Attività di calcolo parallelo in LHCb PD

S. Gallorini, D. Lucchesi, A. Gianelle, S. Amerio, M. Corvo
Università e INFN Padova

Padova - 26/09/2016

1

Introduzione

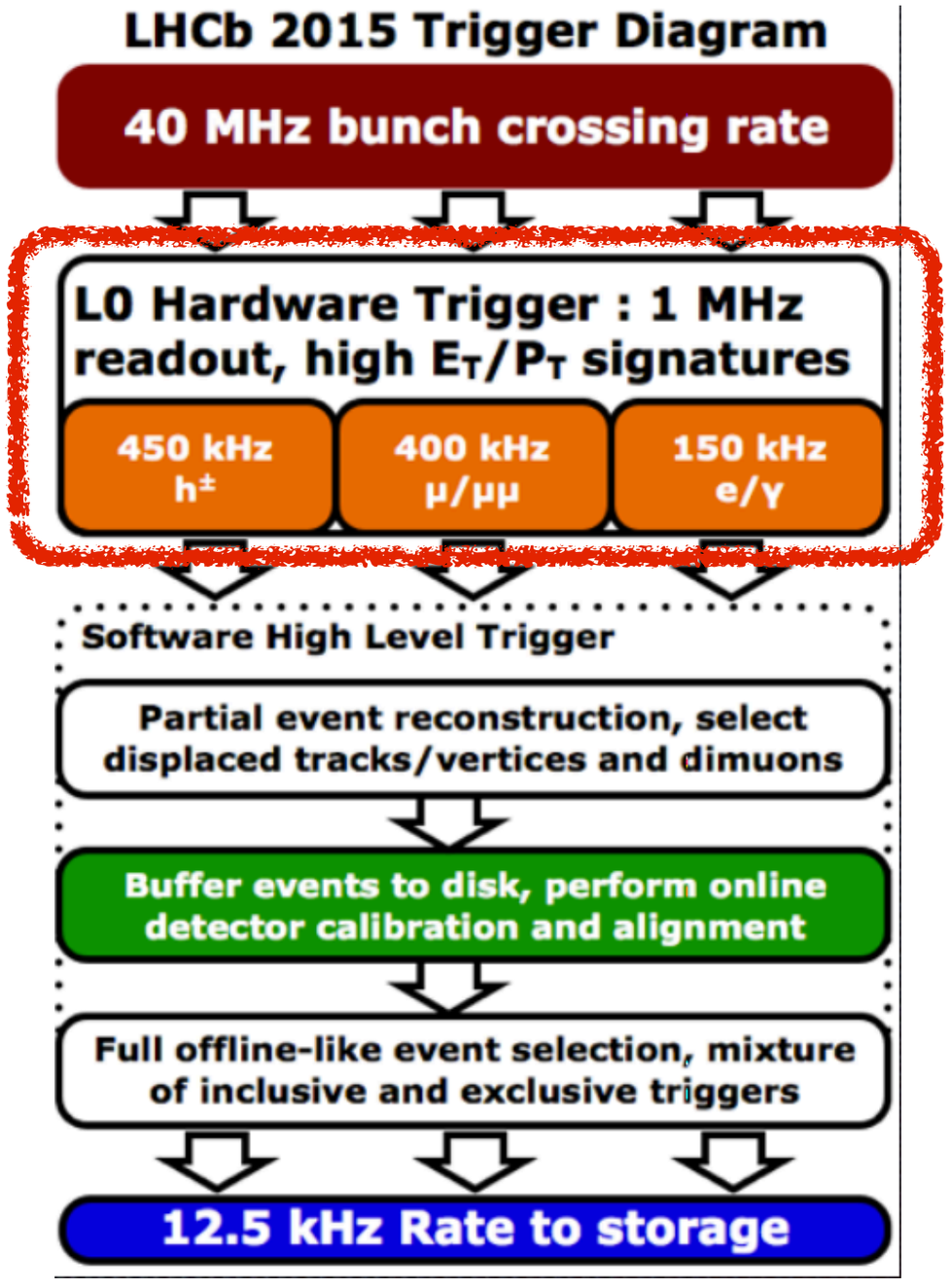
- Con l'aumento di luminosità di LHC previsto per il Run3, è indispensabile un uso più efficiente delle risorse di calcolo per riuscire a processare gli eventi (sia online che offline)

