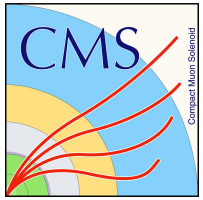


Incontri di Fisica delle Alte Energie 2023 (IFAE 2023)

Catania, 12-14 Aprile 2023

Benchmark di un nuovo modello di analisi per la Fase 2 di CMS su risorse INFN

Tommaso Tedeschi[1,2] (tommaso.tedeschi@pg.infn.it) per la collaborazione CMS

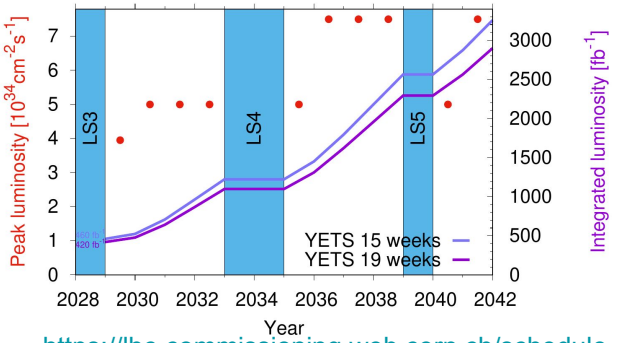
Diego Ciangottini[1], Daniele Spiga[1]

Si ringraziano i ROOT/SWAN developers: V. Padulano, E. Guiraud, E. Teejedor

[1]Istituto Nazionale di Fisica Nucleare - Sezione di Perugia, Perugia, Italy
[2]Università degli Studi di Perugia, Perugia, Italy

Upgrade di CMS Fase-2

- HL-LHC:
  - Luminosità massima: **7.5×10^{34} $s^{-1}cm^{-2}$** (fattore 4x rispetto a Run 2):
  - Pileup previsto 140-200 (rispetto a 55 di Run 3)
- Upgrade di Fase-2 dell'esperimento CMS:
  - Granularità aumentata (e.g. Tracker) e introduzione di detector aggiuntivi (e.g. HGCal e MTD):
    - Incremento dei canali di readout (fattore di almeno 100x)
  - Aumento dell'output prompt trigger rate da 2.5 a 7.5 kHz
- Tutto ciò si traduce in un **aumento significativo delle richieste di CPU per acquisire/generare e processare eventi, e di Disco per la loro archiviazione**



https://lhc-commissioning.web.cern.ch/schedule/HL-LHC-plots.htm

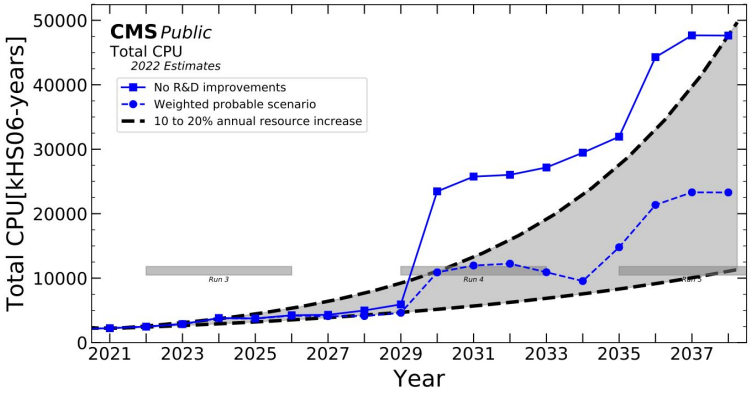
Parameter	Run 4 ('29-'32)	Run 5 ('35-'38)
Common		
LHC Energy [TeV]	14	
Average PU	140 (70 in '29)	200 (100 in '35)
Integrated luminosity / year [fb^{-1}]	270 (135 in '29)	340 (170 in '35)
Livetime pp / year [$s/10^6$]	6	6
Livetime HI / year [$s/10^6$]	1.2	1.2
Yearly capacity evolution under flat budget for disk, CPU, and tape (hardware replacement included)	$+15 \pm 5\%$	
CMS-Specific		
Prompt HLT Rate [kHz]	5	7.5
Collected events / year (10^9)	34	51
MC events / year (10^9)	85	104
CPU-GPU cost ratio per unit computation	2.8x	

https://cds.cern.ch/record/2815292

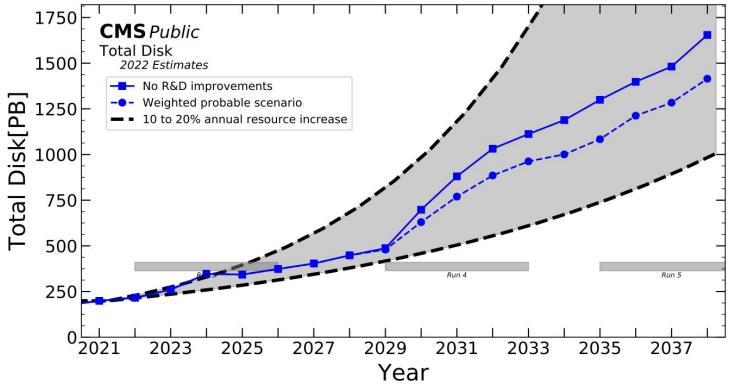
Le proiezioni di HL-LHC

In particolare, le richieste in termini di **risorse di calcolo** saranno **superiori** a ciò che può essere ottenuto con l'attuale modello di calcolo, senza miglioramenti introdotti da **attività di R&D** e assumendo un **budget costante** (fattore di circa 3/4x)

