Highly Parallel Amplitude Analysis with AmpTools

Software and Computing Round Table
July 12, 2022

Matthew Shepherd
Indiana University

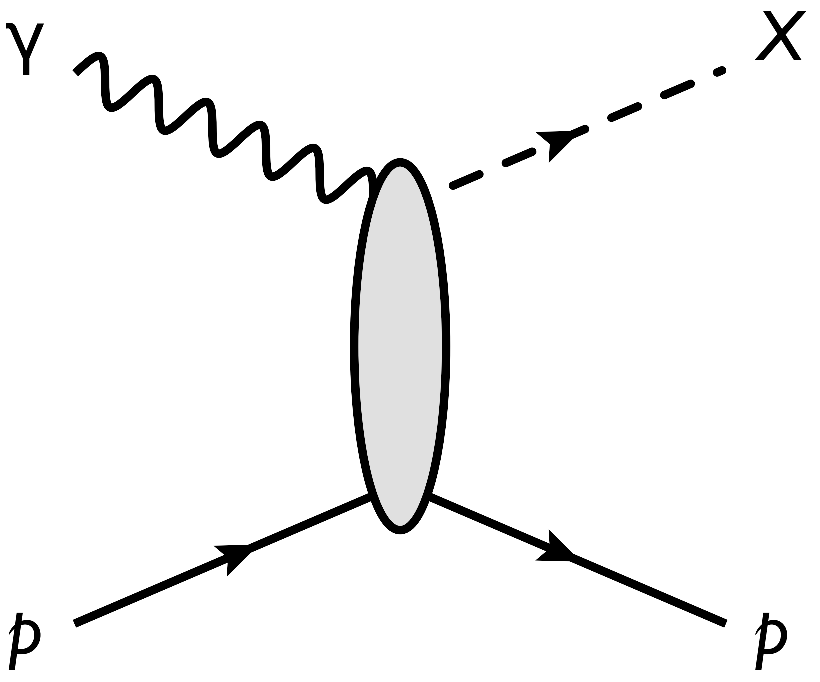
Outline

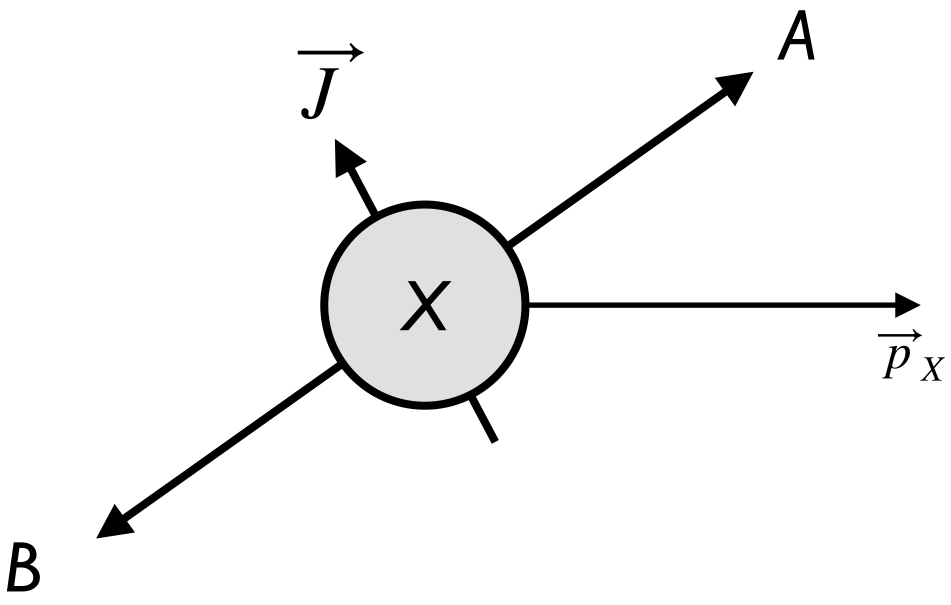
- Amplitudes
  - what are we trying to analyze?
- Method of Maximum Likelihood
  - what is the analysis strategy?
- Parallel Analysis with AmpTools
  - how do we execute the analysis efficiently?

Amplitudes

Amplitude Ideas

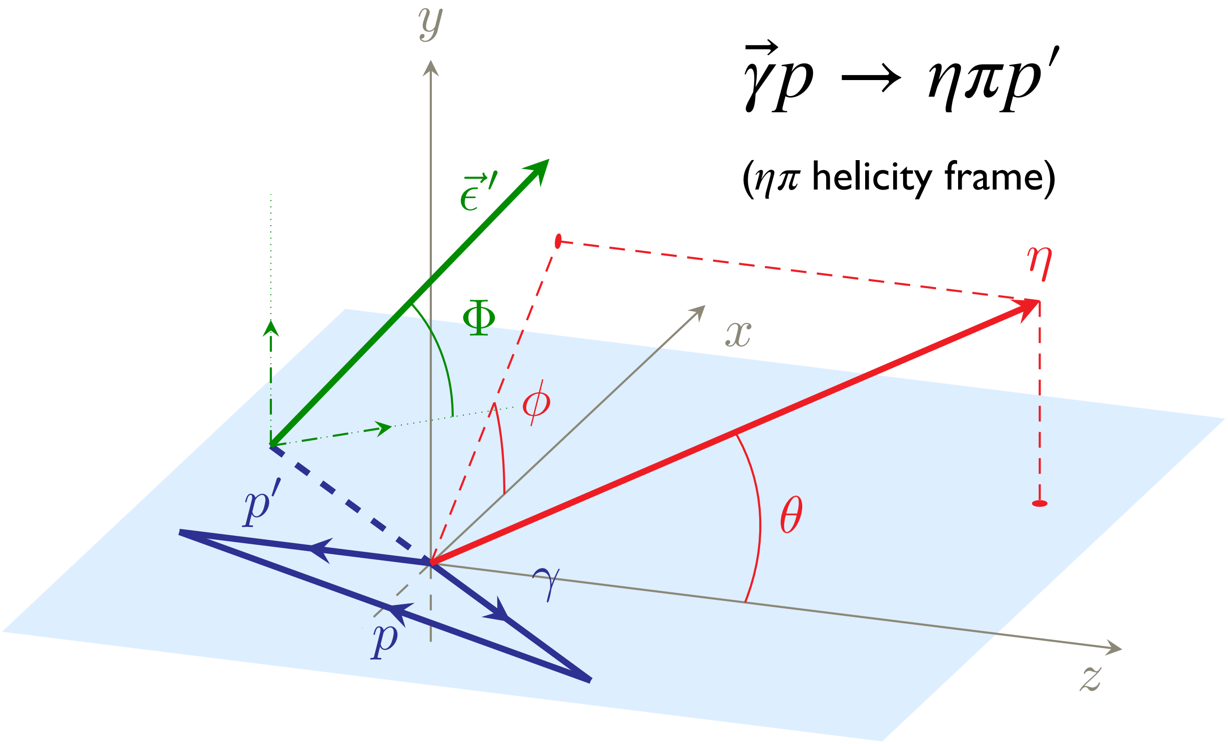
- Quantum mechanics
  - amplitude: complex valued function of particle kinematics
  - indistinguishable amplitudes interfere (add coherently)
  - sum over distinguishable initial and final states (add incoherently)
- Amplitude structure, examples
  - kinematics: Y_ℓ^m for conservation of angular momentum
  - dynamics: Breit-Wigner function to describe lineshape of resonance
- What do we want to learn by fitting to data?
  - magnitude (and phase) of certain amplitudes
  - properties of resonances