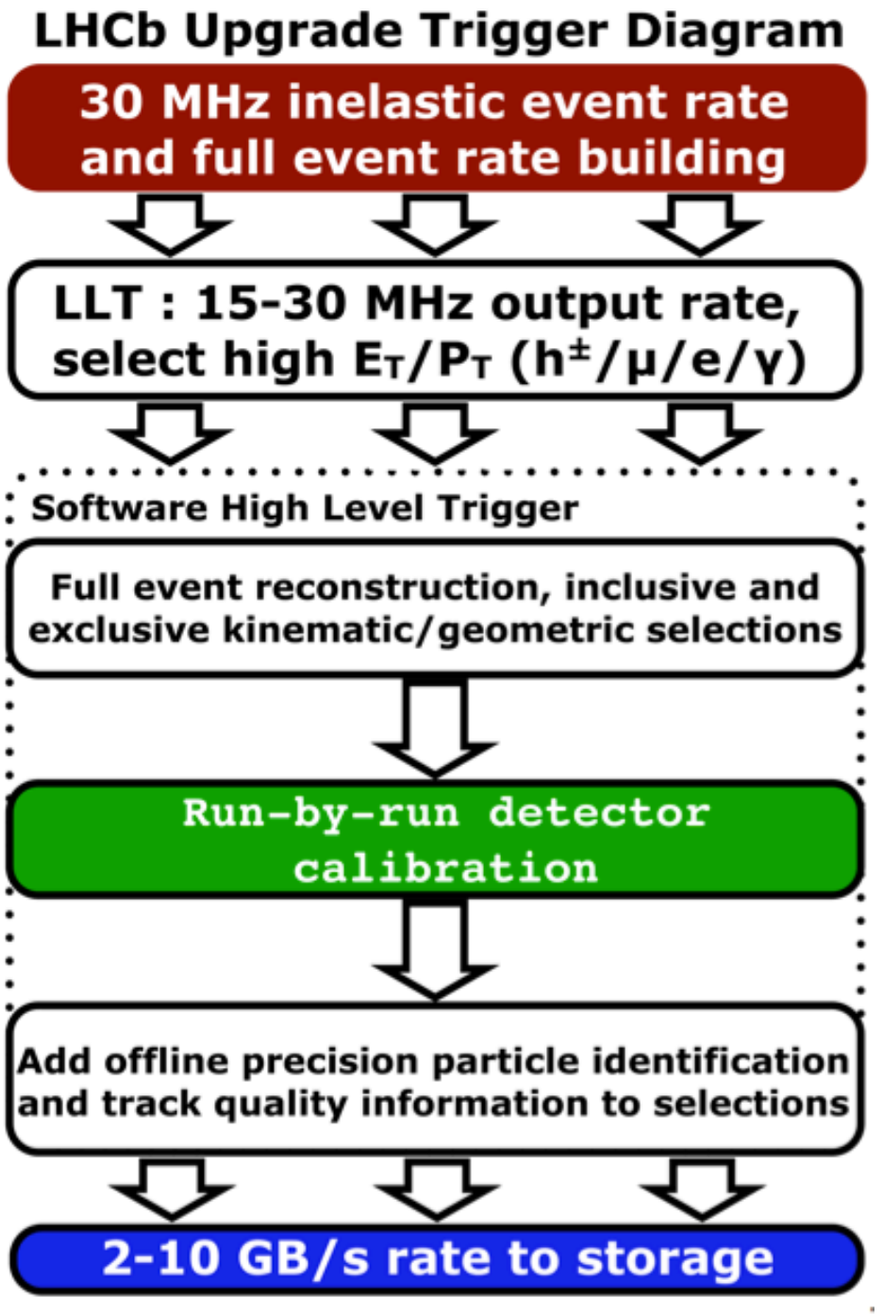
- Architetture many-cores, come ad es. GPGPU, possono essere una valida alternativa alle CPU:
  - Per selezioni di trigger più raffinate e vicine all'offline
  - Per ottimizzare il rapporto costo/performance della farm di trigger
  - Per mitigare il rischio dovuto a possibli deviazioni dalla legge di Moore