CPU



Disco

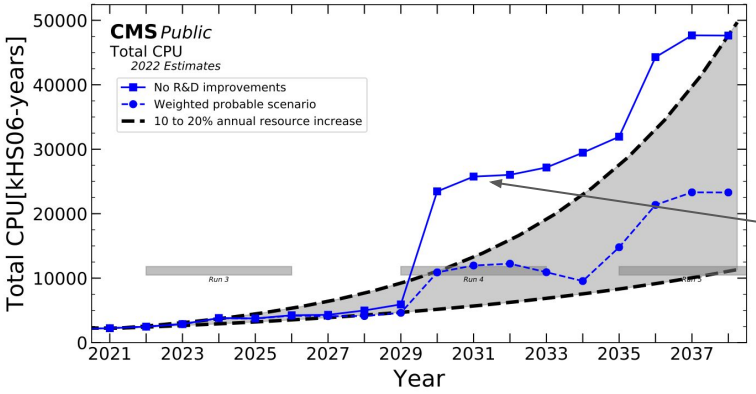


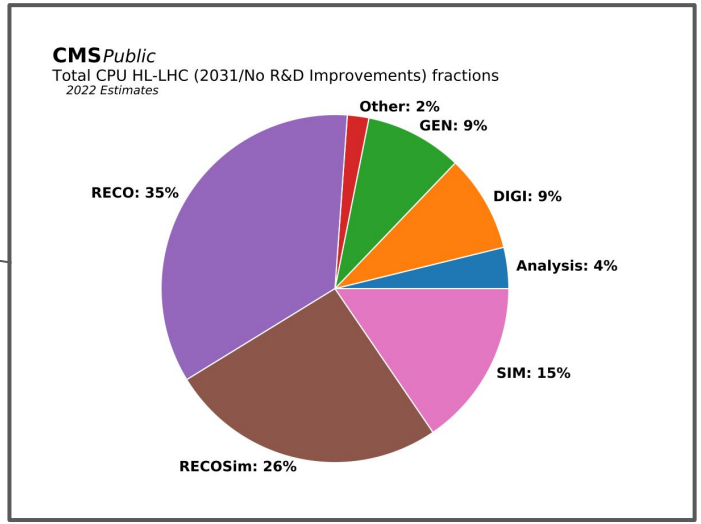
https://cds.cern.ch/record/2815292

Le proiezioni di HL-LHC

In particolare, le richieste in termini di **risorse di calcolo** saranno **superiori** a ciò che può essere ottenuto con l'attuale modello di calcolo, senza miglioramenti introdotti da **attività di R&D** e assumendo un **budget costante** (fattore di circa 3/4x)

CPU



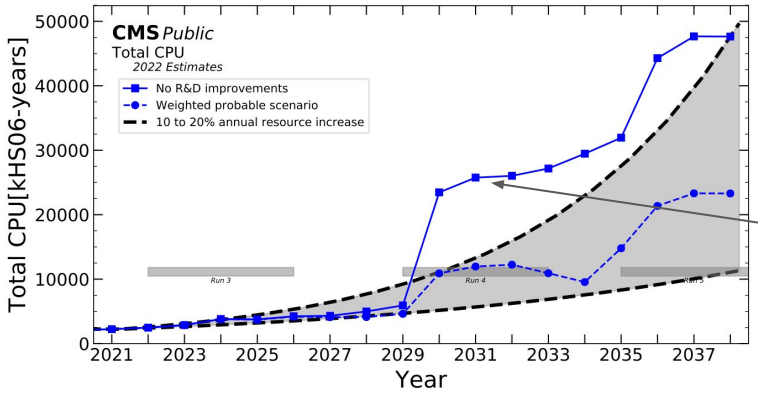


https://cds.cern.ch/record/2815292

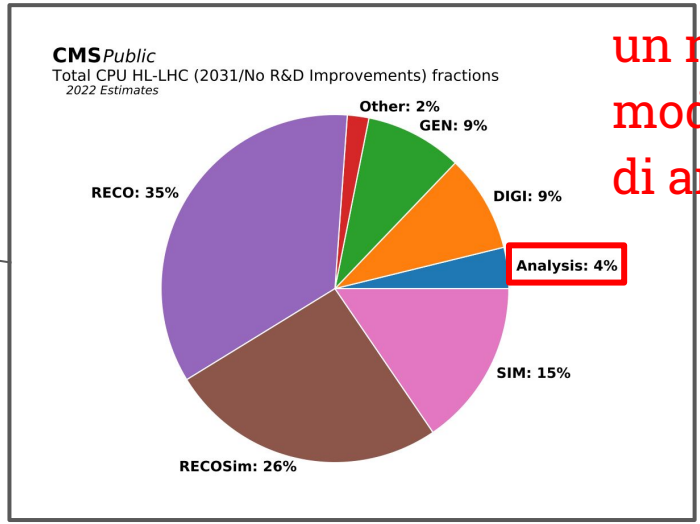
Le proiezioni di HL-LHC

In particolare, le richieste in termini di **risorse di calcolo** saranno **superiori** a ciò che può essere ottenuto con l'attuale modello di calcolo, senza miglioramenti introdotti da **attività di R&D** e assumendo un **budget costante** (fattore di circa 3/4x)

CPU



https://cds.cern.ch/record/2815292



R&D per un nuovo modello di analisi

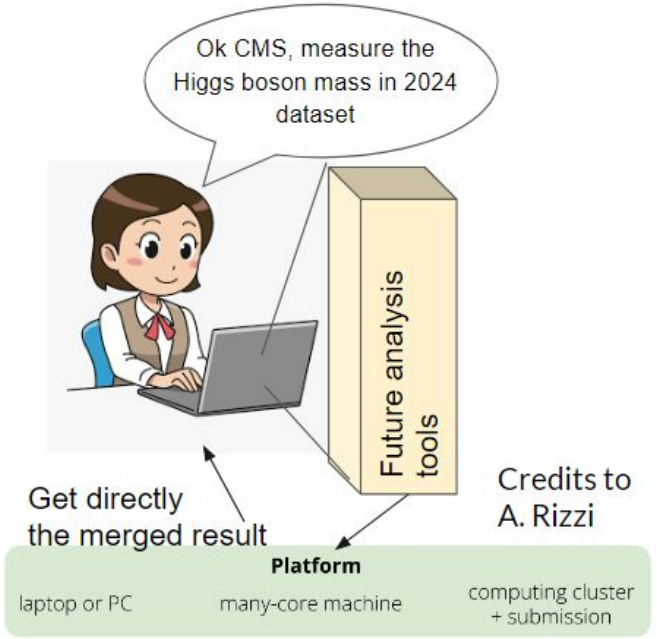
Un nuovo modello di analisi

L'obiettivo finale è sviluppare un modello di analisi che, utilizzando le risorse attuali, consenta di **mantenere lo stesso rate di produzione di risultati di fisica attuale nell'era dell'alta luminosità**

L'approccio attuale basato solamente su "batch" (rate di 100-1000 Hz) potrebbe non essere sufficiente per tale scopo

La soluzione proposta prevede:

- l'accesso alle risorse di calcolo per l'analisi tramite un modello ibrido (sia batch che interattivo):
  - **E.g. INFN Analysis Facility**
- L'adozione sempre più diffusa per l'analisi di formati dati estremamente ridotti:
  - **NanoAOD**
- l'introduzione di tools di analisi dichiarativa:
  - una possibile scelta è **RDataFrame**



NanoAOD

NanoAOD: formato dati colonnare estremamente ridotto (**1/2 kB/evento**):

- Utilizzo di **data types base** (e.g. float, int, arrays),
- Struttura basata su **semplici ROOT TTrees**
- **Solo** variabili legate a **oggetti fisici di alto livello**, incluse quantità precalcolate legate alla loro identificazione:
  - **filtrati** usando soglie appropriate