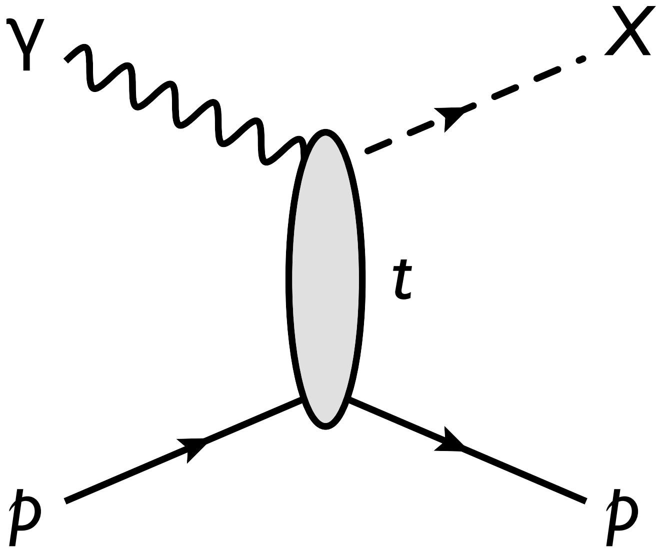
$\eta\pi$ Polarized Photoproduction

- Example: GlueX polarized photon beam allows one to study meson production mechanisms
- GlueX kinematics: distribution reaction plane with respect to photon polarization plane determines properties of exchange
- Ultimate goal:
  - study the properties of X (spin, parity, mass, ...)
  - study the production mechanism of X (interaction with the target)