2

Il trigger di LHCb upgrade

Full detector readout a 40MHz e full software trigger





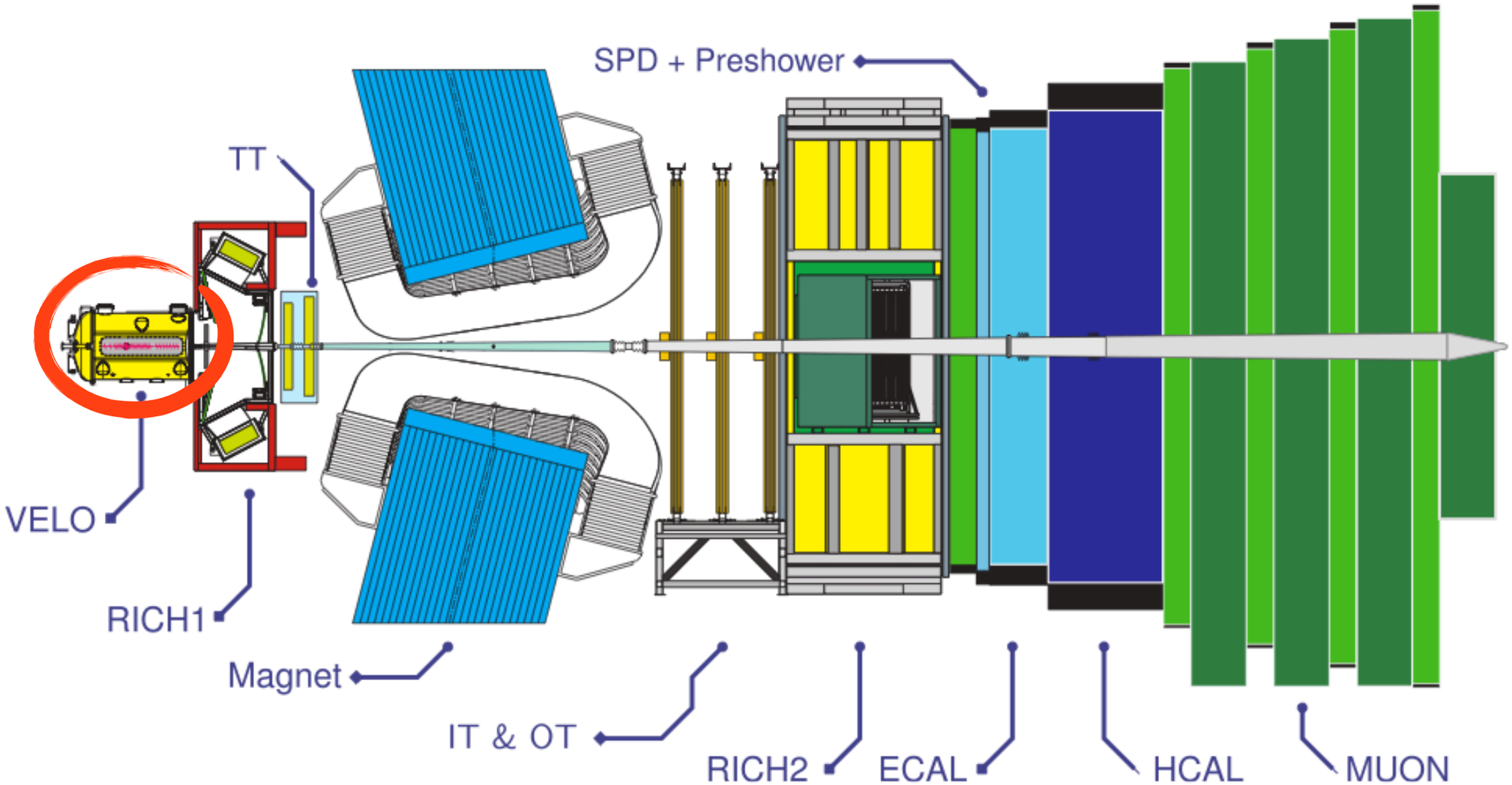
3

Attività many-cores a LHCb Padova

- **Goal**
  - Studiare la fattibilità dell'uso di GPU commerciali (o sistemi eterogenei, ...) nel trigger per l'upgrade di LHCb
  - I Run2 fornisce un opportunità unica per testare le performances delle GPU e la loro integrazione nel sistema online
  - Per far ciò abbiamo svilupato algoritmi di tracking su GPU per il detector attuale
- **Due algoritmi implementati:**
  1. Ricostuzione delle tracce nel VELO ("FastVelo")
  2. Ricostruzione delle tracce nelle T-stations ("Automa")

4

Tracking del VELO



5

Tracking del VELO su GPU (I)

- L'algoritmo originale per il tracking del VELO ("FastVelo") è stato adattato per sfruttare al meglio l'architettura delle GPU (GPU NVidia)

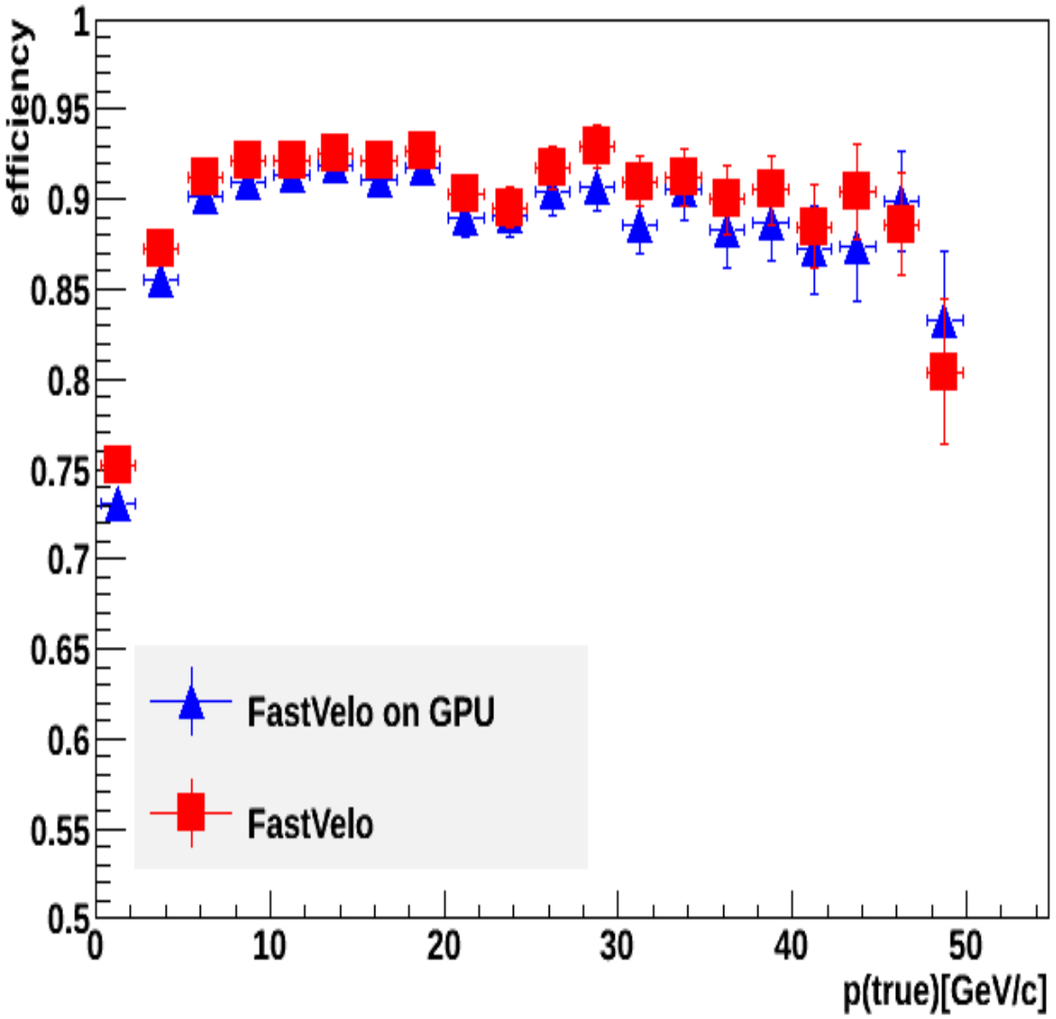
- Per fare ciò, sia la logica che le strutture dati dell'algoritmo sono state in parte modificate

