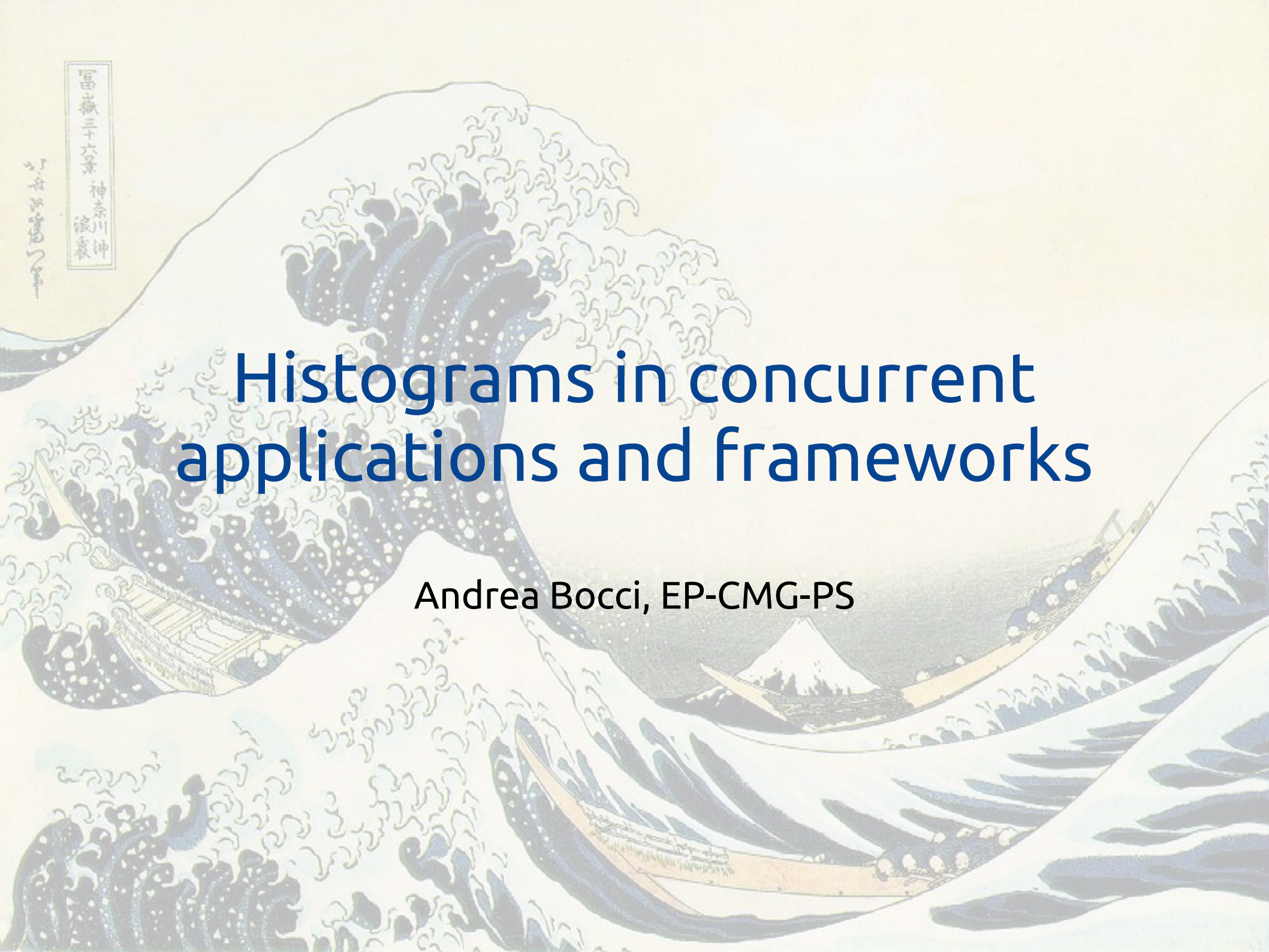


Histograms in concurrent applications and frameworks

Andrea Bocci, EP-CMG-PS

Disclaimer

- My background is mostly the CMS High Lever Trigger
- There are definitely other people in CMS more expert than me in
  - ROOT and histograms
  - concurrency
- My interest is motivated by the limitations we encountered with the approach used so far
  - one copy of each histogram per “stream” (think thread)
  - multiple copies are filled independently and merged
- Unwanted consequences and side effects
  - during the high level trigger: large time spent merging histograms
  - during prompt reconstruction: large memory consumption

Why concurrent histograms ?