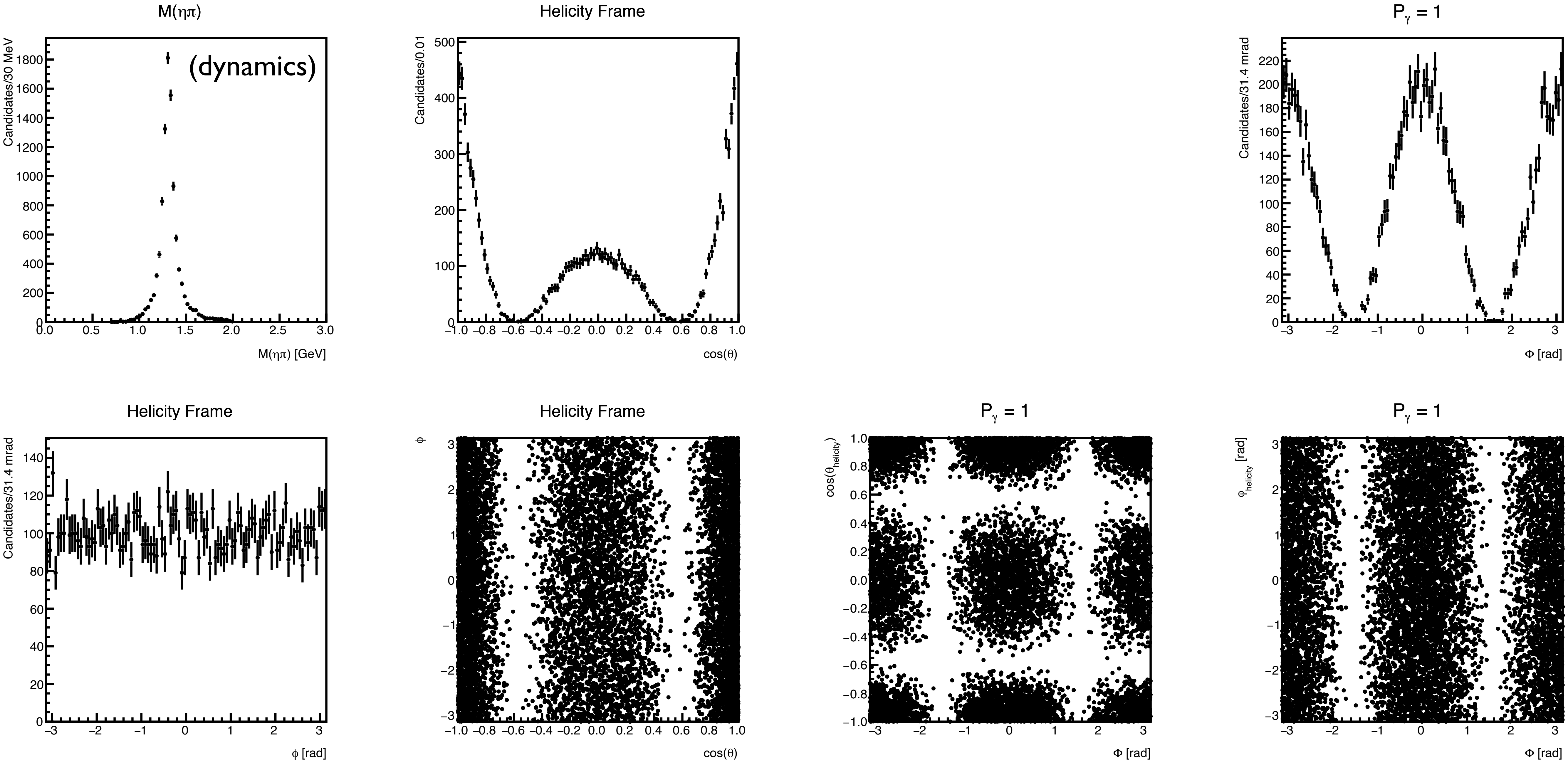


$\vec{\gamma}p \to \eta\pi p'$

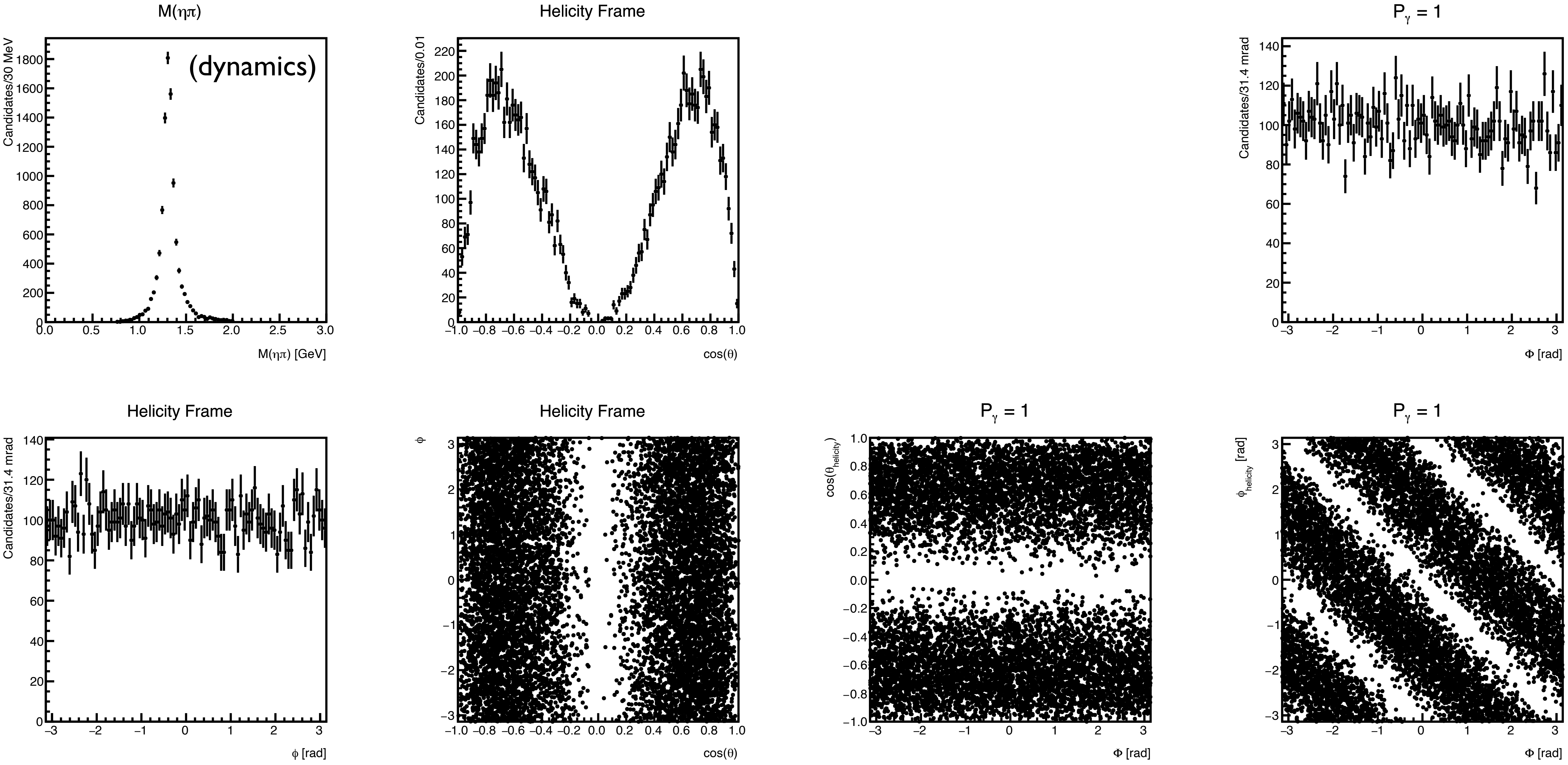
($\eta\pi$ helicity frame)



$a_2(1320) \to \eta\pi$ in the $\ell_m^\epsilon = D_0^+$ amplitude



$a_2(1320) \to \eta\pi$ in the $\ell_m^{\epsilon} = D_1^{-}$ amplitude



The Method of Maximum Likelihood for Parameter Estimation

Maximum Likelihood

- Amplitude analysis uses the method of maximum likelihood for parameter estimation
- Model the intensity in $\vec{x}$ with parameters $\vec{\theta}$

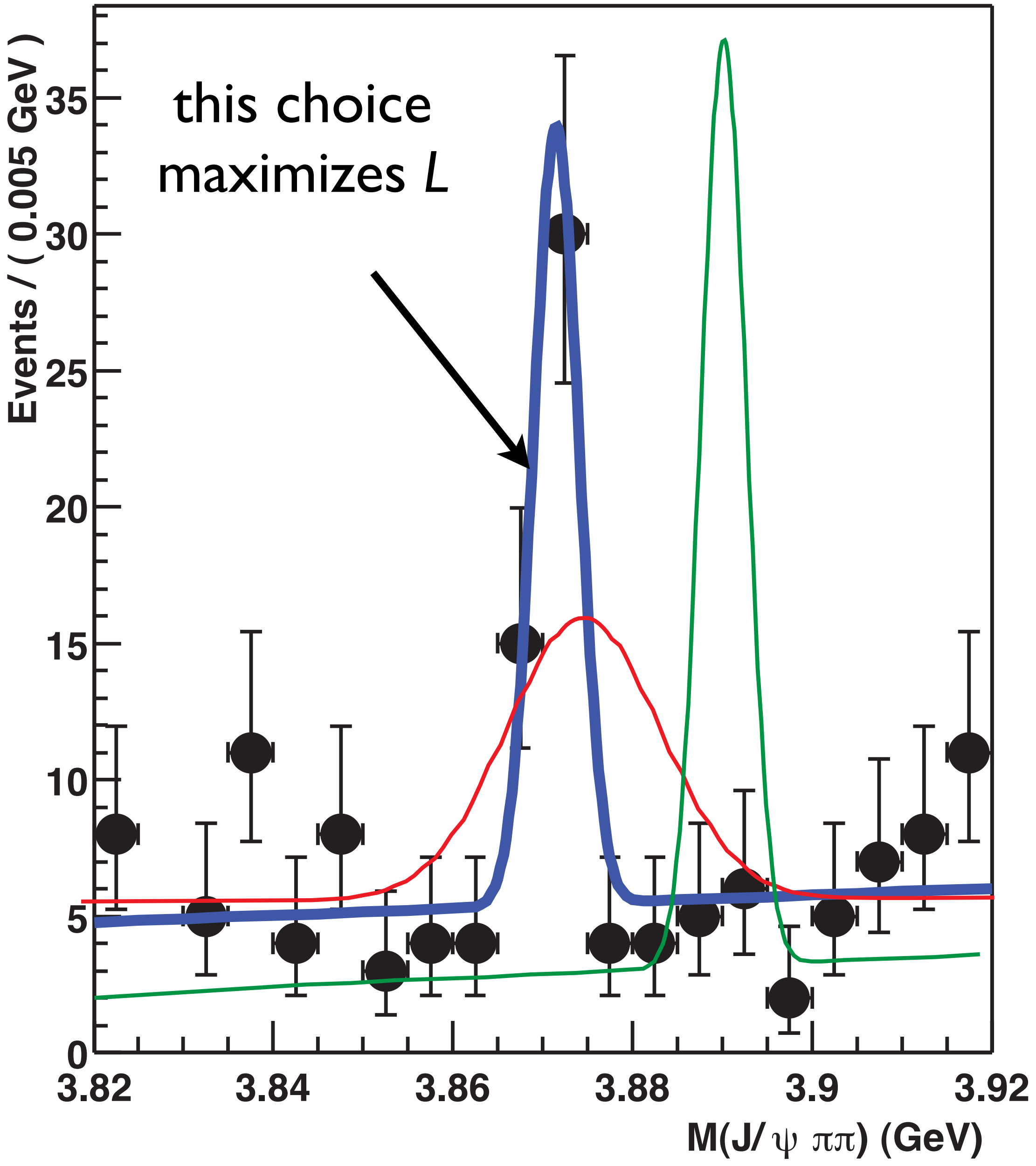
$$\mathcal{P}(\vec{x};\vec{\theta})$$

- Vary the free parameters to maximize the probability of observing one's data set

$$\mathcal{L} = \prod_{i=1}^{N_{\text{events}}} \mathcal{P}(\vec{x}_i;\vec{\theta})$$

- No computation of χ^2: no "binning"