- Le performances (efficienze e tempi di esecuzione) sono state confrontate con l'algoritmo originale (single/multiple CPU cores)
  - GPU: NVidia Titan (14 SMX, 192 CUDA cores/SM, 6GB di memoria)
  - CPU: **single core** (Intel i7, 3.40 GHz, nello stesso PC che ospita la GPU)

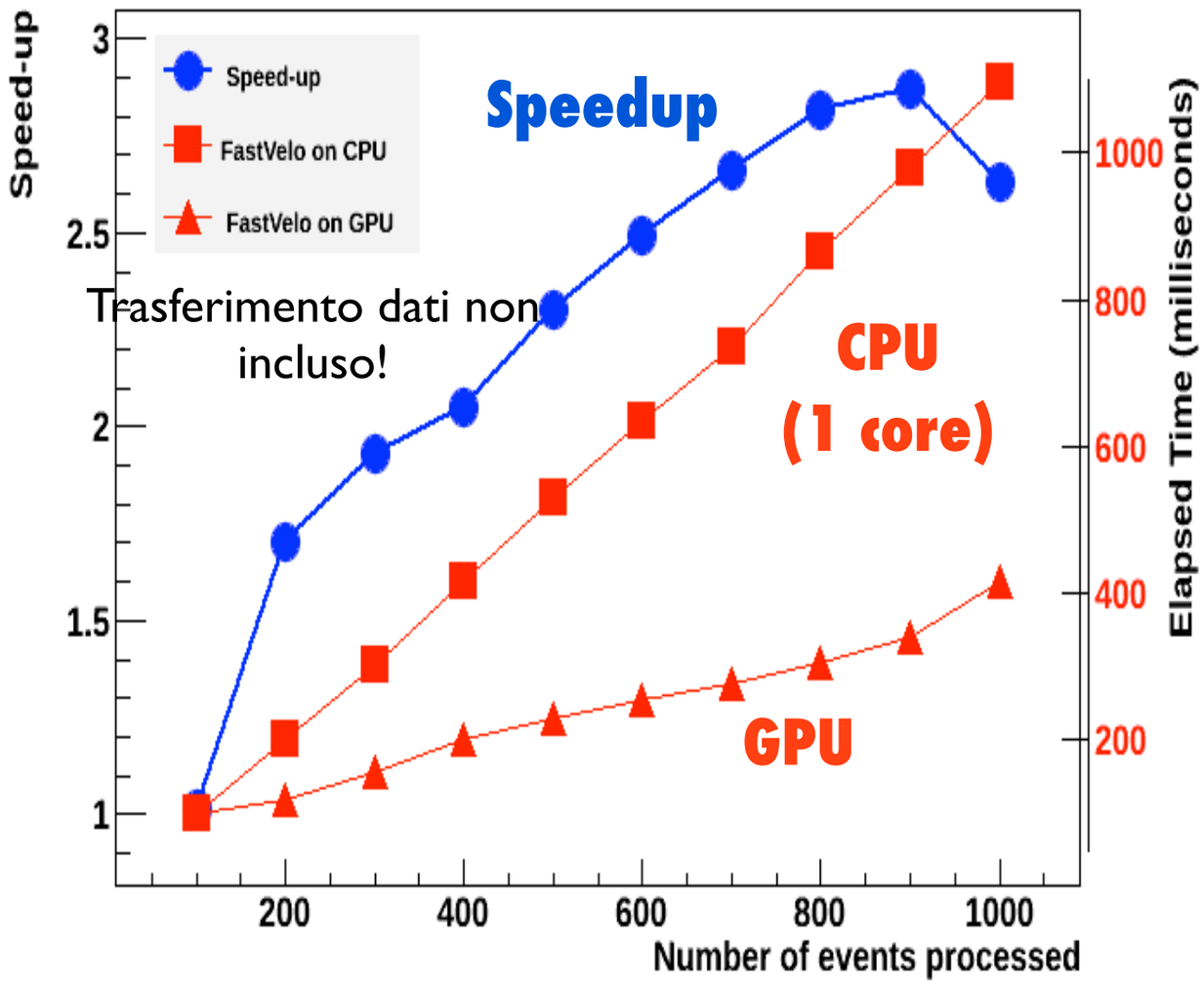
6

Tracking del VELO su GPU (2)

Efficenza vs P(true)



Timing performances

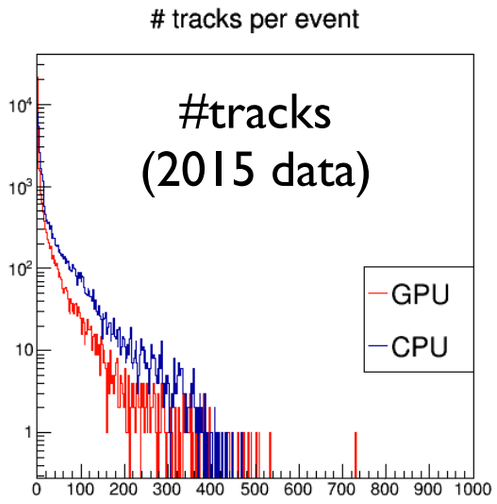


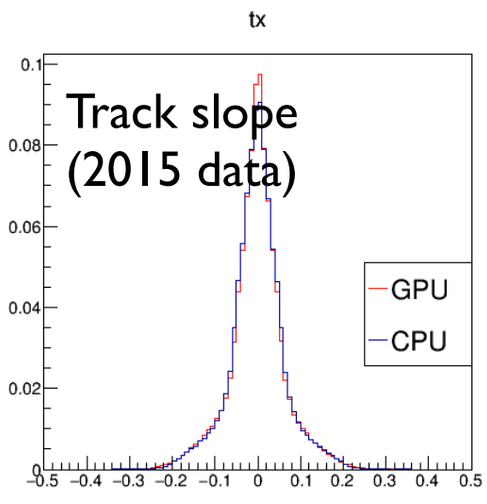
Performances compatibili con l'algoritmo originale (ottimizzato)

