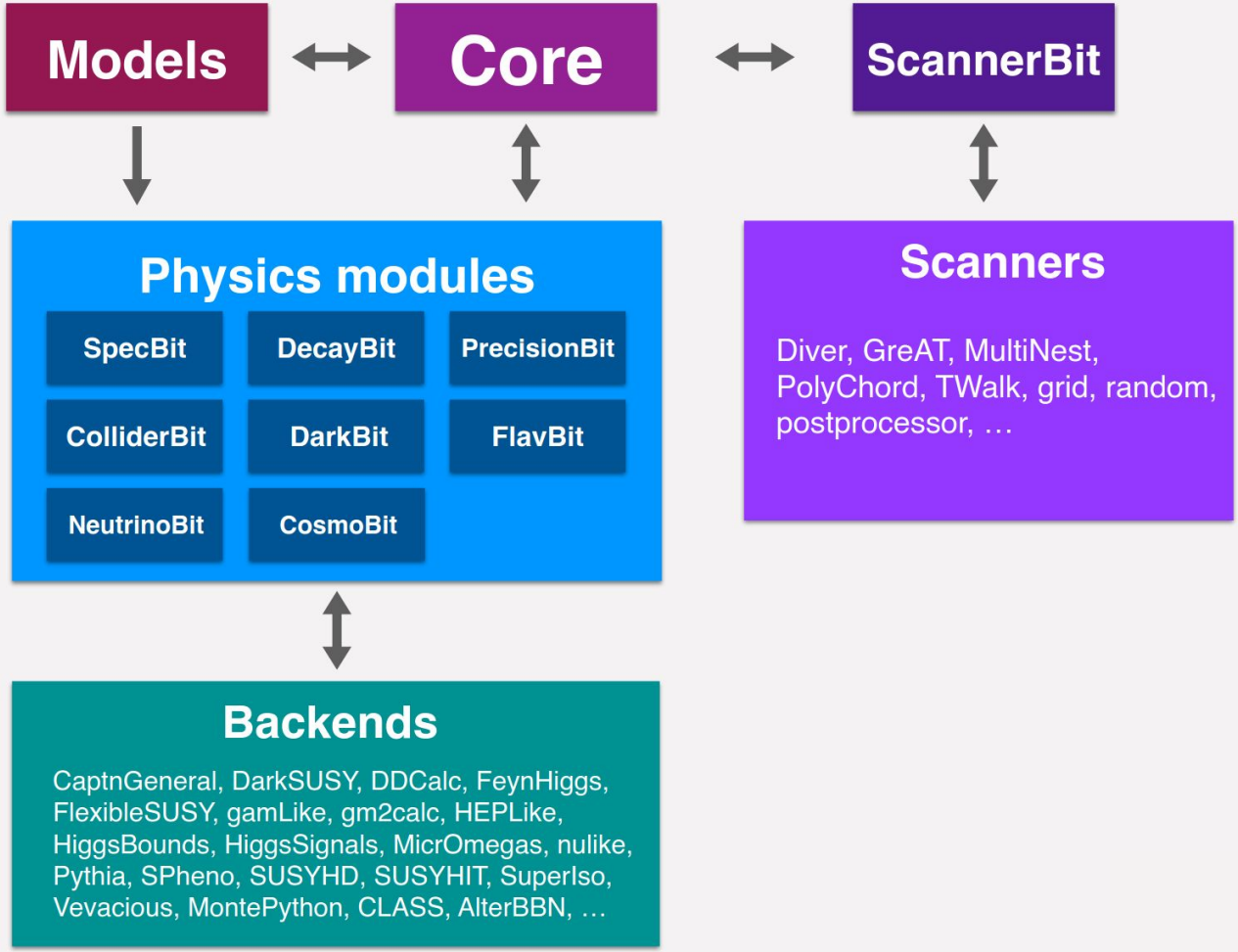
Brainstorming, early code drafts, testing:

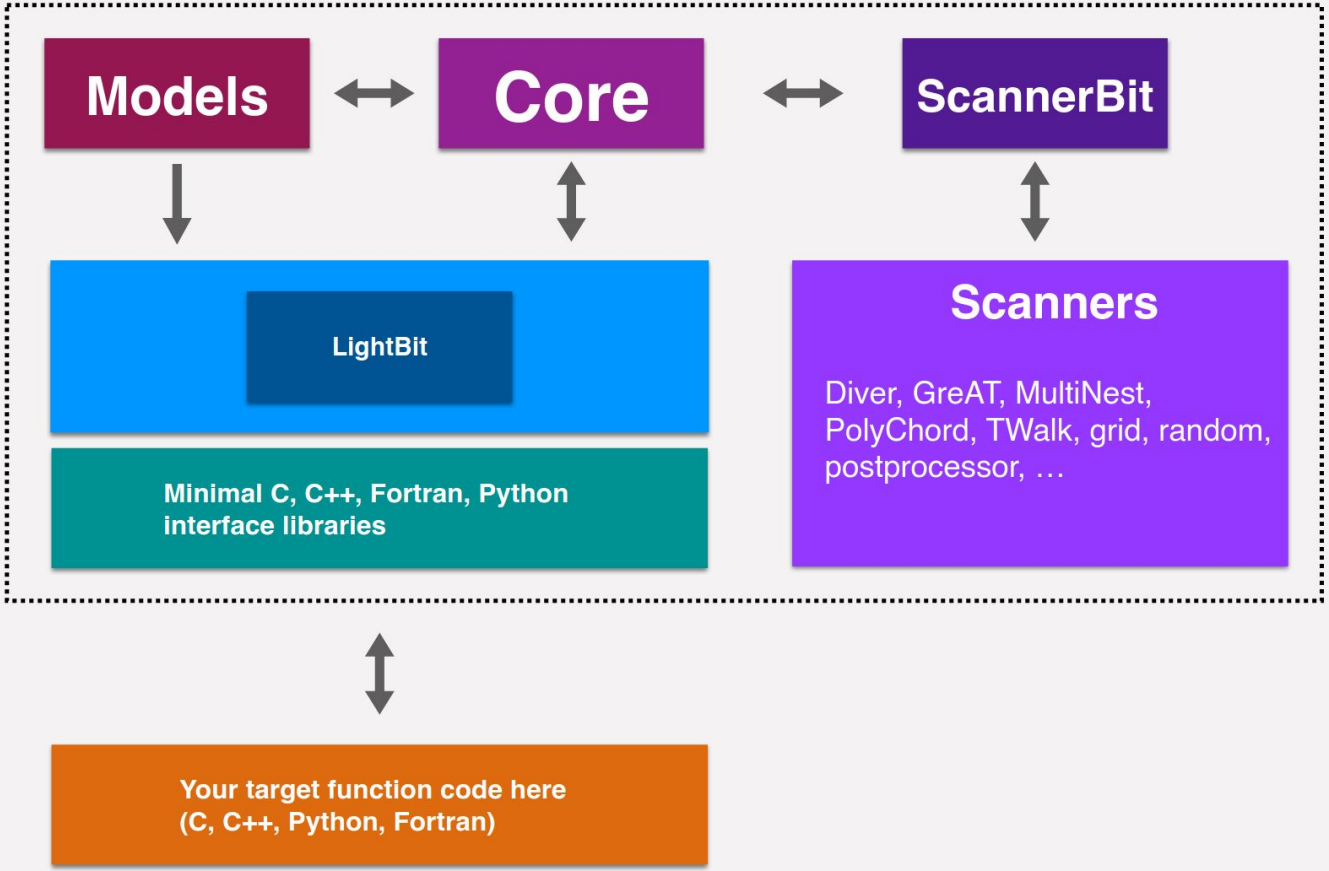
Janina Renk, **Fabio Zeiser**, **Andreas Mjøs**, ++

- **Background:**
  - We designed GAMBIT to be very general and physics-agnostic
  - We put a lot of effort into the main code framework
    (core framework, config files, scanners, output system, logging, error handling, build system, …)
  - → GAMBIT *can* be used for optimisation/fits outside particle/astro/cosmo

- **Practical experience:**
  - GAMBIT is a particle physics power tool → fairly heavyweight
  - Considerable threshold for non-experts to pick up and use/modify
  - In particular: frequent and slow recompilation kills the flow of the early development/experimentation stage of projects

- **Motivation for GAMBIT-light:**
  - Help projects outside particle/astro/cosmo use GAMBIT
  - Within particle/astro/cosmo:
    suitable for quick experimentation, MSc projects, “not-so-global fits”, etc.