- experimental frameworks support concurrency at various levels:
  - multiple “modules” within an event (e.g. tracking vs jet clustering)
  - multiple objects within a single module (e.g. tracks or jets)
  - process multiple events
  - process multiple “runs” and “segments”
  - include capabilities to book, fill and manipulate histograms
  - for example: Data Quality Monitoring in CMSSW, based on ROOT

- two possibilities at the opposite ends of the spectrum
  - the underlying histogram library provides support for concurrency
    - well defined interface to fill the same or multiple histograms at the same time
    - and to access the histograms’ properties, statistics, etc.
  - the underlying histogram library makes no assumptions about concurrency
    - e.g. ROOT 5, ROOT 6
    - experimental software needs to wrap it with *ad hoc* locking, interface, etc.

What does it mean ?

- experimental frameworks support concurrency at various levels:
  - multiple “modules” within an event (e.g. tracking vs jet clustering)
  - multiple objects within a single module (e.g. tracks or jets)
  - process multiple events
  - process multiple “runs” and “segments”

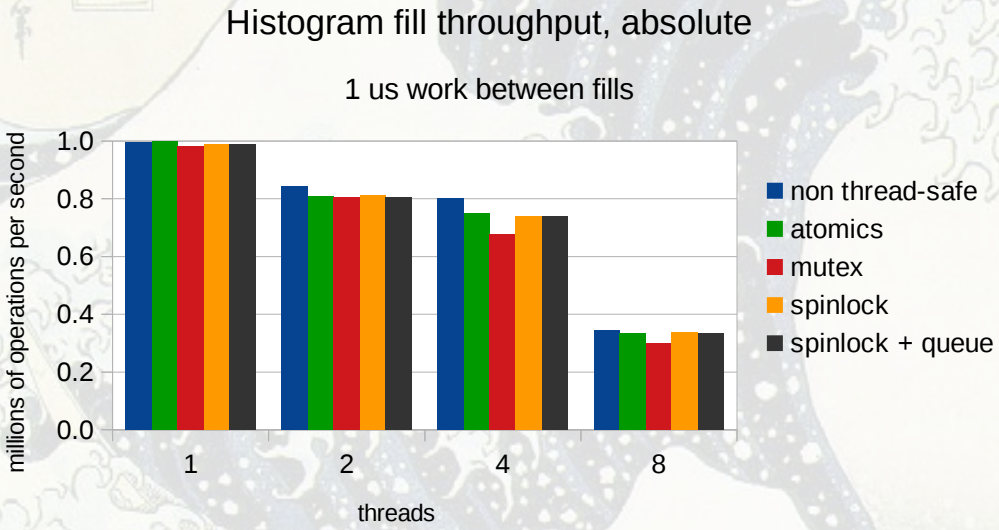
- fill multiple histograms at the same time
  - easy
- fill concurrently an histogram
  - many possible implementations ?
- keep multiple copies of each histogram
  - easy in client code, not trivial in central code ?

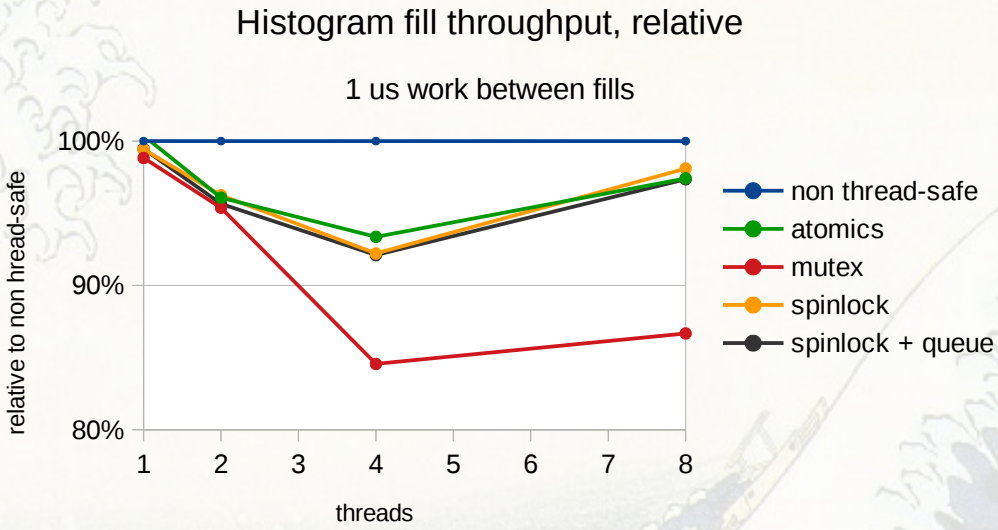
What kind of R&D ?