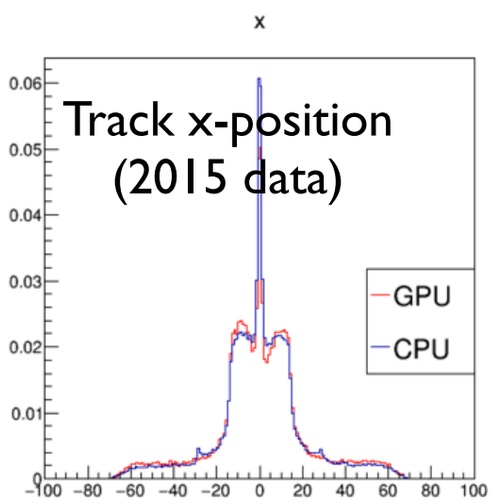
7

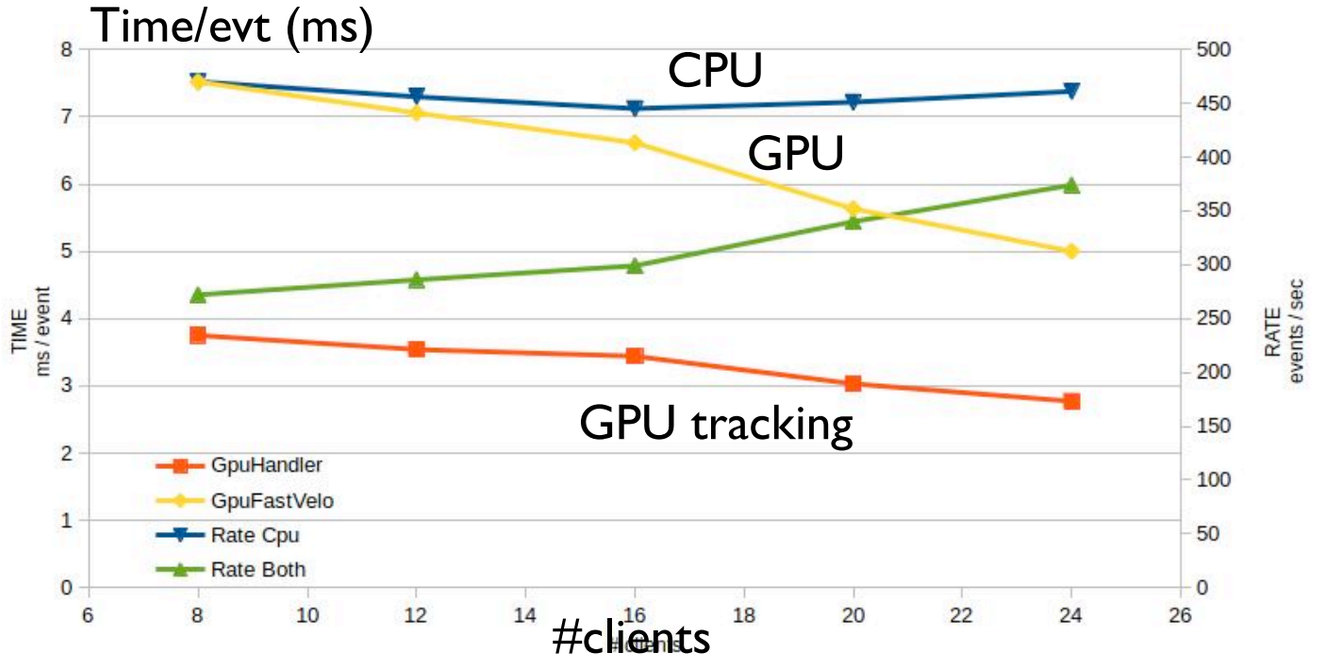
GPU testbed

- Con l'aiuto dell'online group di LHCb siamo riusciti ad installare un PC con una GPU nella monitoring farm
- Questo ci ha permesso di testare il nostro algoritmo in presa dati (seppur a rate moderato) e capire alcune problematiche legate all'integrazione GPU - online FW
- Ogni nodo dell'online farm agisce da client il quale manda l'evento al server GPU (nel nostro caso client = server e #clients = #logical cores)