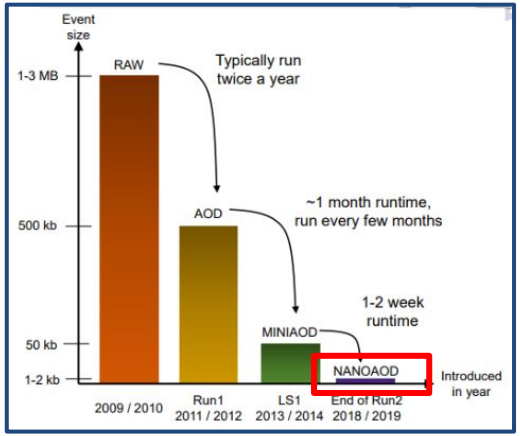
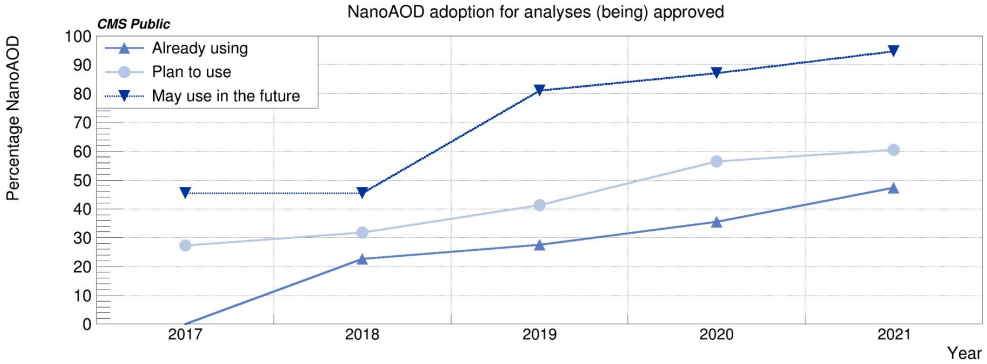


Immagine riprodotta da "NANOAOD: a new compact event data format in CMS", Karl Ehatäht, CHEP 2019 https://indico.cern.ch/event/773049/contributions/3476049/attachments/1933365/3203526/poster_v3.pdf



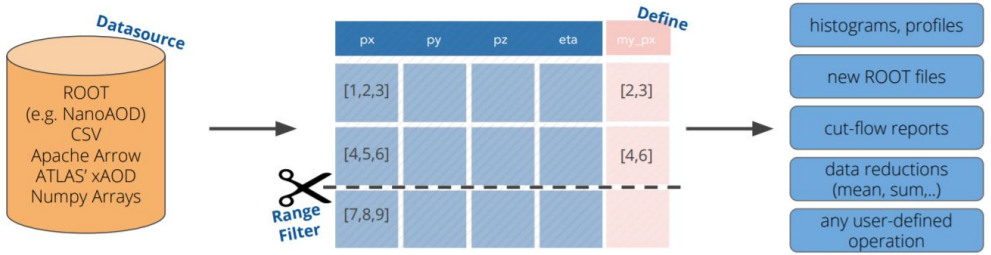
RDataFrame

RDataFrame è l'**interfaccia di alto livello** di ROOT per l'analisi dei dati archiviati in TTree, CSV e altri formati di dati. È caratterizzata da:

- multi-threading
- ottimizzazioni di basso livello (parallelizzazione e caching).

I calcoli sono espressi in termini di una **catena di azioni e trasformazioni**, che costituiscono un grafo computazionale.

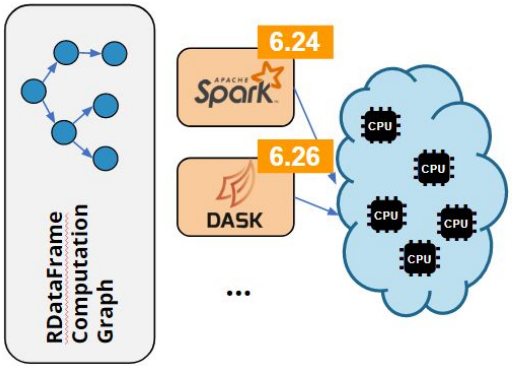
L'esecuzione del grafo può essere effettuata in maniera **distribuita** sfruttando back-end quali Spark e Dask




```
# enable multi-threading
ROOT.EnableImplicitMT()
df = ROOT.RDataFrame(dataset)
```

```
df = df.Range(2)
      .Define("my_px", "px[eta > 0]")
```

```
# filled in a single loop
h1 = df.Histo1D("my_px", "w")
h2 = df.Histo1D("px", "w")
```



Immagini riprodotte da "A Python package for distributed ROOT RDataFrame analysis", V. Padulano, PyHEP 2021

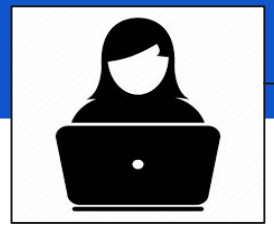
La Analysis Facility INFN

Via Jupyterlab:

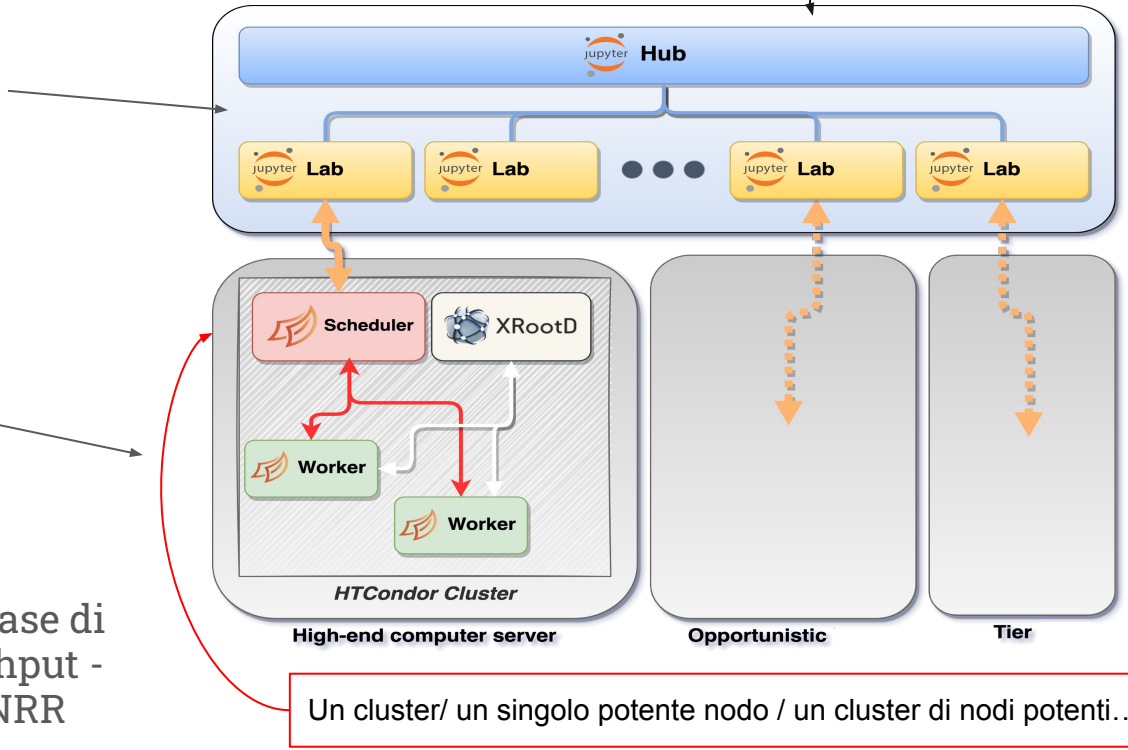
- **Approccio interattivo:**
  - **notebook + Dask + HTCondor**
- Approccio batch:
  - terminale + HTCondor