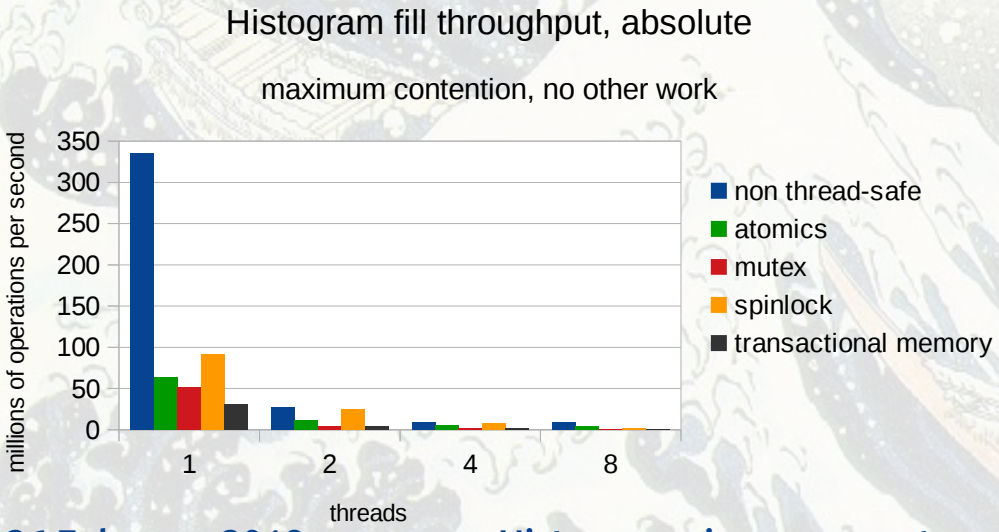
- discussed on and off for the past few years
  - ROOT Team Retreat, July 2014: *Histogram and Concurrency*, L. Moneta
  - work in progress for ROOT 7 ?
- concurrent filling
  - what choice of technology ?
    - mutex vs spinlock vs atomics vs transactional memory vs queues vs …
  - optimise for ...
    - … low or high contention ?
    - … one or few or many threads ?
- access to data and statistics
  - concurrently ? only via “unsafe” methods ?
- whole object operations ?

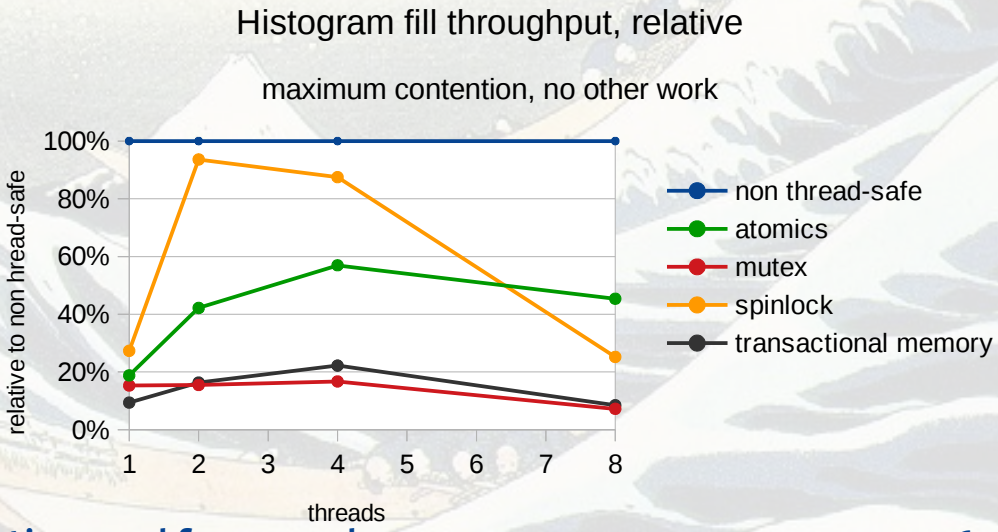
Possibilities …

- Trivial test filling a histogram with fixed bins and minimal stats
  - different contention, number of threads, implementations









… and deliverables

- a library for concurrent histogram booking, filling and handling
  - lightweight
  - development and distribution
    - standalone
    - within ROOT 7
  - lightweight
  - interoperability with other concurrency frameworks
    - TBB
    - C++ 11 – C++ 20