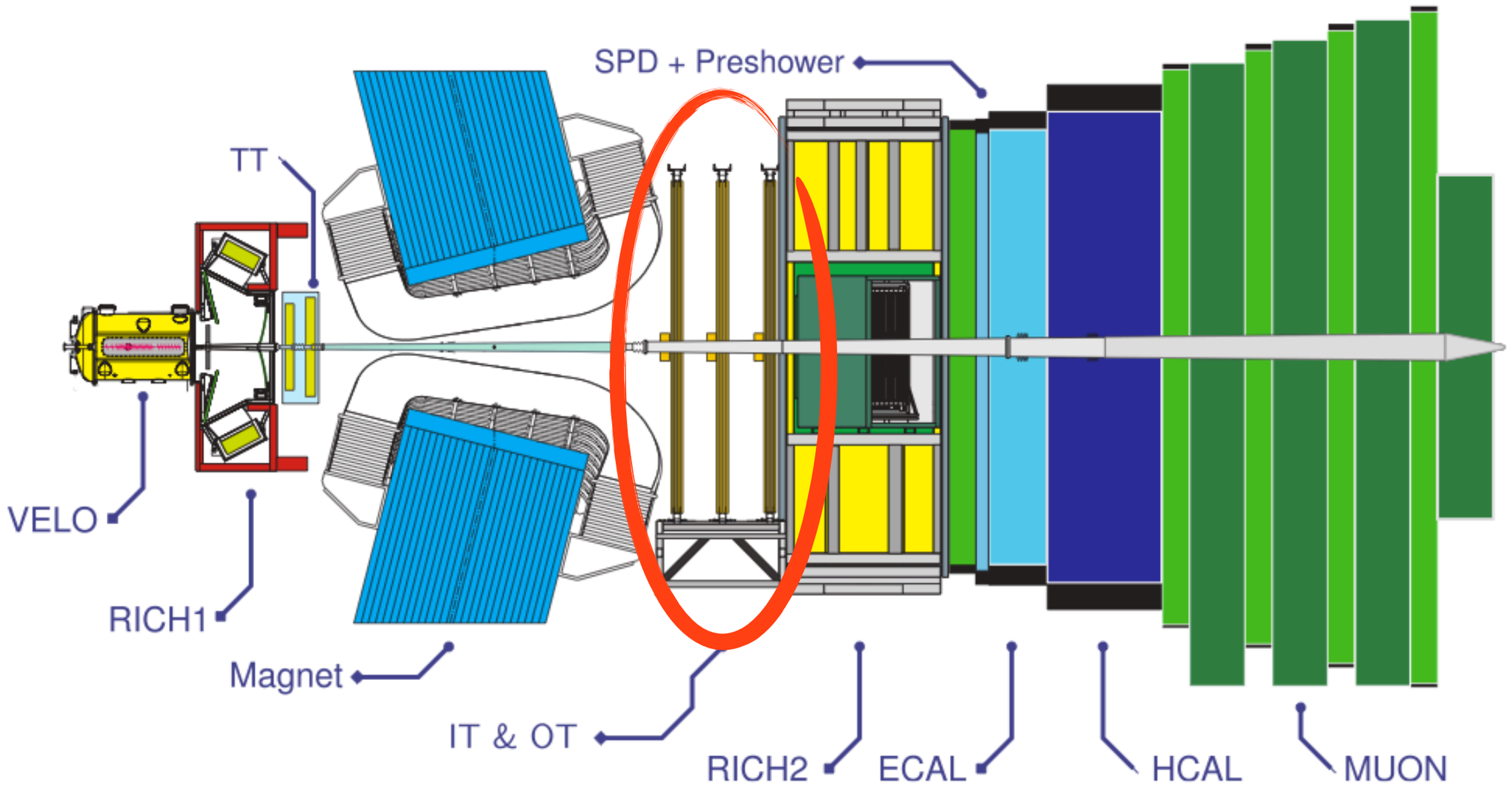








Tracking delle T-stations su GPU (I)



9

Tracking delle T-stations su GPU (2)

- Attualmente, il tracking delle T-stations è responsabile del ~30% del tempo totale dell'HLT2 (anche nell'upgrade)

- Utilizzare il codice esistente in CPU adattandolo a sistemi many-cores è spesso molto complicato (uso estensivo di std library, AoS anzichè SoA, ...)

- Per il tracking delle T-stations abbiamo deciso di riscrivere l'algoritmo da zero, in modo da avere un codice il più possibile "embrassingly parallel"

