# Efficient Histogramming for High-Performance Computing in C++ with YODA


**Christian Gütschow**

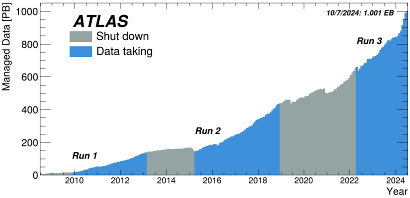

**FOSDEM 2025, Brussels**

**02 February 2025**

# The Data Challenge in Particle Physics

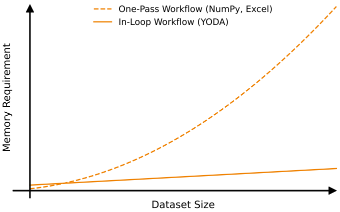
- The Large Hadron Collider (LHC) generates petabytes of data annually from **billions of collision events**.
- Each event records the properties of numerous particles, creating **complex, high-dimensional datasets**.
- To interpret these events, we rely heavily on Monte Carlo (MC) simulations to compare with theoretical models.
- The scale of both real and simulated data presents a **major challenge** for efficient processing and analysis.


[CERN]



# HPC and Data Workflow Challenges

- Traditional histogramming workflows process data after event generation, often in Python.
- For large datasets, this approach hits limits in memory usage and I/O bandwidth.
- We need fast, in-loop analysis tools that summarise statistics during event processing.
- Solution: updatable summary statistics directly in C++ to handle massive bulk samples efficiently.



# Enter YODA

- Yet more Objects for Data Analysis! [yoda.hepforge.org]
- Designed for memory efficiency and speed in high-performance environments.
- First released in 2013, second major version available as of 2023. [gitlab.com/hepcedar/yoda]
- Written in C++ and programmatically usable from C++ and Python, complemented by a set of command-line tools for dataset inspection, manipulation and combination.

**Consistent, multidimensional differential histogramming and summary statistics with YODA 2**

Andy Buckley, Louie Corpe, Matthew Filipovich, Christian Gutschow, Nick Rozinsky, Simon Thor, Yoran Yeh, Jamie Yellen

[arXiv:2312.15070]

- Emerged from the sub-field of MC event generator analysis in particle physics, but library is deliberately agnostic of any particular application

# Design principles I

- Accurate Distribution Estimation
  - Histograms represent best-estimate distributions, not just simple fill counts.
  - Non-uniform binning is essential for optimal data estimation in complex datasets.

# Design principles I

- Accurate Distribution Estimation
  - Histograms represent best-estimate distributions, not just simple fill counts.
  - Non-uniform binning is essential for optimal data estimation in complex datasets.
- Live, Updatable Data Objects
  - Histograms must support continuous updates as events are processed.
  - Unlike tools like `NumPy` or `Excel`, HEP workflows often can't load all data into memory at once.

# Design principles I

- Accurate Distribution Estimation
  - Histograms represent best-estimate distributions, not just simple fill counts.
  - Non-uniform binning is essential for optimal data estimation in complex datasets.
- Live, Updatable Data Objects
  - Histograms must support continuous updates as events are processed.
  - Unlike tools like `NumPy` or `Excel`, HEP workflows often can't load all data into memory at once.
- Weighted Statistics for Precision
  - Bins track weighted statistical moments for key data summaries.
  - Unbinned quantities can be tracked alongside binned data to capture more detailed trends for analysis.

# Design principles I

- Accurate Distribution Estimation
  - Histograms represent best-estimate distributions, not just simple fill counts.
  - Non-uniform binning is essential for optimal data estimation in complex datasets.
- Live, Updatable Data Objects
  - Histograms must support continuous updates as events are processed.
  - Unlike tools like `NumPy` or `Excel`, HEP workflows often can't load all data into memory at once.
- Weighted Statistics for Precision
  - Bins track weighted statistical moments for key data summaries.
  - Unbinned quantities can be tracked alongside binned data to capture more detailed trends for analysis.
- Consistent Projections Across Dimensions
  - Maintain integral consistency when reducing higher-dimensional histograms to lower dimensions.
  - Ensure unbiased trend analysis by exact marginalisation of multidimensional data.