Models
Core
ScannerBit
Physics modules
SpecBit
DecayBit
PrecisionBit
ColliderBit
DarkBit
FlavBit
NeutrinoBit
CosmoBit
Scanners
Diver, GreAT, MultiNest, PolyChord, TWalk, grid, random, postprocessor, …
Backends
CaptnGeneral, DarkSUSY, DDCalc, FeynHiggs, FlexibleSUSY, gamLike, gm2calc, HEPLike, HiggsBounds, HiggsSignals, MicrOmegas, nulike, Pythia, SPheno, SUSYHD, SUSYHIT, SuperIso, Vevacious, MontePython, CLASS, AlterBBN, …


Models
Core
ScannerBit
LightBit
Minimal C, C++, Fortran, Python
interface libraries
Scanners
Diver, GreAT, MultiNest,
PolyChord, TWalk, grid, random,
postprocessor, …
Your target function code here
(C, C++, Python, Fortran)

1. Build GAMBIT once

```
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DWITH_MPI=On -DCMAKE_CXX_COMPILER=g++-11 -DCMAKE_C_COMPILER=gcc-11
make -jN scanners   # where N is the number of cores to use for the build, e.g. 4
cmake ..            # this step is needed for GAMBIT to detect the built scanners
make -jN gambit
```

2. Develop your target/likelihood function code

```python
# To import gambit_light_interface, first append the directory containing
# gambit_light_interface.so to sys.path. (Alternatively, add this directory
# to the PYTHONPATH environment variable.)
import sys
import os
current_dir = os.path.dirname(os.path.abspath(__file__))
sys.path.append(os.path.join(current_dir, "../lib"))
import gambit_light_interface as gambit_light


# User-side log-likelihood function, which can be called by GAMBIT-light
def user_loglike(input_names, input_vals, output):

    # Make a dictionary of the inputs?
    input = {input_names[i]: input_vals[i] for i in range(len(input_names))}

    # Error handling: Report an invalid point using gambit_light.invalid_point.
    # gambit_light.invalid_point("This input point is no good.")

    # Error handling: Report a warning using gambit_light.warning.
    gambit_light.warning("Some warning.")

    # Error handling: Report an error using gambit_light.error.
    # gambit_light.error("Some error.")

    # Error handling: Error handling, alternative to using gambit_light.error: Throw an exception.
    # raise Exception("Some exception.")

    # Compute loglike
    loglike = input["param_name_1"] + input["param_name_2"] + input["param_name_4"]

    # Save some extra outputs
    output["py_user_loglike_output_1"] = 1
    output["py_user_loglike_output_2"] = 2
    output["py_user_loglike_output_3"] = 3

    return loglike

```

2. Develop your target/likelihood function code
(C++/C/Fortran: build as shared library)

```cpp
#include "gambit_light_interface.h"

// User-side log-likelihood function, which can be called by GAMBIT-light.
double user_loglike(const std::vector<std::string>& input_names,
                    const std::vector<double>& input_vals,
                    std::map<std::string,double>& output)
{

    // Make a map of the inputs?
    std::map<std::string,double> input;
    for (size_t i = 0; i < input_names.size(); i++)
    {
        input[input_names[i]] = input_vals[i];
    }

    // Error handling: Report an invalid point using gambit_light_invalid_point.
    // gambit_light_invalid_point("This input point is no good.");

    // Error handling: Report a string warning using gambit_light_warning.
    gambit_light_warning("Some warning.");

    // Error handling: Report an error using gambit_light_error.
    // gambit_light_error("Some error.");

    // Error handling, alternative to using gambit_light_error: Throw a runtime_error.
    // throw std::runtime_error("Some runtime_error.");

    // Compute loglike
    double loglike = input.at("param_name_2") + input.at("param_name_3");

    // Save some extra outputs
    output["cpp_user_loglike_output_1"] = 1;
    output["cpp_user_loglike_output_2"] = 2;
    output["cpp_user_loglike_output_3"] = 3;

    return loglike;
}

GAMBIT_LIGHT_REGISTER_LOGLIKE(user_loglike)