Example: 1D in *x*

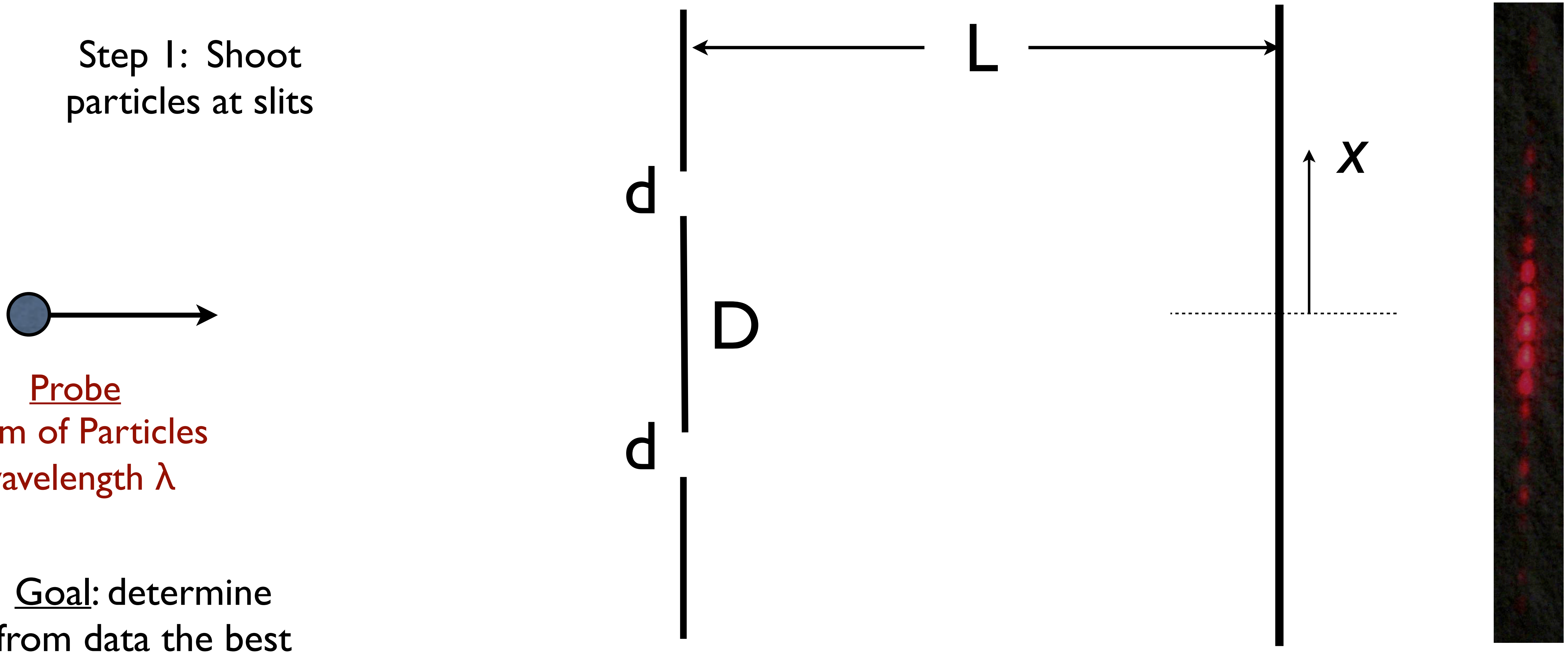


this choice maximizes *L*

Experiment Application

Step 2: For each particle record location *x* where it was detected

Step 1: Shoot particles at slits



Probe
Beam of Particles
wavelength λ

Goal: determine from data the best estimates for *d* and *D*

Physical System Under Study
Two Slits: width d, separation D

Detector
Measures location x_i for each arriving particle

The Fit Procedure

- Construct model

$$I(x) = I_0 \left(\frac{\sin(d\pi x/\lambda L)}{d\pi x/\lambda L} \right)^2 \cos^2(2D\pi x/\lambda L)$$

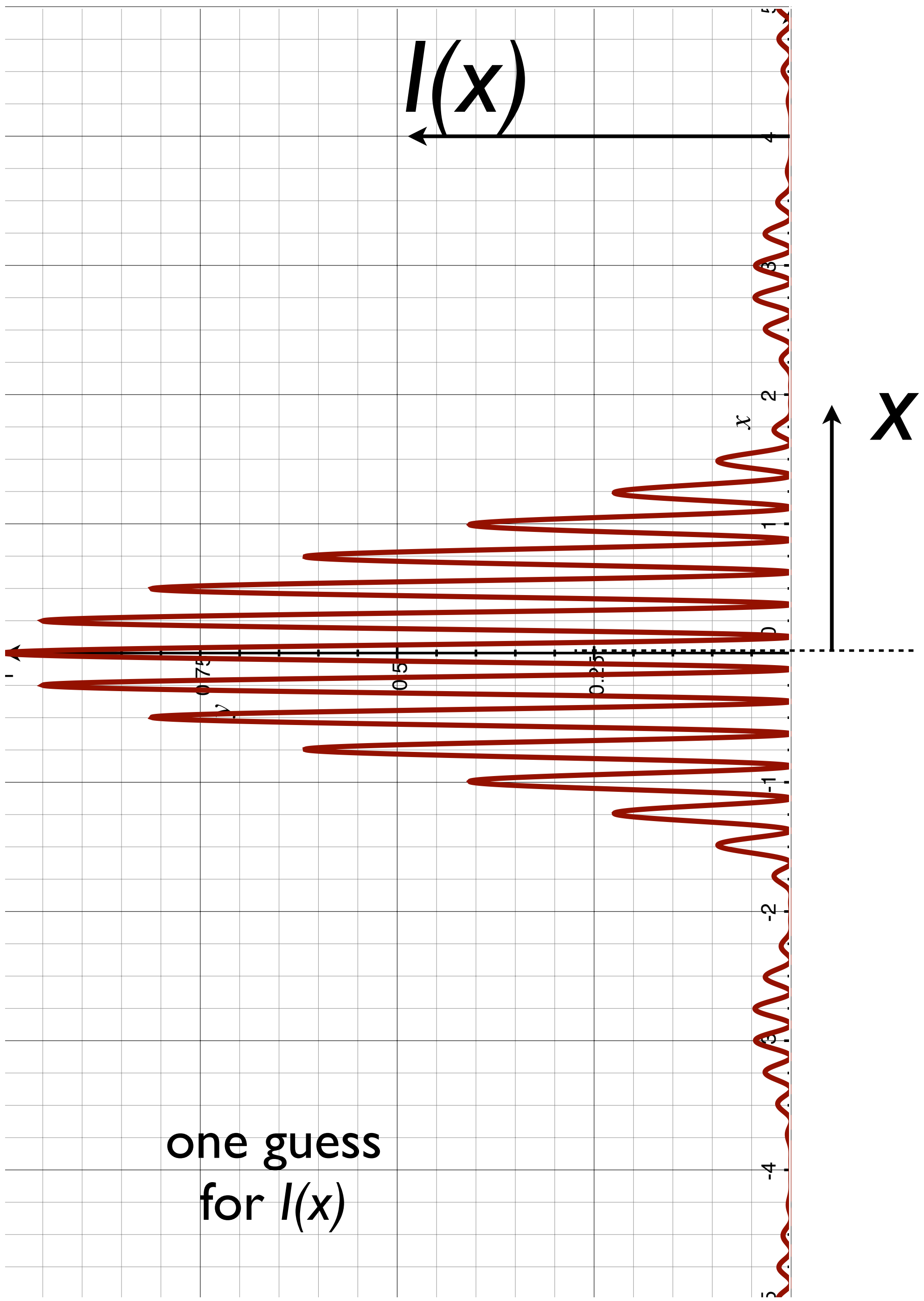
- Guess parameters *d* and *D*
- Construct PDF:

$$\mathcal{P}(x) = \frac{I(x)}{\int_{x_{\min}}^{x_{\max}} I(x)dx}$$

- Compute likelihood

$$\mathcal{L} = \prod_{i=1}^{N} \mathcal{P}(x_i)$$

- Iterate to maximize $\mathscr{L}$ or minimize $-2\ln\mathscr{L}$ (AmpTools uses MINUIT for this)



one guess for *I*(*x*)