# Design principles II

- Style Independence
  - Statistical data remains consistent, regardless of changes to plotting style or presentation.

# Design principles II

- Style Independence
  - Statistical data remains consistent, regardless of changes to plotting style or presentation.
- Decoupling Binning and Bin Content
  - Supports both *live* objects for ongoing data updates and *inert* representations for finalised data summaries (values and uncertainties).

# Design principles II

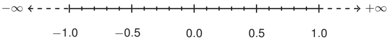
- Style Independence
  - Statistical data remains consistent, regardless of changes to plotting style or presentation.
- Decoupling Binning and Bin Content
  - Supports both *live* objects for ongoing data updates and *inert* representations for finalised data summaries (values and uncertainties).
- User-Friendly Interface
  - Clean API designed for data scientists, focusing on familiar statistical and data-analytic concepts.
  - Internal complexity is abstracted away to maintain statistical consistency and type safety.
  - Focused on binned statistical analysis, with zero external dependencies for seamless embedding in core C++ applications.

# Flexible Bin Partitioning for Modern Analysis Needs

- New flexible `Axis` class supports both continuous and discrete data types.
  - Template-based design adapts to different edge types automatically.

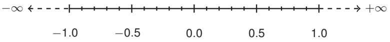
# Flexible Bin Partitioning for Modern Analysis Needs

- New flexible `Axis` class supports both continuous and discrete data types.
  - Template-based design adapts to different edge types automatically.
- Continuous Axis (classic mode)
  - Triggered by floating-point types using standard type traits.
  - Binning defined by $N + 1$ edges for $N$ bins, plus under-/overflow bins.
  - Bin widths handle infinite ranges where needed.

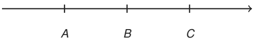


# Flexible Bin Partitioning for Modern Analysis Needs

- New flexible `Axis` class supports both continuous and discrete data types.
  - Template-based design adapts to different edge types automatically.
- Continuous Axis (classic mode)
  - Triggered by floating-point types using standard type traits.
  - Binning defined by $N + 1$ edges for $N$ bins, plus under-/overflow bins.
  - Bin widths handle infinite ranges where needed.

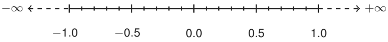



- Discrete Axis (new mode)
  - Designed for non-continuous types (e.g. integers, categories).
  - Bins defined by $N$ edges and a special “otherflow” bin for outliers.
  - Ideal for multiplicities, cutflows, and categorical data handling.

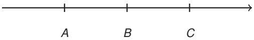


# Flexible Bin Partitioning for Modern Analysis Needs

- New flexible `Axis` class supports both continuous and discrete data types.
  - Template-based design adapts to different edge types automatically.
- Continuous Axis (classic mode)
  - Triggered by floating-point types using standard type traits.
  - Binning defined by $N + 1$ edges for $N$ bins, plus under-/overflow bins.
  - Bin widths handle infinite ranges where needed.



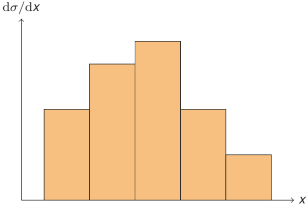

- Discrete Axis (new mode)
  - Designed for non-continuous types (e.g. integers, categories).
  - Bins defined by $N$ edges and a special “otherflow” bin for outliers.
  - Ideal for multiplicities, cutflows, and categorical data handling.




- Advanced `Binning` Features
  - Seamlessly translates between local bin indices and global index positions.
  - Supports slicing and marginalisation across multi-dimensional spaces.

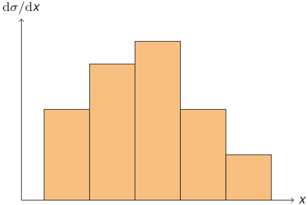
# Flexible Bin Content Types for Advanced Data Handling

- Live Content (`Dbn` Class)
  - Generalised multi-dimensional version of YODA1's distribution class.
  - Tracks exact first- and second-order statistical moments, including mixed moments.
  - Flexible `fill()` method: accepts coordinates, weights, and fill fractions for dynamic updates.