```

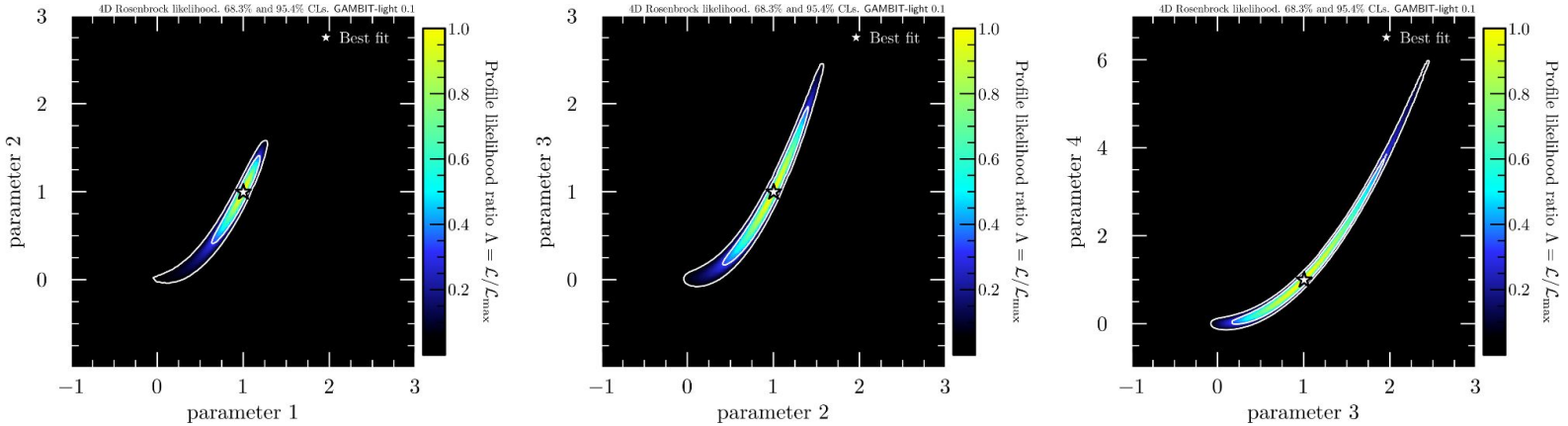
3. Configure GAMBIT run with a YAML file

```yaml
UserModel:

  p1:
    name: param_name_1
    prior_type: flat
    range: [0.0, 5.0]
  p2:
    name: param_name_2
    prior_type: flat
    range: [0.0, 5.0]
  p3:
    name: param_name_3
    fixed_value: 3.0
  p4:
    name: param_name_4
    same_as: UserModel::p1
  p5-p7:
    name: param_name_
    prior_type: flat
    range: [-1.0, 1.0]

UserLogLikes:

  py_user_loglike:
    lang: python
    user_lib: gambit_light_interface/example_python/example.py
    func_name: user_loglike
    output:
      - py_user_loglike_output_1
      - py_user_loglike_output_2
      - py_user_loglike_output_3

  cpp_user_loglike:
    lang: c++
    user_lib: gambit_light_interface/example_cpp/example.so
    func_name: user_loglike
    input:
      - param_name_2
      - param_name_3
    output:
      - cpp_user_loglike_output_1
      - cpp_user_loglike_output_2
      - cpp_user_loglike_output_3

```

4. Run GAMBIT

```
mpiexec -np 4 ./gambit -f yaml_files/your_configuration_file.yaml
```

5. Modify your own code, rerun GAMBIT, modify your own code, rerun GAMBIT, …



6. Analyse output samples (saved in HDF5 or ascii format)

Also: user-supplied prior transformation

Python

```python
# User-side prior transform function, which can be called by GAMBIT-light.
def user_prior(input_names, input_vals, output):

    for i,v in enumerate(input_vals):
        output[i] = v * 10.