Tutte le risorse INFN disponibili (distribuite geograficamente) vengono integrate e nascoste dietro un singolo Hub

Implementazioni tecnologiche ed infrastrutturali come questa sono alla base di soluzioni per analisi dati ad alto throughput - attività contestualizzate nel contesto PNRR



https://infn-cms-analysisfacility.readthedocs.io/en/latest/



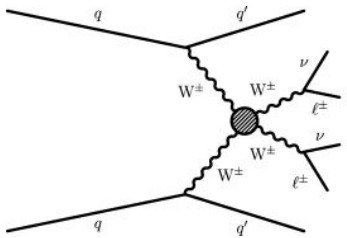
Un cluster/ un singolo potente nodo / un cluster di nodi potenti…

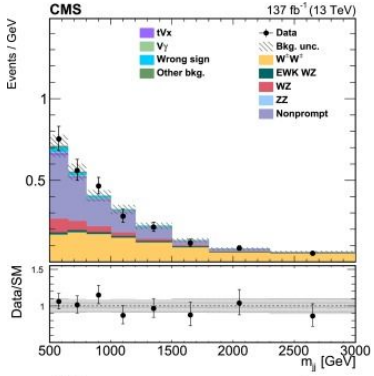
VBS ssWW come benchmark

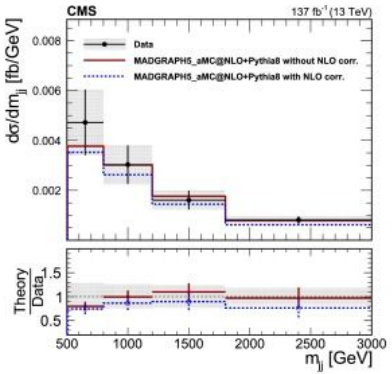
Vector Boson Scattering (VBS):

- **A LHC**:
  - due quark ad alta energia provenienti da ciascuno dei due protoni in collisione emettono due bosoni vettori, che a loro volta interagiscono tra loro, decadendo infine nelle particelle rilevate dagli esperimenti
- **Segnatura sperimentale peculiare:**
  - due jet forward-backward molto energetici, elevati m_{jj} e $\Delta\eta_{jj}$
- **Sonda eccezionale per la fisica EW**:
  - la divergenza dell'ampiezza della sua **componente longitudinale** è evitata da una perfetta cancellazione tra i diagrammi di Feynman mediati dal bosone di Higgs e quelli non mediati dal bosone di Higgs nel canale t
- Uno dei canali più puliti per studiare un tale processo è **lo scattering di due bosoni W dello stesso segno (ssWW)**:
  - elevata sezione d'urto (0.03 pb) e alto contributo relativo della produzione EW rispetto a quella QCD (quasi 1:1)

ssWW VBS già osservato a CMS nel canale fully-leptonic







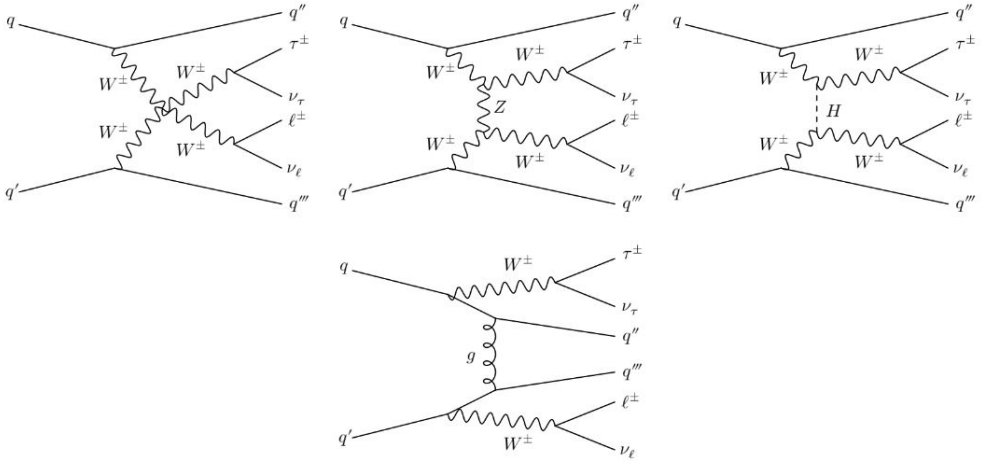
https://doi.org/10.1016/j.physletb.2020.135710

ssWW in VBS in e/mu + tau

Per ampliare lo spazio delle fasi, CMS sta sviluppando l'analisi di eventi di **VBS ssWW nello stato finale e/mu + tau adronico:** il tau importante sonda grazie alla sua massa elevata

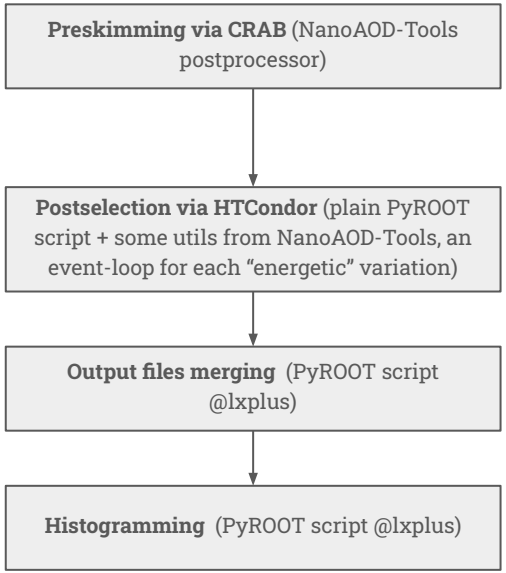
Fondi principali:

- **Leptoni prompt (stima MC):**
  - Leptoni negli stati finali da interazioni prompt
  - Contributo principale all'analisi:
    - ttbar dileptonico
    - DY + jet
- **Leptoni non-prompt (stima data-driven)**:
  - Jet ricostruiti come leptoni (e, mu, tau)
  - Principalmente dovuto a:
    - Multijet,
    - W + jet,
    - ttbar full/semi-adronico



Porting dell'analisi

Implementazione legacy



Analisi portata dall'approccio "legacy" (PyROOT/NanoAOD-tools) ad uno basato su RDataFrame ed utilizzata per fare studi di comparazione.