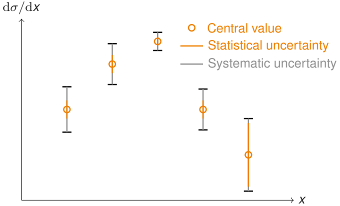


# Flexible Bin Content Types for Advanced Data Handling

- Live Content (`Dbn` Class)
  - Generalised multi-dimensional version of YODA1's distribution class.
  - Tracks exact first- and second-order statistical moments, including mixed moments.
  - Flexible `fill()` method: accepts coordinates, weights, and fill fractions for dynamic updates.

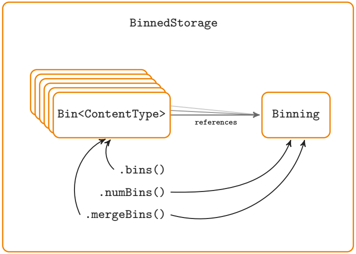



- Inert Content (`Estimate` Class)
  - Central value representation, optionally with detailed error breakdowns.
  - Encodes uncertainties as labeled {down, up} variations to capture dependence on theoretical or experimental parameters.
  - Supports both correlated and uncorrelated treatments of errors.
  - Arithmetic operations respect these uncertainty relationships for robust statistical handling.



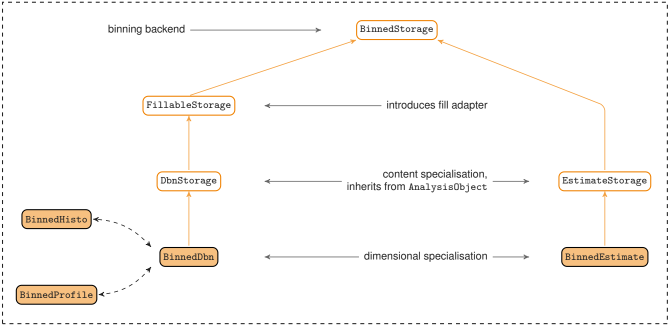
# Flexible Bin Management with BinnedStorage

- `Bin` wrapper class
  - Links bin content to both local and global bin properties.
  - Provides dimension-aware methods for volume calculations: `dVol()` for general volume, plus `dLen()`, `dArea()` aliases for 1D and 2D.
  - Templated accessors retrieve axis-specific properties seamlessly.
  - CRTP ensures intuitive method names for first 3 dimensions.




- `BinnedStorage` class
  - Holds arbitrary data types, enabling versatile content management.
  - Flexible bin lookups: Index-based (`bin(i)`) and coordinate-based (`binAt(x)`) retrieval.
  - Supports bin masking to emulate data “gaps” without requiring bin erasure: Mask bins by index (`mask(i)`) or coordinates (`maskAt(x)`).

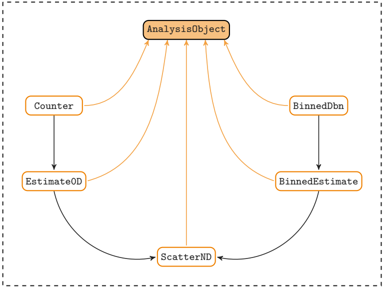
# FillableStorage: Managing Dynamic Bin Content




- Inherits from `BinnedStorage`, adding support for dynamic updates in "live" bin content.
- Introduces a fill adapter to manage bin-content updates for each fill operation.
- Ensures consistent handling of complex binning scenarios and statistical tracking.
- Fill function returns bin position as a global index or `-1` for invalid (`NaN`) coordinates.

# Type and Dimensionality Reductions for Flexible Data Handling

- Live to Inert Transformations
  - Live `BinnedDbn` objects reduce to inert `BinnedEstimate` objects.
  - 0-Dimensional Case: `Counter` (live) reduces to `Estimate0D` (inert).
  - Easily slice higher-dimensional data into lower-dimensional subsets along any axis.
- Scatter Objects for Visualisation
  - Both live and inert types reduce to `Scatter` objects for plotting and presentation.