```

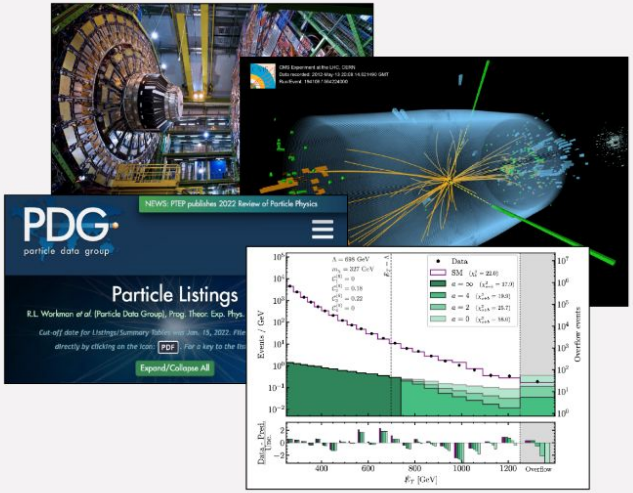
C++

```cpp
// User-side prior transform function, which can be called by GAMBIT-light.
void user_prior(const std::vector<std::string>& input_names,
                const std::vector<double>& input_vals,
                std::vector<double>& output)
{
    for (size_t i = 0; i < input_vals.size(); i++)
    {
        output[i] = input_vals[i] * 10.;
    }
}

GAMBIT_LIGHT_REGISTER_PRIOR(user_prior)
```

Also: user-supplied prior transformation

```yaml
UserModel:

  p1:
    name: param_name_1
  p2:
    name: param_name_2
  p3:
    name: param_name_3
  p4:
    name: param_name_4
  p5-p7:
    name: param_name_

UserPrior:

  lang: python
  user_lib: gambit_light_interface/example_python/example_prior_transform.py
  func_name: user_prior

UserLogLikes:

  py_user_loglike:
    lang: python
    user_lib: gambit_light_interface/example_python/example.py
    func_name: user_loglike
    output:
      - py_user_loglike_output_1
      - py_user_loglike_output_2
      - py_user_loglike_output_3

  cpp_user_loglike:
    lang: c++
    user_lib: gambit_light_interface/example_cpp/example.so
    func_name: user_loglike
    input:
      - param_name_2
      - param_name_3
    output:
      - cpp_user_loglike_output_1
      - cpp_user_loglike_output_2
      - cpp_user_loglike_output_3

```

4. Particles, pandemics and parameter spaces

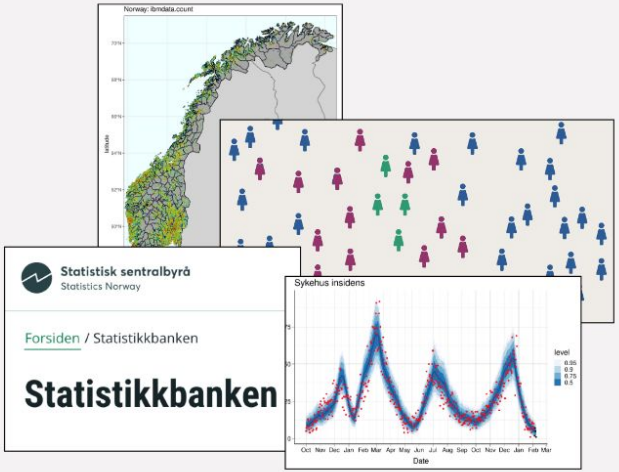


Modelling LHC physics

- Models with many free parameters
- Non-deterministic dynamics (quantum mechanics)
- Each prediction requires expensive Monte Carlo simulations
- *Task:* estimate model parameters, make robust predictions

Modelling outbreaks of infectious diseases

- Model with many free parameters
- Non-deterministic dynamics (people)
- Each prediction requires expensive Monte Carlo simulations
- *Task:* estimate model parameters, make robust predictions



- **Collaboration with FHI** (Norwegian Institute of Public Health)

- *Overall goal:*
  Tackle computational challenges common to both particle physics and disease modelling

- *Starting point:*
  Connect **FHI's individual-based model** to **GAMBIT-light**
  → model optimisation and uncertainty estimation

- *Some possible next steps:*
  - Improved multi-level parallelisation schemes (GPUs?)
  - Develop + use fast ML-based surrogate models (e.g. with *continual learning*)
  - Tailored parameter sampling algorithms
  - …

- **Ida-Marie** (UiO + FHI)

- Are (UiO)
- Me (UiO)

- Jørgen Eriksson Midtbø (FHI)
- Birgitte Freiesleben de Blasio (FHI)
- Francesco Di Ruscio (FHI)
- Yat Hin Chan (FHI)