Putting It Together

- Construct the likelihood for a data set of *N* observed events -- minimize $-2\ln \mathscr{L}$

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{e^{-\mu}\mu^N}{N!}\prod_{i=1}^{N}\mathcal{P}(\mathbf{x}_i;\boldsymbol{\theta})$$

$$\mathcal{P}(\mathbf{x};\boldsymbol{\theta}) = \frac{1}{\mu}\mathcal{I}(\mathbf{x};\boldsymbol{\theta})\eta(\mathbf{x})$$

$$\mu = \int \mathcal{I}(\mathbf{x};\boldsymbol{\theta})\eta(\mathbf{x})\ d\mathbf{x}$$

- AmpTools provides a framework to construct the model for the intensity under some assumptions and manage issues like the detector and analysis acceptance $\eta(\mathbf{x})$

$$\mathcal{I}(\mathbf{x}) = \sum_{\sigma}\left|\sum_{\alpha} s_{\sigma,\alpha}V_{\sigma,\alpha}A_{\sigma,\alpha}(\mathbf{x})\right|^2$$

$$A_{\sigma,\alpha}(\mathbf{x}) = \prod_{\gamma=1}^{n_{\sigma,\alpha}} a_{\sigma,\alpha,\gamma}(\mathbf{x})$$

α: indistinguishable amplitudes; σ: distinguishable coherent sums; γ: amplitude factors

Parallelization for Practical Problems

- To properly normalize the p.d.f. one needs

$$\int \mathcal{I}(\mathbf{x};\boldsymbol{\theta})\eta(\mathbf{x})\ d\mathbf{x}$$

- This must be numerically computed using a large set of phase space MC (no physics model) and subjected to detector + analysis requirements
    - for log likelihood minimization the integral can be replaced with the average value of the integrand

$$\int_R f(x)\ dx = R\langle f(x)\rangle \qquad \langle \mathcal{I}(\mathbf{x};\boldsymbol{\theta})\eta(\mathbf{x})\rangle = \frac{1}{M_g}\sum_{i=1}^{M_a}\mathcal{I}(\mathbf{x}_i;\boldsymbol{\theta})$$

insert RHS term here

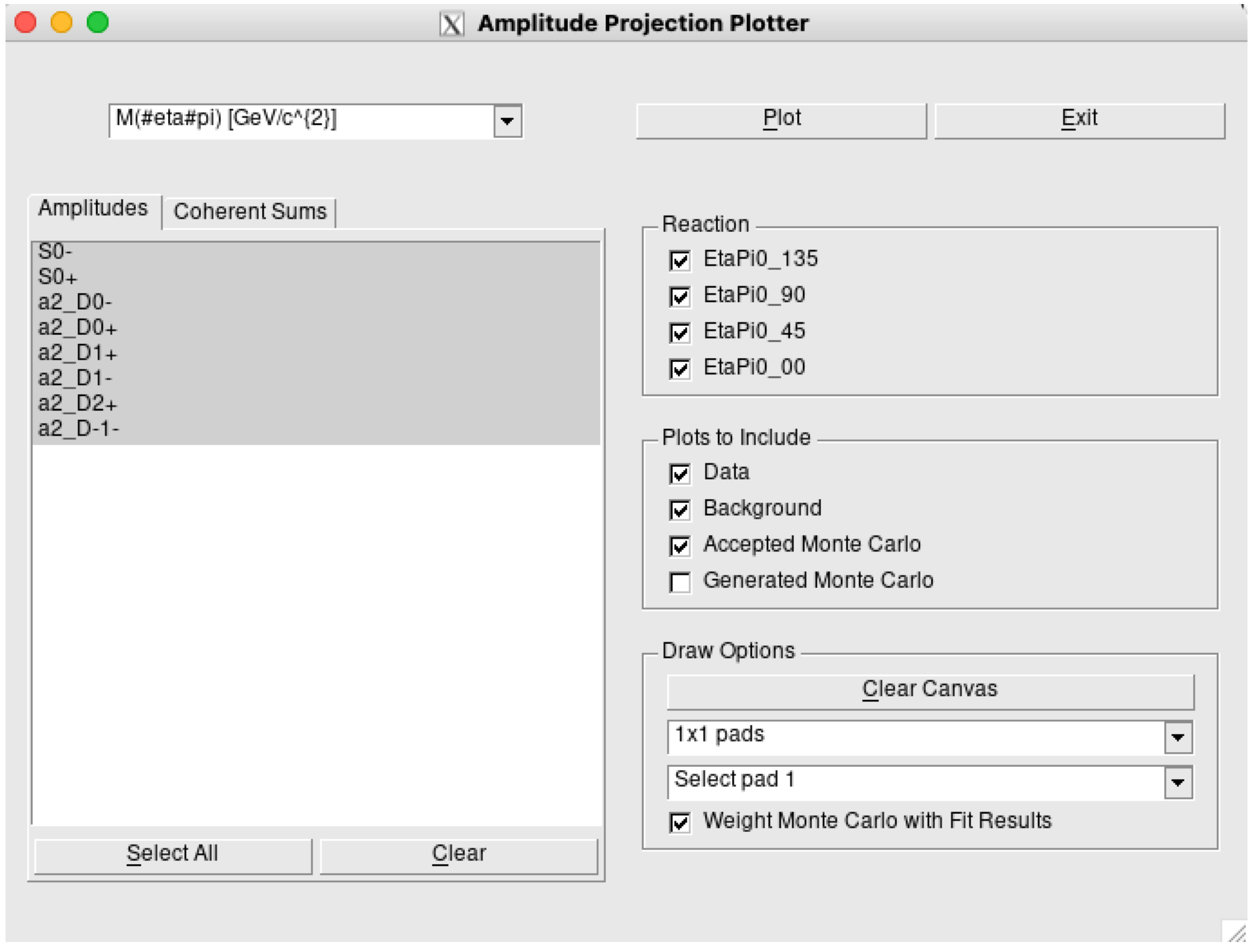
$$-2\ln\mathcal{L}(\boldsymbol{\theta}) = -2\left(\sum_{i=1}^{N}\ln\mathcal{I}(\mathbf{x}_i;\boldsymbol{\theta}) - \int \mathcal{I}(\mathbf{x};\boldsymbol{\theta})\eta(\mathbf{x})\ d\mathbf{x}\right) + c_1$$

sum over data

sum over accepted MC

Need: large data and accepted MC set in RAM and the ability to compute sums over all events