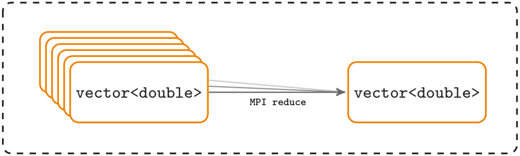

- Unified Metadata and Transformation Support
  - All user-facing types inherit from the `AnalysisObject` base class, enabling attribute storage for metadata.
  - Global scaling operations and arbitrary transformations (e.g. lambda functions) apply seamlessly to inert types like `Estimates` and `Scatters`.

# HPC Support for Distributed and Parallel Workflows

- Efficient Serialisation for MPI Communication:
  - `AnalysisObject` base class can be (de-)serialised into/from a `std::vector<double>`.
  - Facilitates easy communication of data across nodes in distributed environments like MPI.

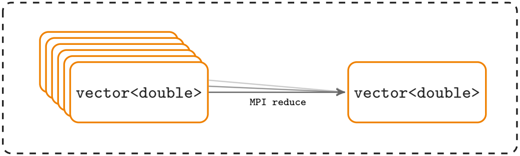
# HPC Support for Distributed and Parallel Workflows

- Efficient Serialisation for MPI Communication:
  - `AnalysisObject` base class can be (de-)serialised into/from a `std::vector<double>`.
  - Facilitates easy communication of data across nodes in distributed environments like MPI.
- On-the-Fly Stacking & Merging
  - Serialised data enables efficient stacking and merging of histograms during computation.
  - Supports parallel workflows where intermediate results can be combined dynamically across multiple processes.



# HPC Support for Distributed and Parallel Workflows

- Efficient Serialisation for MPI Communication:
  - `AnalysisObject` base class can be (de-)serialised into/from a `std::vector<double>`.
  - Facilitates easy communication of data across nodes in distributed environments like MPI.
- On-the-Fly Stacking & Merging
  - Serialised data enables efficient stacking and merging of histograms during computation.
  - Supports parallel workflows where intermediate results can be combined dynamically across multiple processes.




- Optimised for Scalability
  - Built to handle large datasets with **minimal memory overhead**, making it well-suited for HPC applications.
  - Seamless integration with parallel computing frameworks ensures **scalability** for big data analysis in particle physics.
  - Applications in Machine Learning: Serialised data can be **easily integrated** into machine learning pipelines for model training, feature extraction, and data preprocessing.

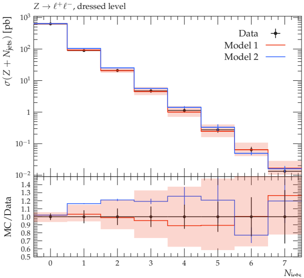
# Flexible I/O Formats for Analysis and HPC Applications

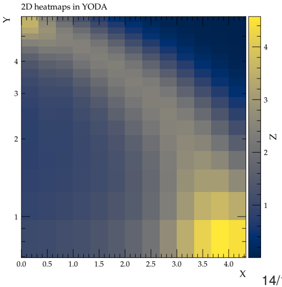
```
BEGIN YODA_HISTO1D_V3 /H1D_d
Path: /H1D_d
Title:
Type: Histo1D
---
# Mean: 3.470588e-01
# Integral: 1.700000e+01
Edges(A1): [0.000000e+00, 5.000000e-01, 1.000000e+00]
# sumW          sumW2           sumW(A1)        sumW2(A1)       numEntries
0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
1.000000e+01    1.000000e+02    1.000000e+00    1.000000e-01    1.000000e+00
7.000000e+00    4.900000e+01    4.900000e+00    3.430000e+00    1.000000e+00
0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
END YODA_HISTO1D_V3

BEGIN YODA_BINNEDHISTO<S>_V3 /H1D_s
Path: /H1D_s
Title:
Type: BinnedHisto<s>
---
# Mean: 3.750000e-01
# Integral: 8.000000e+00
Edges(A1): ["A"]
# sumW          sumW2           sumW(A1)        sumW2(A1)       numEntries
5.000000e+00    2.500000e+01    0.000000e+00    0.000000e+00    1.000000e+00
3.000000e+00    9.000000e+00    3.000000e+00    3.000000e+00    1.000000e+00
END YODA_BINNEDHISTO<S>_V3
```