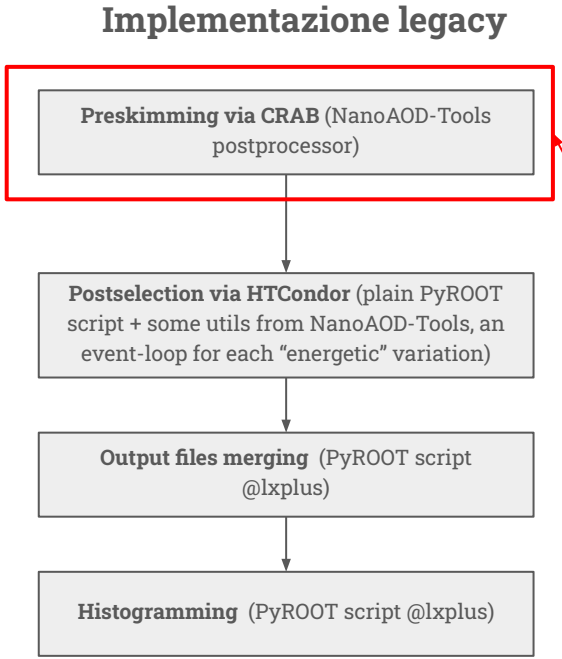
Motivi della scelta:

- Dimensione "media" dell'analisi:
  - O(TB) di dati + MC da analizzare: O(1mld) di eventi MC per anno di presa dati
  - O(100k) eventi MC negli istogrammi finali per anno di presa dati
- Interesse per il VBS a Run 3 e oltre
- Già implementata su NanoAOD

Procedura di analisi basata su due step:

- Preselection
- Postselection

Porting dell'analisi

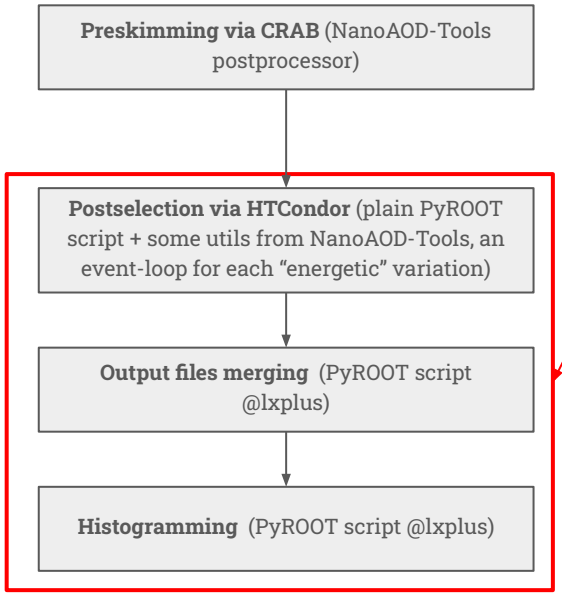


Preselection:
Filtri basati su trigger e richieste "loose" sugli oggetti dello stato finale, definizione di quantità di correzione

- Dimensione dataset da analizzare: 2 TB per anno di presa dati (1 TB dati, 1 TB MC di cui segnale 12 GB)
- Tempo di esecuzione: O(10h) su CRAB

Porting dell'analisi

Implementazione legacy



Postselection:
Ricostruzione vera e propria dell'evento con la produzione degli istogrammi di variabili rilevanti per ciascuna variazione, sample, regione cinematica e stato finale

- Dimensione dataset da analizzare: O(10 GB) per anno
- Tempo di esecuzione: O(1h) sul CERN Batch Service

Porting dell'analisi

Implementazione legacy

Preskimming via CRAB (NanoAOD-Tools postprocessor)

Postselection via HTCondor (plain PyROOT script + some utils from NanoAOD-Tools, an event-loop for each "energetic" variation)

Output files merging (PyROOT script @lxplus)

Histogramming (PyROOT script @lxplus)

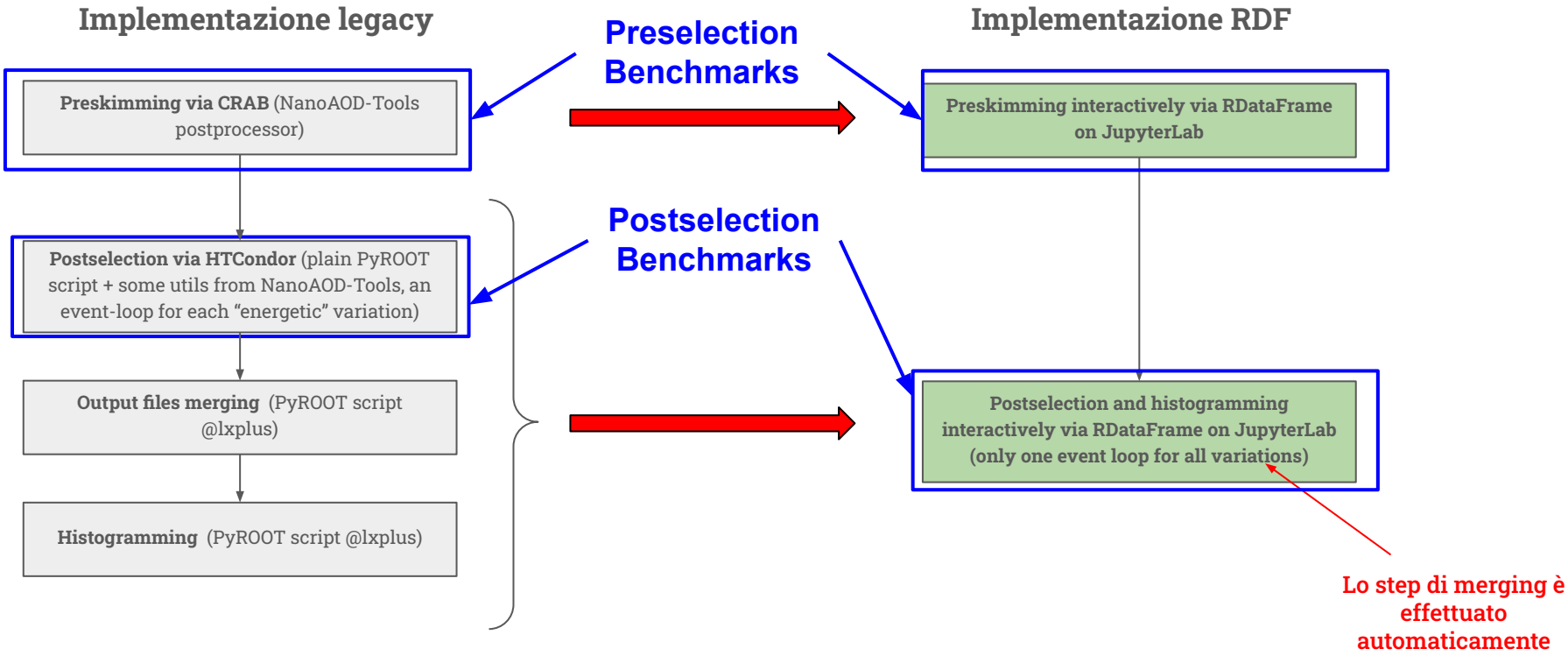
Implementazione RDF

Preskimming interactively via RDataFrame on JupyterLab

Postselection and histogramming interactively via RDataFrame on JupyterLab (only one event loop for all variations)