Goal: A Good Fit to Data



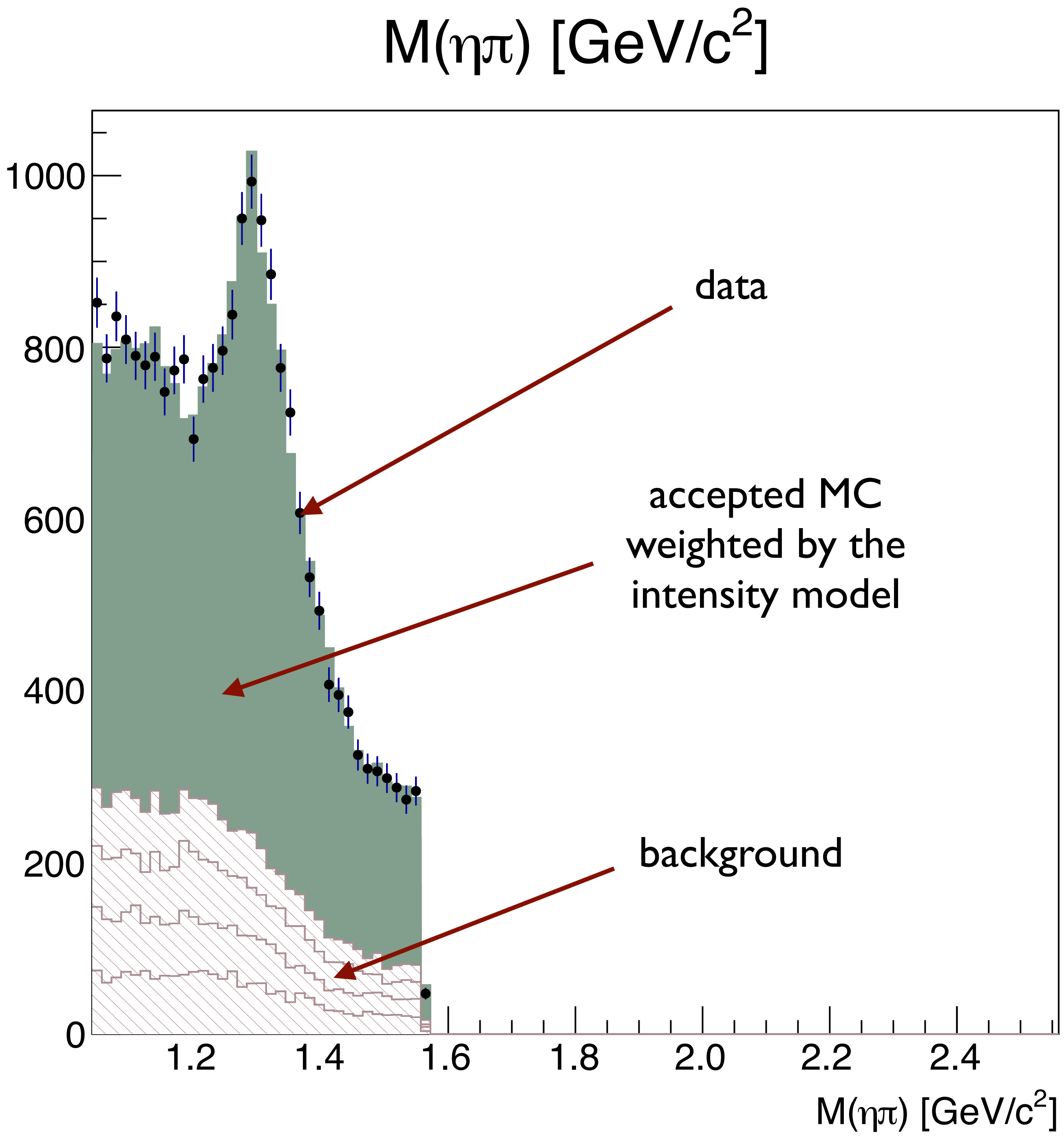
M(#eta#pi) [GeV/c^{2}]

S0-
S0+
a2_D0-
a2_D0+
a2_D1+
a2_D1-
a2_D2+
a2_D-1-

Reaction
- ☑ EtaPi0_135
- ☑ EtaPi0_90
- ☑ EtaPi0_45
- ☑ EtaPi0_00

Plots to Include
- ☑ Data
- ☑ Background
- ☑ Accepted Monte Carlo
- ☐ Generated Monte Carlo

Draw Options
- 1x1 pads
- Select pad 1
- ☑ Weight Monte Carlo with Fit Results



data

accepted MC weighted by the intensity model

background

Parallel Analysis with AmpTools

AmpTools Design Goals

- Separate physics from computing
- The "user" provides:
  - an algorithm to unpack four-vectors from a file
  - algorithms to compute various physics amplitudes from four-vectors
  - a recipe for assembling the amplitudes into an intensity
- AmpTools provides:
  - a general framework that makes no assumptions about experiment or physics model (other than quantum mechanics)
  - a set of core libraries optimized for unbinned likelihood fitting and parallel processing
    - MPI parallelization was always a part of design: knew eventual problem size would exceed RAM on one machine
    - GPU acceleration per process (multiple GPUs supported through MPI)
  - modular code that can also be used for MC generation and displaying fit results