- Generalised ASCII Output
  - Extended to support arbitrary dimensions and string-based edges for greater flexibility.
  - Backward compatibility: YODA2 reader supports legacy YODA1 ASCII formats.
- HDF5 Output for High-Performance Computing
  - Ideal for HPC workflows requiring high-throughput processing and scalable data management.
  - Uses the lightweight `HighFive` library for streamlined C++ integration.

# Python API & Plotting for Seamless Integration

- Python Bindings via `Cython`
  - YODA provides Python bindings for scripting and integration into Python-based workflows.
  - Enables efficient use of YODA objects and operations from within Python scripts.
- **Customisable** `matplotlib`-based Plotting
  - Automatically generates Python scripts that produce plots with `matplotlib`.
  - Self-contained plots: Once the script is generated, no YODA installation is required to produce the plot. Ideal for sharing results with collaborators.
  - Full control over plot aesthetics, allowing for customisation without altering the underlying data structures.
- Share Python-generated plotting scripts with collaborators, ensuring **consistency** in results and **reproducibility**.



# Summary & Key Takeaways

[yoda.hepforge.org]

[gitlab.com/hepcedar/yoda]

[packages.spack.io:yoda]

[arXiv:2312.15070]

- Efficient, Scalable Data Handling
  - YODA2 supports live and inert statistical objects with flexible bin partitioning and content storage.
  - Optimised for large-scale datasets and HPC environments through serialisation and parallel computation.
- User-Centric Design
  - Clean and intuitive APIs in both C++ and Python.
  - Self-consistent, customisable plotting with minimal dependencies for easier collaboration.
- Versatility & Extensibility
  - Seamless integration into modern workflows, including machine learning and distributed computing.
  - Backward compatibility with YODA1 and support for both ASCII and HDF5 formats.
- Empowering Particle Physics and Beyond
  - From particle collision data to broader applications in data science and machine learning, YODA2 is **designed for robust, efficient analysis at scale**.

# Backup

# Summary statistics

Analytic first- and second-order statistical moments for probably density function $f(x) \equiv \mathrm{d}P/\mathrm{d}x$

$$\langle x \rangle \equiv \int_{x \in X} x f(x)\, \mathrm{d}x$$

$$\langle x^2 \rangle \equiv \int_{x \in X} x^2 f(x)\, \mathrm{d}x$$

$$\sigma^2(x) \equiv \langle x^2 \rangle - \langle x \rangle^2$$

# Unweighted moments

Unweighted mean and variance for finite-size sample with $1 \leq n \leq N$:

$$\langle \hat{x} \rangle_{\mathrm{U}} \equiv \frac{\sum_{n=1}^{N} x_n}{N}$$

$$\sigma_{\mathrm{U}}^2(\hat{x}) \equiv \frac{\sum_{n=1}^{N} (x_n - \langle x \rangle)^2}{N-1}$$

$$= \langle x^2 \rangle_{\mathrm{U}} - \langle x \rangle_{\mathrm{U}}^2$$

$$= \frac{\sum_{n=1}^{N} x_n^2}{N-1} - \frac{\left(\sum_{n=1}^{N} x_n\right)^2}{(N-1)^2}$$

# Weighted moments

Weighted mean and variance:

$$\langle x \rangle = \frac{\sum_n w_n x_n}{\sum_n w_n}$$

$$\sigma^2(x) = \mathcal{B} \cdot \frac{\sum_n w_n \left(x_n - \sum_m w_m x_m\right)^2}{\left(\sum_n w_n\right)} = \frac{\left(\sum_n w_n x_n^2\right) \cdot \left(\sum_n w_n\right) - \left(\sum_n w_n x_n\right)^2}{\left(\sum_n w_n\right)^2 - \sum_n w_n^2}$$

with weighted Bessel factor:

$$\mathcal{B} = \frac{N_\text{eff}}{N_\text{eff} - 1} = \frac{(\sum_n w_n)^2}{(\sum_n w_n)^2 - \sum_n w_n^2}$$

for effective fill count:

$$N_\text{eff} = \frac{(\sum_n w_n)^2}{\sum_n w_n^2}$$

# Counts and efficiencies

Closely related quantities are Poisson mean and variance:

$$\langle \hat{x} \rangle_\mathrm{P} \equiv N$$

$$\sigma^2_\mathrm{P}(\hat{x}) \equiv N$$

Classic Monte Carlo scaling then given by

$$\frac{\sigma_\mathrm{P}(\hat{x})}{\langle \hat{x} \rangle_\mathrm{P}} = \frac{\sqrt{N}}{N} = \frac{1}{\sqrt{N}}$$

Sample efficiency for selected events $N_\mathrm{sel}$ from a known number of total events $N$ is

$$\hat{\epsilon} \equiv \frac{N_\mathrm{sel}}{N}$$

Binomial statistics gives an estimator for the uncertainty on the efficiency

$$\hat{\sigma}^2(\hat{\epsilon})_\mathrm{B} = \frac{\hat{\epsilon}(1-\hat{\epsilon})}{N}$$

# Connection to differential calculus

- statistical histogram: a **discrete approximation to entire probability density function** $f(\Omega) = \mathrm{d}P/\mathrm{d}\Omega$ or population density $\mathrm{d}N/\mathrm{d}\Omega$, not just a collection of fill counts

# Connection to differential calculus

- statistical histogram: a **discrete approximation to entire probability density function** $f(\Omega) = \mathrm{d}P/\mathrm{d}\Omega$ or population density $\mathrm{d}N/\mathrm{d}\Omega$, not just a collection of fill counts
- bin measure $\mathrm{d}\Omega$ (or $\Delta\Omega$) representing the volume element of the bin crucial for differential consistency

# Connection to differential calculus

- statistical histogram: a **discrete approximation to entire probability density function** $f(\Omega) = \mathrm{d}P/\mathrm{d}\Omega$ or population density $\mathrm{d}N/\mathrm{d}\Omega$, not just a collection of fill counts

- bin measure $\mathrm{d}\Omega$ (or $\Delta\Omega$) representing the volume element of the bin
  crucial for differential consistency

- $\Delta N/\Delta\Omega = \left[N(\Omega + \Delta\Omega) - N(\Omega)\right]/\Delta\Omega \overset{\Delta\Omega\to 0}{=} \mathrm{d}N/\mathrm{d}\Omega$ **necessitates division by bin width**
  - generally not desirable for finite bins to have the same width
  - using non-uniform bin sizes ensures statistical relative uncertainty on bin populations
    is equally distributed across histogram
  - failing to divide by the bin measure distorts the distribution away from its physical shape

# Connection to differential calculus

- statistical histogram: a **discrete approximation to entire probability density function** $f(\Omega) = \mathrm{d}P/\mathrm{d}\Omega$ or population density $\mathrm{d}N/\mathrm{d}\Omega$, not just a collection of fill counts
- bin measure $\mathrm{d}\Omega$ (or $\Delta\Omega$) representing the volume element of the bin crucial for differential consistency
- $\Delta N/\Delta\Omega = \left[N(\Omega + \Delta\Omega) - N(\Omega)\right]/\Delta\Omega \overset{\Delta\Omega\to 0}{=} \mathrm{d}N/\mathrm{d}\Omega$ **necessitates division by bin width**
  - generally not desirable for finite bins to have the same width
  - using non-uniform bin sizes ensures statistical relative uncertainty on bin populations is equally distributed across histogram
  - failing to divide by the bin measure distorts the distribution away from its physical shape
- actual bin populations are better computed using a discrete binning expressed in terms of finite probabilities rather than densities
  - awkward workaround: multiply each density by the fill volume
  - prefer to refer to this not as a histogram but a bar chart, reflecting its typical use