Lo step di merging è effettuato automaticamente

Porting dell'analisi



Risultati del benchmark

Considerando l'analisi MC Run 2017 (1.1 TB, 1274 file, circa 700 mln di eventi):

I due approcci sono stati testati sulle stesse identiche risorse e nelle stesse identiche condizioni, per i due step di pre e postselection

Le metriche di confronto sono:

- **tempo complessivo** di esecuzione
- **rate** medio **del singolo job/task** (numero di eventi processati al secondo) considerando (totale) o meno (event loop) il tempo di inizializzazione
- **Network read** (quantità totale di dati letti da remoto)

I risultati dimostrano che per questo use case, considerando pre + post:

- il **guadagno in termini di tempo è di un fattore 6**
- e **la riduzione in termini di network read è del -33%**

Preselection		
	Legacy	**RDF**
Tempo complessivo [min]	181 ± 1	23.8 ± 0.6
Rate totale [Hz]	786 ± 12	6915 ± 35
Rate event loop [Hz]	858 ± 14	7632 ± 34
Network read totale [GB]	485 ± 1	362.5 ± 0.1

Postselection (3 variabili considerate - circa 7000 istogrammi)		
	Legacy	**RDF**
Tempo complessivo [min]	48.3 ± 0.5	12.6 ± 0.3
Rate totale [Hz]	62.9 ± 0.1	288 ± 1
Rate event loop [Hz]	65.69 ± 0.05	355 ± 3
Network read totale [GB]	84.46 ± 0.08	17.46 ± 0.08

Conclusioni e proiezioni future

- Se questi numeri venissero confermati su un range più ampio di analisi il cambio di paradigma per Run 4 potrebbe essere giustificato:
  - Cambiamento necessario solo a livello software:
    - Il prototipo dell'AF prevede l'utilizzo di risorse attuali
- Ciò dimostra l'importanza strategica delle attività di R&D per CMS
- Questo motiva ulteriori studi:
  - Inclusione nei test di diversi strumenti (ad esempio Coffea)
  - integrazione di interfacce legacy (ad esempio quella dei strumenti NanoAOD) con i back-end moderni

BACK UP

How RDF code looks like, in a nutshell

```
def inizialization_function():
    ROOT.gInterpreter.Declare('#include "utils_functions.h"')

df = ROOT.RDF.Experimental.Distributed.Dask.RDataFrame("Events", chain, nPartitions = N, client = client)  #define the dataframe

df_processed = df.Define("column_c", "function(column_a, column_b)")\
                 .Filter("filtering_function(column_d)", "A filter")
                 ...

# book a snapshot (i.e. a saving)-> used in preselection
opts = ROOT.RDF.RSnapshotOptions()
opts.fLazy = True
df_lazy_snapshot = df_processed.Snapshot("treeName", "fileName.root", opts)

# book an histogram -> used in postselection
lazy_histo = df_lazy_snapshot.Histo1D("column_c", "weights_column")

# to trigger execution
histo = lazy_histo.GetValue()

# to inspect data
df_saved.Display(["column_a", "column_b", "column_c"], nRows = 1).Print()
+-----+----------+----------+----------+
| Row | column_a | column_b | column_c |
+-----+----------+----------+----------+
| 0   | -1       | -1       | -1       |
+-----+----------+----------+----------+
```

How RDF code looks like, in a nutshell

```
def inizialization_function():
    ROOT.gInterpreter.Declare('#include "utils_functions.h"')

df = ROOT.RDF.Experimental.Distributed.Dask.RDataFrame("Events", chain, nPartitions = N, client = client)  #define the dataframe

df_processed = df.Define("column_c", "function(column_a, column_b)")\
                 .Filter("filtering_function(column_d)", "A filter")
                 ...

# book a snapshot (i.e. a saving)-> used in preselection
opts = ROOT.RDF.RSnapshotOptions()
opts.fLazy = True
df_lazy_snapshot = df_processed.Snapshot("treeName", "fileName.root", opts)

# book an histogram -> used in postselection
lazy_histo = df_lazy_snapshot.Histo1D("column_c", "weights_column")

# to trigger execution
histo = lazy_histo.GetValue()

# to inspect data
df_saved.Display(["column_a", "column_b", "column_c"], nRows = 1).Print()
+-----+----------+----------+----------+
| Row | column_a | column_b | column_c |
+-----+----------+----------+----------+
| 0   | -1       | -1       | -1       |
+-----+----------+----------+----------+
```

Funzioni C++ che manipolano oggetti ROOT::RVec

How RDF code looks like, in a nutshell

```
def inizialization_function():
    ROOT.gInterpreter.Declare('#include "utils_functions.h"')

df = ROOT.RDF.Experimental.Distributed.Dask.RDataFrame("Events", chain, nPartitions = N, client = client)  #define the dataframe
```

```
df_processed = df.Vary("column_a", "...", "...")\
                 .Define("column_c", "function(column_a, column_b)")\
                 .Filter("filtering_function(column_d)", "A filter")
                 ...
```