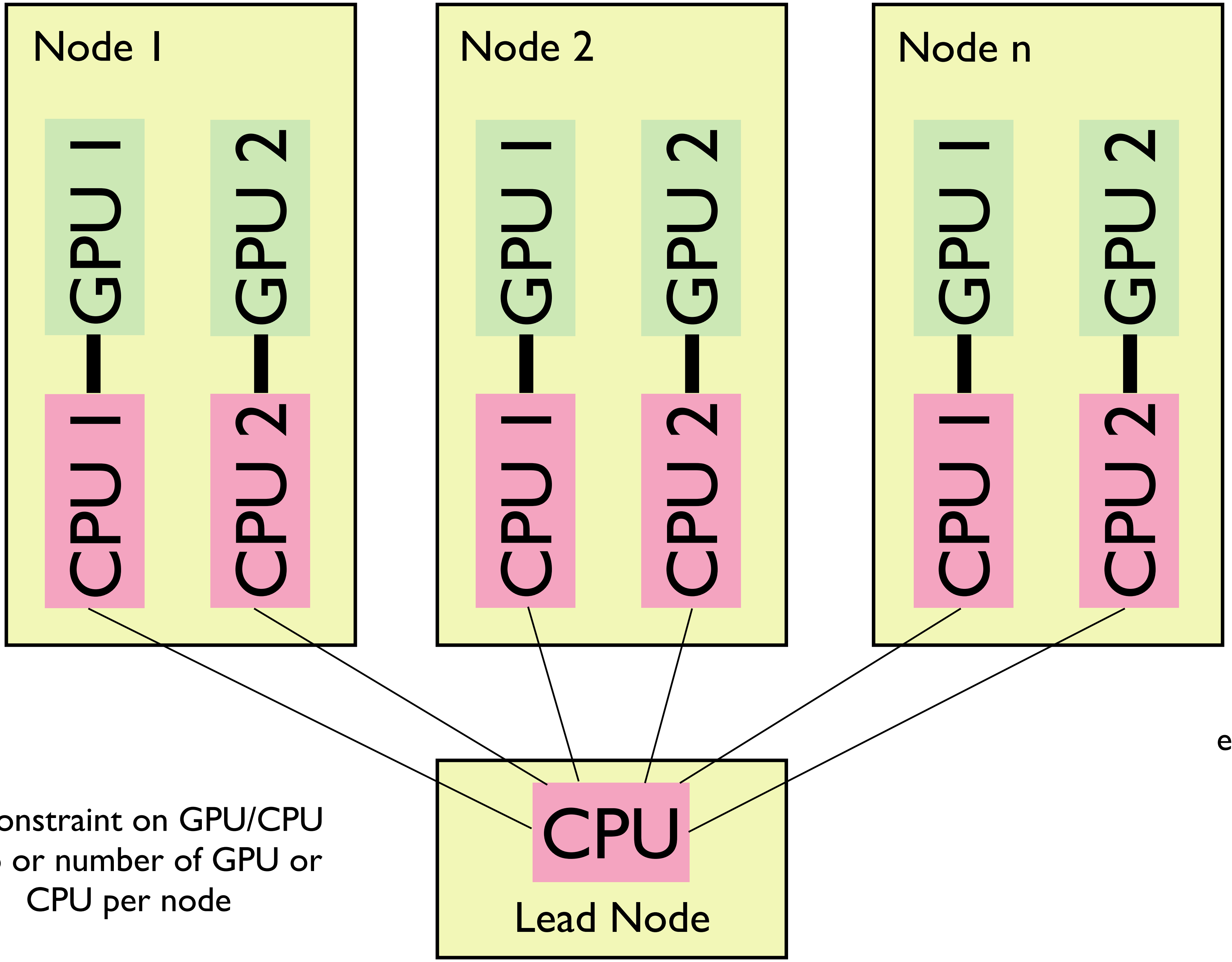
What Drives Fit Speed

$$-2\ln \mathcal{L}(\boldsymbol{\theta}) = -2\left(\sum_{i=1}^{N} \ln \mathcal{I}(\mathbf{x}_i;\boldsymbol{\theta}) - \int \mathcal{I}(\mathbf{x};\boldsymbol{\theta})\eta(\mathbf{x})\ d\mathbf{x}\right) + c_1$$

sum over data sum over accepted MC

- Typical fit may need $\mathcal{O}(100) - \mathcal{O}(100{,}000)$ computations of $-2\ln\mathscr{L}$ and a "large" data set may have millions of data and MC events
  - cost of intensity calculation grows like $N^2_{\text{amplitudes}}$
- Fit speed is dominated by two things:
  - speed of computing $-2\ln\mathscr{L}$
  - reducing the number of computations of $-2\ln\mathscr{L}$ by choice of algorithm used find the minimum, convergence criteria, etc.
- Large, independent sums lend themselves well to parallel processing
  - partial sums over partial data sets computed on individual processes
  - GPUs enable event-level parallelization for amplitude computations and other sums

Generic Fitting Topology



Node 1

GPU 1

GPU 2

CPU 1

CPU 2

Node 2

GPU 1

GPU 2

CPU 1

CPU 2

Node n

GPU 1

GPU 2

CPU 1

CPU 2

CPU

Lead Node

Optional GPU acceleration

Finely parallel: typically one CUDA thread per event or amplitude

Coarsely parallel: one MPI process per GPU

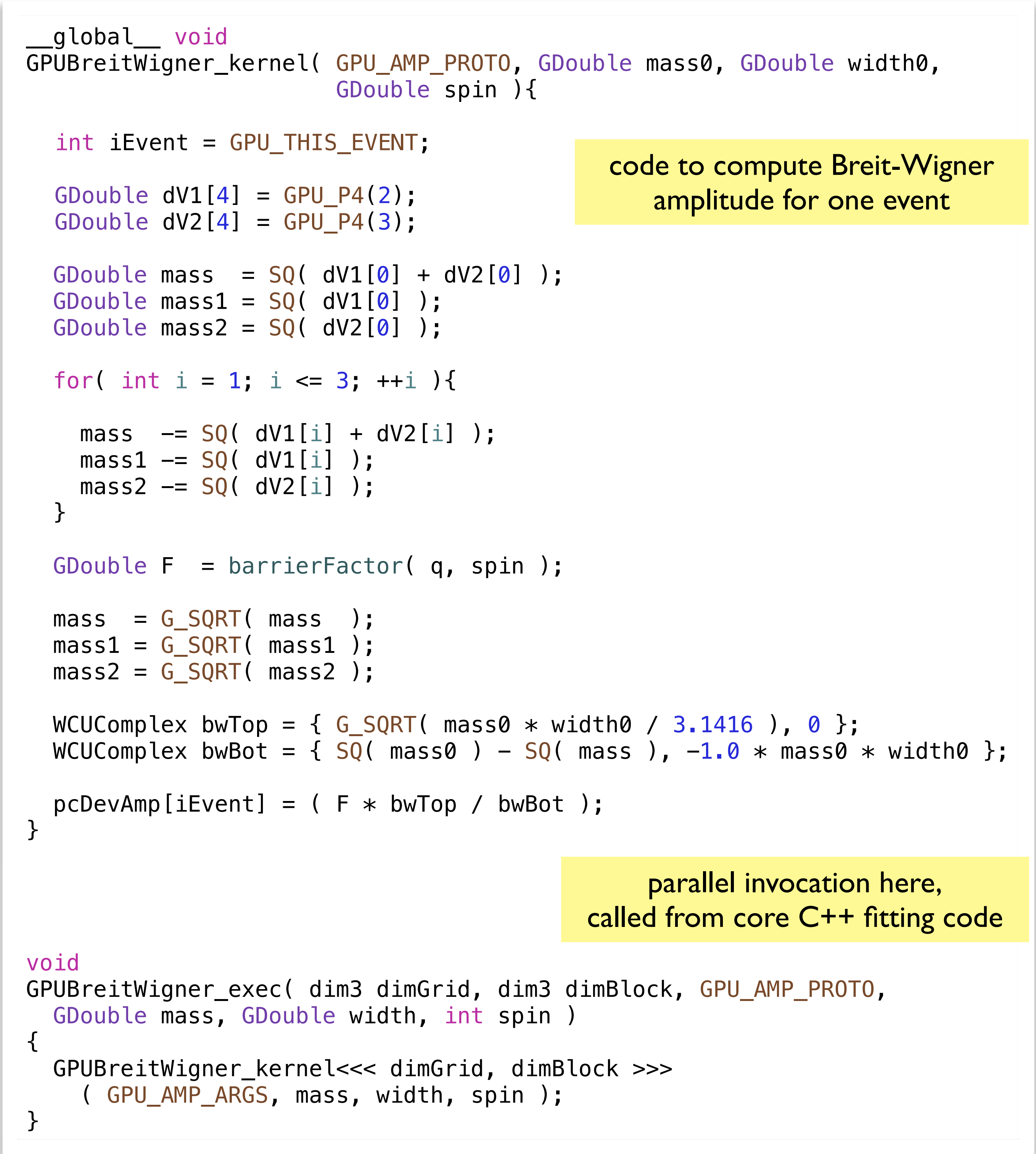
Parameters and partial sums exchanged over network layer