# Lessons from YODA1: Motivation for YODA2

- Initial goals established at YODA1's release in 2013, but structural limitations highlighted the need for a complete redesign.
- Limited support for multi-dimensional data objects and only continuous-valued axes.
- Inability to store arbitrary data types in binnings restricted flexibility.
- Correct but rigid overflow bin treatment lacked flexibility for complex analyses.
- No unified scheme for local and global bin indexing across multiple dimensions, complicating data management.
- Redundant internal implementations to support both C++ and Python APIs for various dimensionalities and content types.
- Difficulty integrating "inert" scatter data types (e.g. measured data from an experiment) with "live" binned objects generated during MC runs.
- Limited, cumbersome support for representing and managing uncertainty breakdowns and correlations in scatter data types.

# Histograms

- generalise measured variable $x$ to vector variable-space $\Omega$
  - composed of vectors $\omega$ with differential volume elements $\mathrm{d}\Omega$
- partition $\Omega$ into disjoint (sub)set of bins $\{\Omega_b\} \subset \Omega$

# Histograms

- generalise measured variable $x$ to vector variable-space $\Omega$
  - composed of vectors $\omega$ with differential volume elements $\mathrm{d}\Omega$
- partition $\Omega$ into disjoint (sub)set of bins $\{\Omega_b\} \subset \Omega$
- moments in each bin $b$ converge to summary properties of that bin's variable-space partition

$$\langle \omega^{(i)} \rangle_b \equiv \int_{\omega \in \Omega_b} \omega^{(i)} f(\omega) \, \mathrm{d}\Omega$$

$$\langle \omega^{(i)} \omega^{(j)} \rangle_b \equiv \int_{\omega \in \Omega_b} \omega^{(i)} \omega^{(j)} f(\omega) \, \mathrm{d}\Omega$$

# Histograms

- generalise measured variable $x$ to vector variable-space $\Omega$
  - composed of vectors $\omega$ with differential volume elements $\mathrm{d}\Omega$
- partition $\Omega$ into disjoint (sub)set of bins $\{\Omega_b\} \subset \Omega$
- moments in each bin $b$ converge to summary properties of that bin's variable-space partition

$$\langle \omega^{(i)} \rangle_b \equiv \int_{\omega \in \Omega_b} \omega^{(i)} f(\omega) \, \mathrm{d}\Omega$$

$$\langle \omega^{(i)} \omega^{(j)} \rangle_b \equiv \int_{\omega \in \Omega_b} \omega^{(i)} \omega^{(j)} f(\omega) \, \mathrm{d}\Omega$$

- need to recover unbinned values when expanding the partition to whole space
- need to recover differential properties of the pdf itself as $\Omega_b \to \mathrm{d}\Omega(\omega)$
- merging bins must converge to the same result as having originally constructed a lower-dimensional or less finely binned partition of space

# Profiles

- useful class of histogram mixing binned and unbinned variable subspaces
- allow characterisation of the unbinned dimensions $\Upsilon$ via their moments as projected into each partition of the bin-space $\Theta$
  - allow statistical aggregation of finite samples into "independent variable" bins $\theta \in \Theta_b$, while characterising the mean dependence of the unbinned dependent variables $y$ on $\theta$
  - linearity of statistical moments again ensures consistency when merging bins

# Profiles

- useful class of histogram mixing binned and unbinned variable subspaces
- allow characterisation of the unbinned dimensions $\Upsilon$ via their moments as projected into each partition of the bin-space $\Theta$
  - allow statistical aggregation of finite samples into "independent variable" bins $\theta \in \Theta_b$, while characterising the mean dependence of the unbinned dependent variables $y$ on $\theta$
  - linearity of statistical moments again ensures consistency when merging bins
- unbinned space $\Upsilon$ can in general be multidimensional but canonical bin value then ambiguous
- definiteness retained for single-dimensional unbinned space with moments $\langle y \rangle$ and $\langle y^2 \rangle$
  - profile canonical bin value is the mean $\langle y(\Theta) \rangle$ as a function of binned coordinates
  - nominal uncertainty given by standard error $\hat{\sigma}_{\bar{y}}(\theta) = \hat{\sigma}_b / \sqrt{N_b}$ for effective sample count $N_b$ in bin $b \subset \theta$