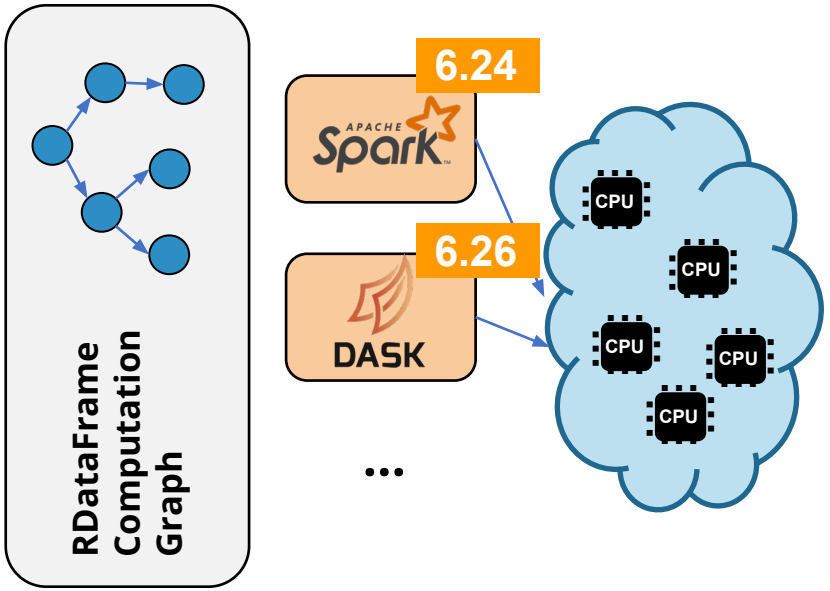
```
# book a snapshot (i.e. a saving)-> used in preselection
opts = ROOT.RDF.RSnapshotOptions()
opts.fLazy = True
df_lazy_snapshot = df_processed.Snapshot("treeName", "fileName.root", opts)

# book an histogram -> used in postselection
lazy_histo = df_lazy_snapshot.Histo1D("column_c", "weights_column")
```

```
lazy_histo_varied = ROOT.RDF.Experimental.Distributed.VariationsFor(lazy_histo)
```

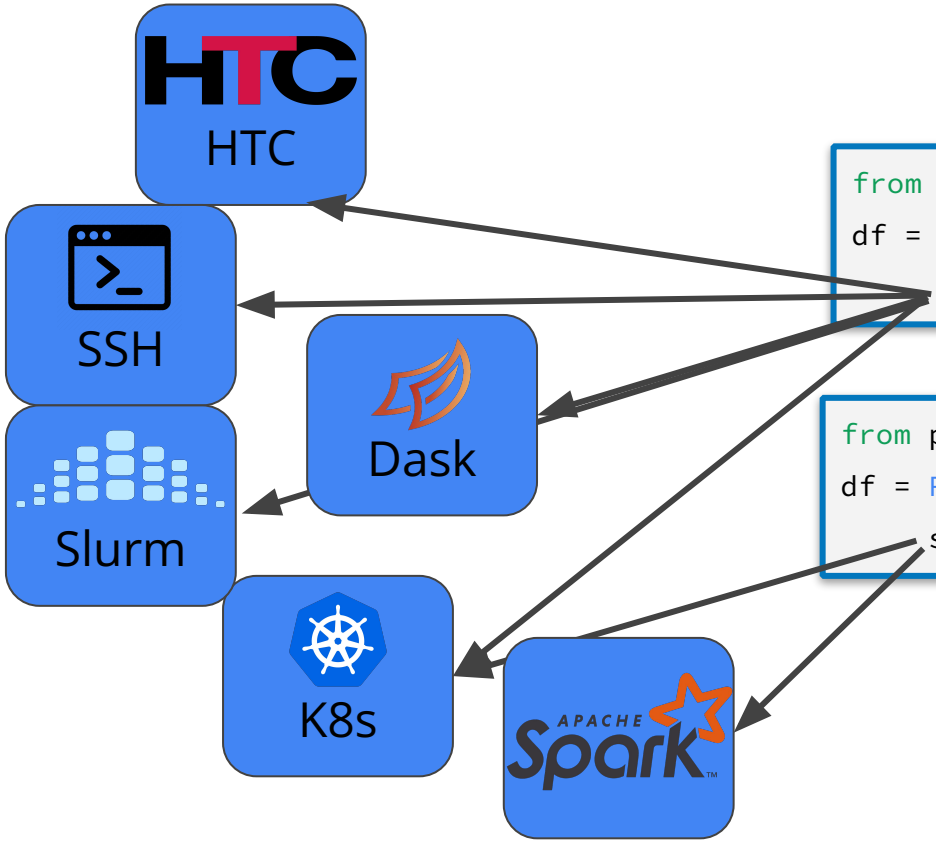
```
# to inspect data
df_saved.Display(["column_a", "column_b", "column_c"], nRows = 1).Print()
+-----+----------+----------+----------+
| Row | column_a | column_b | column_c |
+-----+----------+----------+----------+
| 0   | -1       | -1       | -1       |
+-----+----------+----------+----------+
```

Distributed RDataFrame

- Avvia un'applicazione RDataFrame su un cluster
- Splitting automatico del workflow
- Si occupa dell'esecuzione dei job e del merge dei risultati
- Può essere eseguito con diversi scheduler: Dask, Spark, ...
- Analisi end-to-end con un'unica interfaccia



One API, Many Backends




```python
from dask.distributed import Client
df = RDataFrame('treename', 'filename.root',
     daskclient = Client('tcp://hostname:port'))
```

```python
from pyspark import SparkContext
df = RDataFrame('treename', 'filename.root',
     sparkcontext = SparkContext('spark://IP:PORT'))
```

Perché un approccio dichiarativo

Approccio batch-like:

- Invia O(1000) single-core batch jobs
- Risottomissione dei job falliti
- Fai il merging

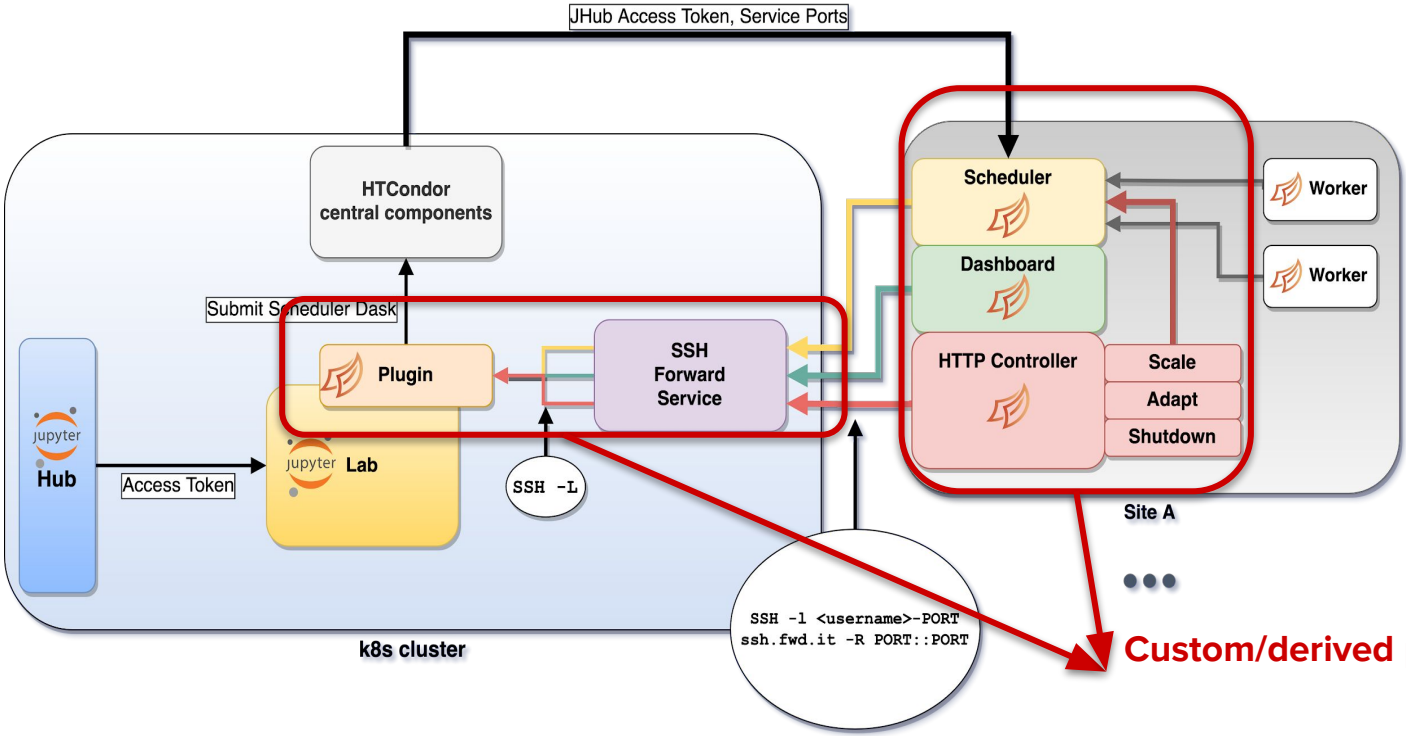
Approccio dichiarativo:

- Evitare di scrivere codice “boilerplate” in favore di un **Event loop implicito**
- **Nessuna ottimizzazione necessaria** per la suddivisione dei dati o per il multithreading esplicito
- Il codice condiviso tra i membri della collaborazione è solamente quello “**human readable**"

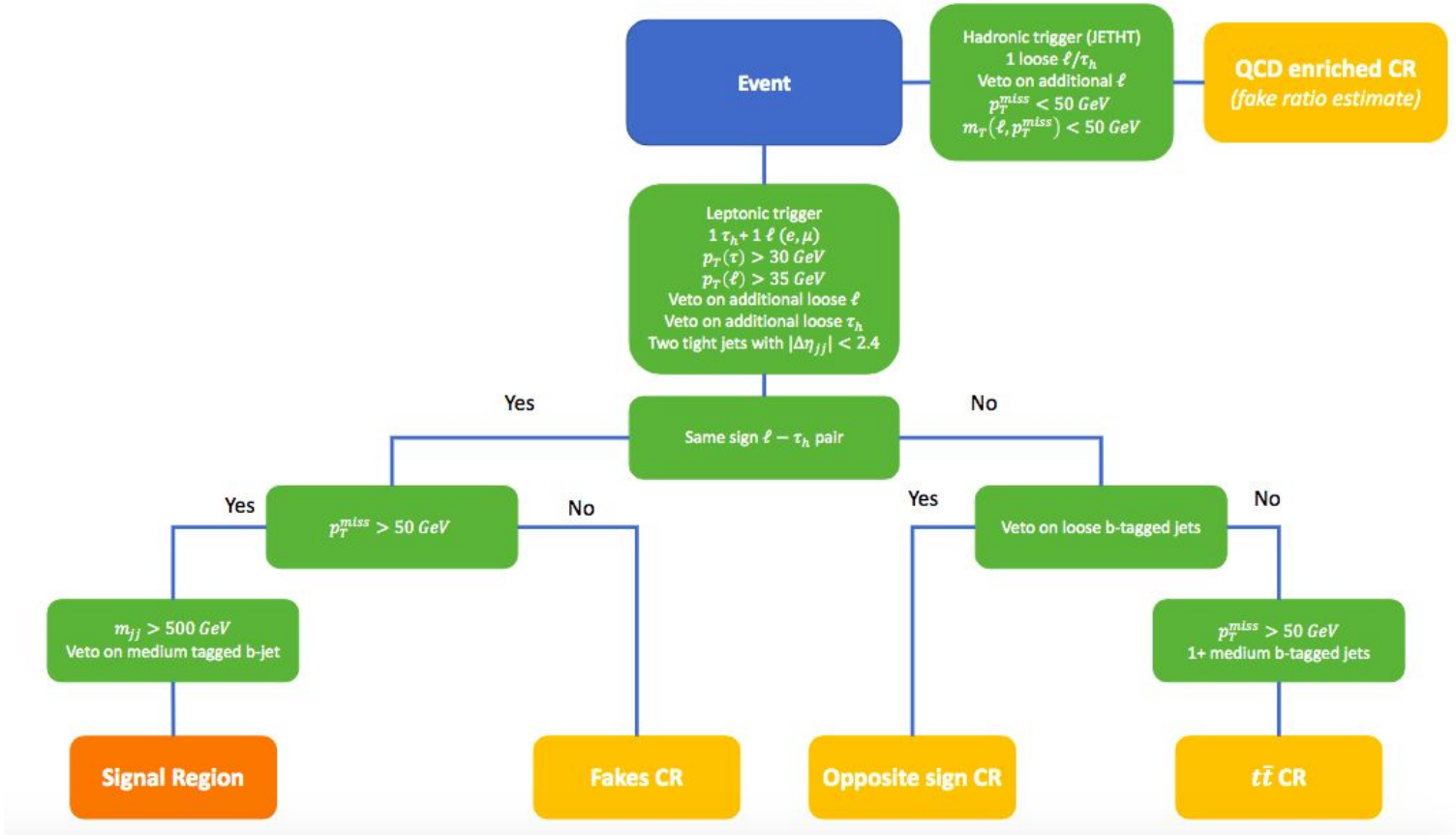
Dettagli della AF INFN

Tre elementi costitutivi:

- JupyterHub (JHub) e JupyterLab (JLab) per gestire la parte dell'infrastruttura rivolta all'utente (possibile anche accesso via CLI)
- Dask per introdurre lo scale-up su un sistema batch (HTCondor)
- XRootD come protocollo di accesso ai dati verso AAA



Regioni cinematiche analisi



Comparazione dei due approcci

- Metriche:
  - tempo di esecuzione complessivo
  - rate (eventi/s), sia complessiva (considerando il tempo di inizializzazione) che relativa al solo event-loop, calcolata sui singoli job/task
  - bytes letti dalla rete
- Entrambi gli approcci sono stati testati sulle stesse risorse:
  - Stessa connessione di rete
  - Stesso storage
  - Stessi nodi: 3 nodi ciascuno con 32 CPU logiche (16 fisiche) - 128 GB RAM - 1 Gb/s @ T2_LNL_PD