Minimizing algorithm runs here

no constraint on GPU/CPU ratio or number of GPU or CPU per node

Accelerating Code through Parallelization

- Design challenge: make user-provided code run fast
    - minimize calls to user-written functions
    - add functionality like caching and examples for how to use it
- MPI: very little custom user code needed
    - coarse: one process per core
    - prefer MPI (multi-node) over multithread
- GPU: compute intensive amplitudes require user-provided CUDA kernel to get maximum performance in some cases



```
__global__ void
GPUBreitWigner_kernel( GPU_AMP_PROTO, GDouble mass0, GDouble width0,
                       GDouble spin ){

  int iEvent = GPU_THIS_EVENT;

  GDouble dV1[4] = GPU_P4(2);
  GDouble dV2[4] = GPU_P4(3);

  GDouble mass  = SQ( dV1[0] + dV2[0] );
  GDouble mass1 = SQ( dV1[0] );
  GDouble mass2 = SQ( dV2[0] );

  for( int i = 1; i <= 3; ++i ){

    mass  -= SQ( dV1[i] + dV2[i] );
    mass1 -= SQ( dV1[i] );
    mass2 -= SQ( dV2[i] );
  }

  GDouble F  = barrierFactor( q, spin );

  mass  = G_SQRT( mass  );
  mass1 = G_SQRT( mass1 );
  mass2 = G_SQRT( mass2 );

  WCUComplex bwTop = { G_SQRT( mass0 * width0 / 3.1416 ), 0 };
  WCUComplex bwBot = { SQ( mass0 ) - SQ( mass ), -1.0 * mass0 * width0 };

  pcDevAmp[iEvent] = ( F * bwTop / bwBot );
}
```

code to compute Breit-Wigner amplitude for one event

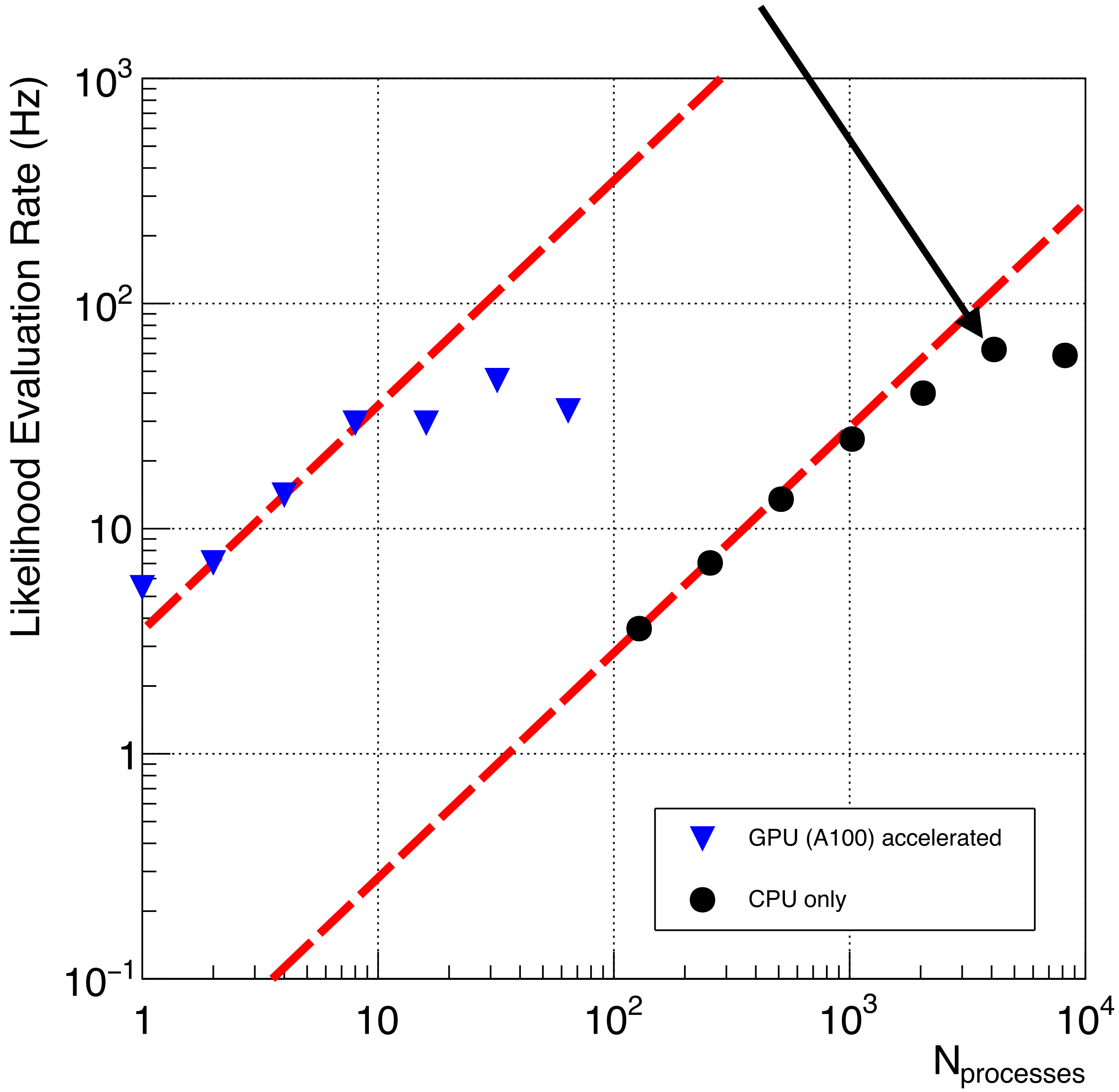
parallel invocation here, called from core C++ fitting code

```
void
GPUBreitWigner_exec( dim3 dimGrid, dim3 dimBlock, GPU_AMP_PROTO,
  GDouble mass, GDouble width, int spin )
{
  GPUBreitWigner_kernel<<< dimGrid, dimBlock >>>
    ( GPU_AMP_ARGS, mass, width, spin );
}
```

compiled with NVIDIA's compiler: nvcc,
linked into standard C/C++ code

Performance and Scaling

- Benchmark: $J/\psi \to \phi K K$ (from BESIII)
  - 1.3M data events and 2.4M MC events
  - about 50 amplitudes and 110 free parameters
  - ~100K function calls to convergence
    - 40 days (!) with a single core
- Test platform: Indiana U. BigRed 200 (HPE Cray Shasta)
  - 640 nodes: 2 x 64 core AMD
  - 64 nodes: 4 x NVIDIA A100



4096 processes
communicating simultaneously
across 32 nodes

Thanks to Nils Hüsken for these data

Performance Comments

- Likelihood calculation on large data sets lends itself well to parallelization
    - MPI solutions are transparent to the user and exhibit excellent scaling up to thousands of cores (with appropriate hardware)
    - A single GPU with enough RAM typically provides at least 100x speed gain
- GPU notes
    - usually limited by memory bandwidth
    - hardware with large amounts of GPU RAM is preferred
    - strong preference to do all computations in double precision
    - some non-trival user development is required
- For a flexible framework, one size fits all optimization is challenging -- guidance to users is needed
    - recent GlueX fit: factor of 4 speedup in better memory use and caching complicated angle calculations which enabled a factor of 100 going to GPU

History, Acknowledgements, and More Information

- AmpTools was part of 2007 NSF ""Physics at the Information Frontier" award
  - initial development in collaboration with Ryan Mitchell at Indiana U.
  - initial NVIDIA acceleration implemented in 2010 by Hrayr Matevosyan
- 2011: first public release of package v0.1 corresponding with first publication that used AmpTools "Amplitude analysis of the decays $\chi_{c1} \to \eta\pi^+\pi^-$ and $\chi_{c1} \to \eta'\pi^+\pi^-$," by the CLEO-c Collaboration
- Thanks to the Indiana University High Performance Computing group for tools and guidance to optimize parallel computing
- BigRed200 is maintained by Indiana University Research Technologies
- AmpTools source code is here: *https://github.com/mashephe/AmpTools*
  - the "Dalitz" tutorial distributed with the code is fully functional: it will generate and fit pseudodata and can do so in parallel and on a GPU
- Have additional questions or need more information? *mashephe@indiana.edu*