# Example: construction and filling

```
// declaration examples
Histo1D h1; // histogram with 1 continuous axis
Profile2D p1; // profile with 2 continuously binned axes + 1 unbinned axis
HistoND<5> h2; // histogram with 5 continuous axes


// constructor examples
Histo1D h3(10, 0, 100); // 10 bins between 0 and 100
const std::vector<double> edges = {0, 10, 20, 30, 40, 50};
Histo1D h4(edges);
BinnedHisto<int, std::string> h5({ 1, 2, 3 }, { "A", "B", "C" });


// fill examples
Histo1D h6(5, 0.0, 1.0);
h6.fill(0.2);
Profile1D p2(5, 0.0, 1.0);
p2.fill(0.2, 3.5);


// marginalisation examples
Histo2D h7 = p1.mkHisto(); //< marginalise over unbinned axis
Histo1D h8 = h7.mkMarginalHisto<1>(); //< marginalise over secomd binned axis
Histo1D h9 = p1.mkMarginalProfile<0>(); //< marginalise over first binned axis
```

# Example: looping and indexing

```
size_t nbinsX = 4, nbinsY = 6;
double lowerX = 0, lowerY = 0;
double upperX = 4, upperY = 6;
Histo2D h2(nbinsX, lowerX, upperX,
           nbinsY, lowerY, upperY);

// loop over bins and fill with increasing weight
double w = 0;
for (auto& b : h2.bins()) { //< iterators passes through using templated bin wrappers
  h2.fill(b.xMid(), b.yMid(), ++w);
}

for (size_t idxY = 0; idxY < h2.numBinsY(true); ++idxY) { //< true includes overflows
  for (size_t idxX = 0; idxX < h2.numBinsX(true); ++idxX) { //< true includes overflows
    std::cout << "\t(" << idxX << "," << idxY << ")\t=\t";
    std::cout << h2.bin(idxX, idxY).sumW();
  }
  std::cout << std::endl;
}
std::cout << std::endl;

# H2 bins using local indices + under/overflows:
#   (0,0) = 0 (1,0) =  0 (2,0) =  0 (3,0) =  0 (4,0) =  0 (5,0) = 0
#   (0,1) = 0 (1,1) =  1 (2,1) =  2 (3,1) =  3 (4,1) =  4 (5,1) = 0
#   (0,2) = 0 (1,2) =  5 (2,2) =  6 (3,2) =  7 (4,2) =  8 (5,2) = 0
#   (0,3) = 0 (1,3) =  9 (2,3) = 10 (3,3) = 11 (4,3) = 12 (5,3) = 0
#   (0,4) = 0 (1,4) = 13 (2,4) = 14 (3,4) = 15 (4,4) = 16 (5,4) = 0
#   (0,5) = 0 (1,5) = 17 (2,5) = 18 (3,5) = 19 (4,5) = 20 (5,5) = 0
#   (0,6) = 0 (1,6) = 21 (2,6) = 22 (3,6) = 23 (4,6) = 24 (5,6) = 0
#   (0,7) = 0 (1,7) =  0 (2,7) =  0 (3,7) =  0 (4,7) =  0 (5,7) = 0
```

# Variadic templates and parameter packs

➜ Metaprogramming using C++17 takes care of generalisation to arbitrary dimensions:

```
#include <iostream>
#include <string>
#include <tuple>
#include <vector>

template <typename... Args>
class MyHisto {
public:
  MyHisto(const std::vector<Args>& ... edges)
    : _axes(edges ...) { }

  size_t dim() const { return sizeof...(Args); }

  template<size_t I>
  void printBinning() const {
    if constexpr (I < sizeof...(Args)) {
      std::cout << "Axis" << (I+1) << "has";
      std::cout << std::get<I>(_axes).size();
      std::cout << "bins." << std::endl;
      printBinning<I+1>();
    }
  }

  void print() const {
    std::cout << dim() << "D:" << std::endl;
    printBinning<0>();
  }

private:
  std::tuple<std::vector<Args>...> _axes;
};
```