- L'approccio è ispirato all' Automa Cellulare

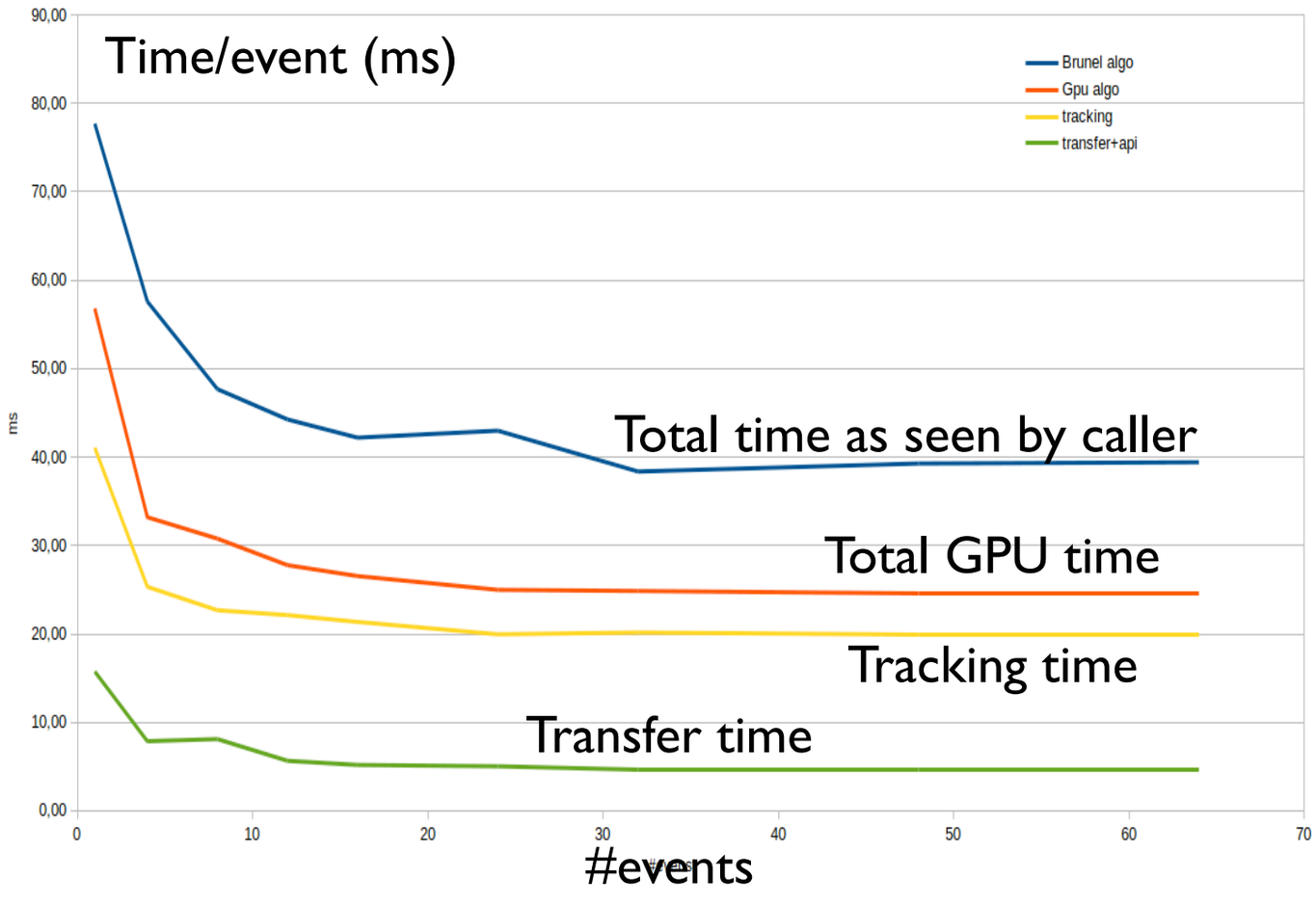
10

Automata

- Efficienze di tracking abbastanza simili all'algo. ufficiale di tracking
- Algoritmo già riscritto su GPU
- La maggior parte del tempo è speso per fittare le tracce!

Automata (1000 evts)		Clones	Hit purity	Hit eff
all long	69.5%	0.1%	96.3%	86.9%
long, p > 5 GeV	63.1%	0.0%	96.1%	88.9%
all long, B daughters	73.4%	0.1%	96.6%	88.7%
long B daughters, p > 5 GeV	71.0%	0.0%	96.6%	89.8%
all long K_S/Lambda	74.6%	0.0%	96.4%	85.0%
Ghosts	34.1%			

PatSeeding (1000 evts, v=1.6,		Clones	Hit purity	Hit eff
all long	73.9%	0.0%	95.1%	94.1%
long, p > 5 GeV	66.9%	0.0%	94.9%	95.4%
all long, B daughters	78.1%	0.0%	95.6%	95.5%
long B daughters, p > 5 GeV	75.6%	0.0%	95.5%	96.1%
all long K_S/Lambda	78.0%	0.0%	95.1%	92.7%
Ghosts	22.9%			



Conclusioni

- In conclusione, GPU e altri acceleratori possono fornire un valida alternativa alle CPU (anche in termini economici) in applicazioni real-time come il trigger

- Ottenere speedups molto elevati mantenendo le **stesse performances** fisiche è pero' tutt'altro che banale

    - Comunque il metro di giudizio è il throughput/costo HW!

- Il Run2 fornisce un opportunità unica per testare le performances delle GPU e la loro integrazione nel sistema online

- Altri algoritmi, ad es. la ricostruzione del RICH, sono buoni candidati per architetture parallele e verranno esplorati in futuro

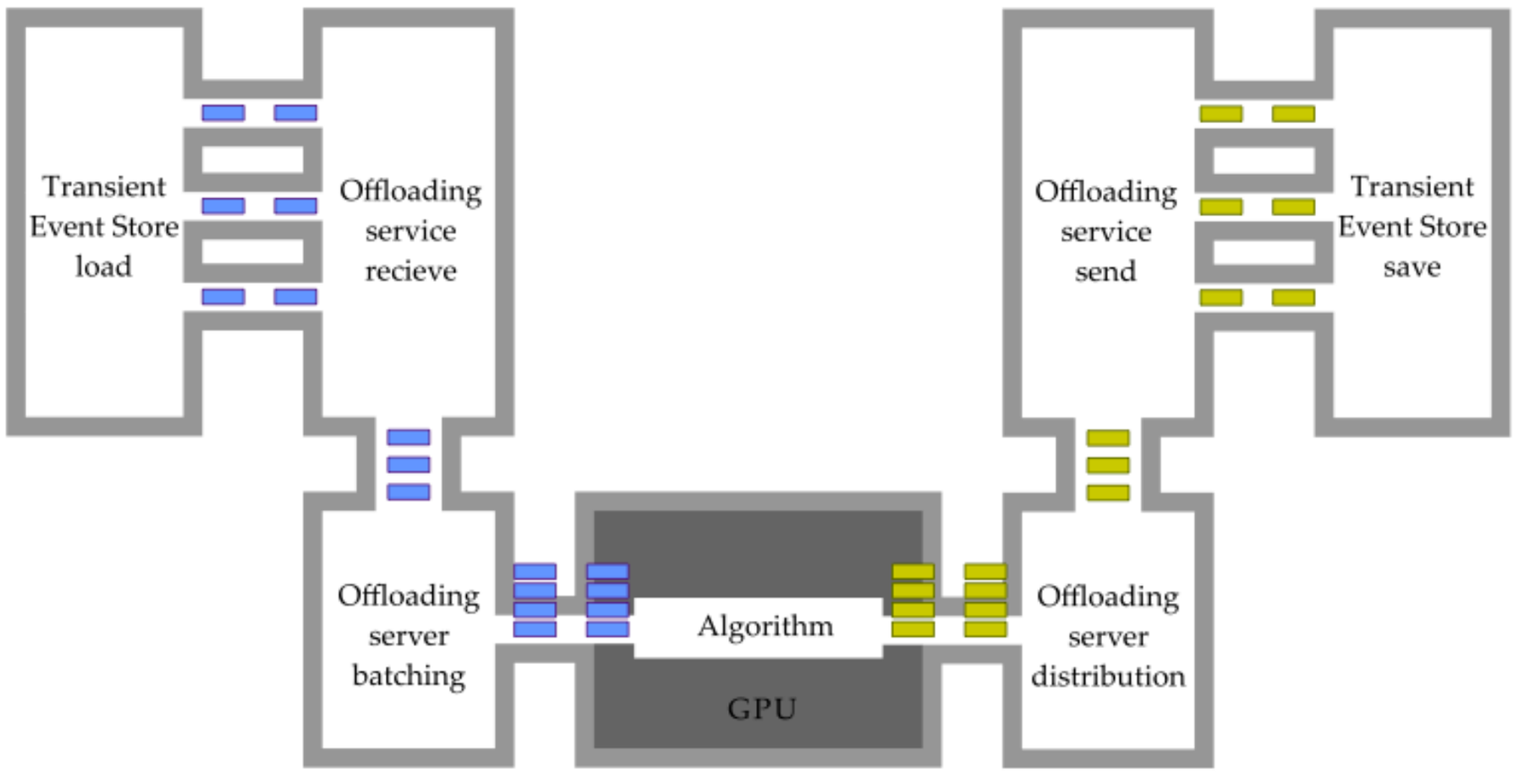
12

Back-up slides

13

Integrazione nel FW (I)

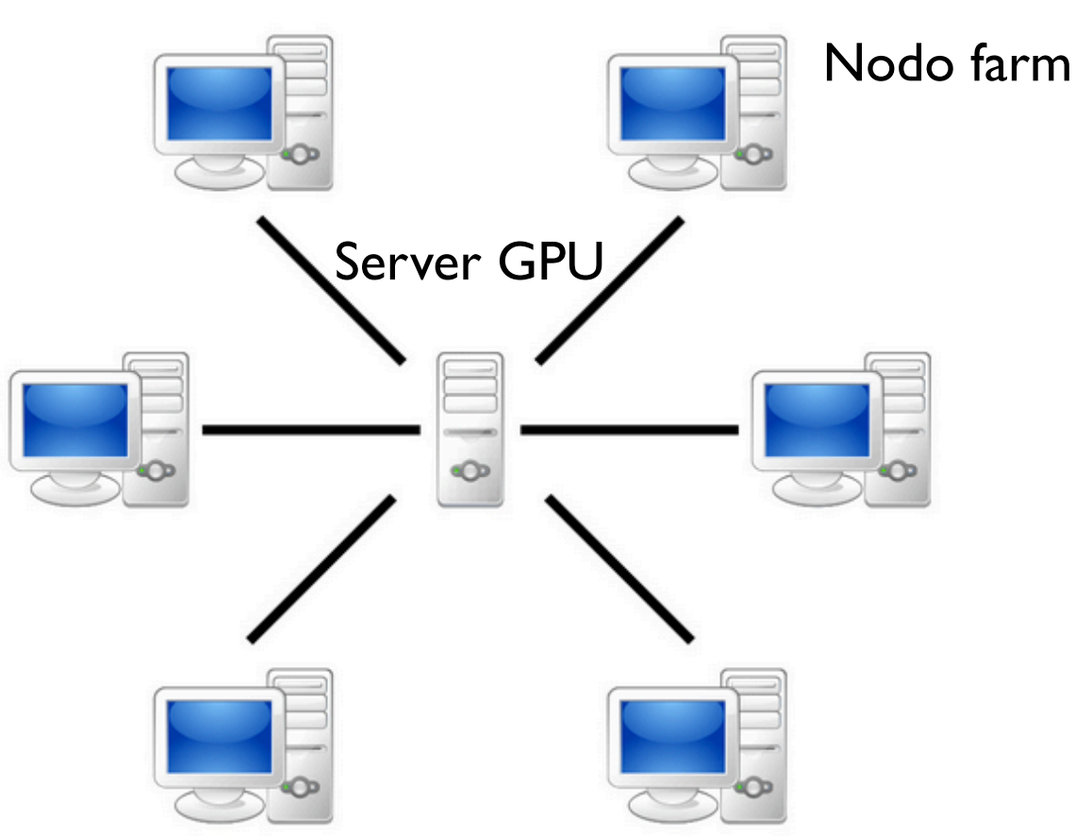
- Un aspetto critico per l'utilizzo di acceleratori è l'integrazione con il framework software del trigger ("Gaudi")
- In LHCb stiamo sviluppando un sistema di "offloading" usando un sistema client-server ("GPUManager")



14

Integrazione nel FW (2)

- Ogni nodo dell'online farm agisce da client il quale manda l'evento al server GPU (es. le hits di un rivelatore)
- Il server riceve gli eventi e le accumula in un buffer
- Il bunch di eventi viene processato in GPU e il risultato rispedito a ciascun client



15

Integrazione nel FW (3)

Esempio di utilizzo:

I dati input/output devono essere convertiti in SoA dal client:

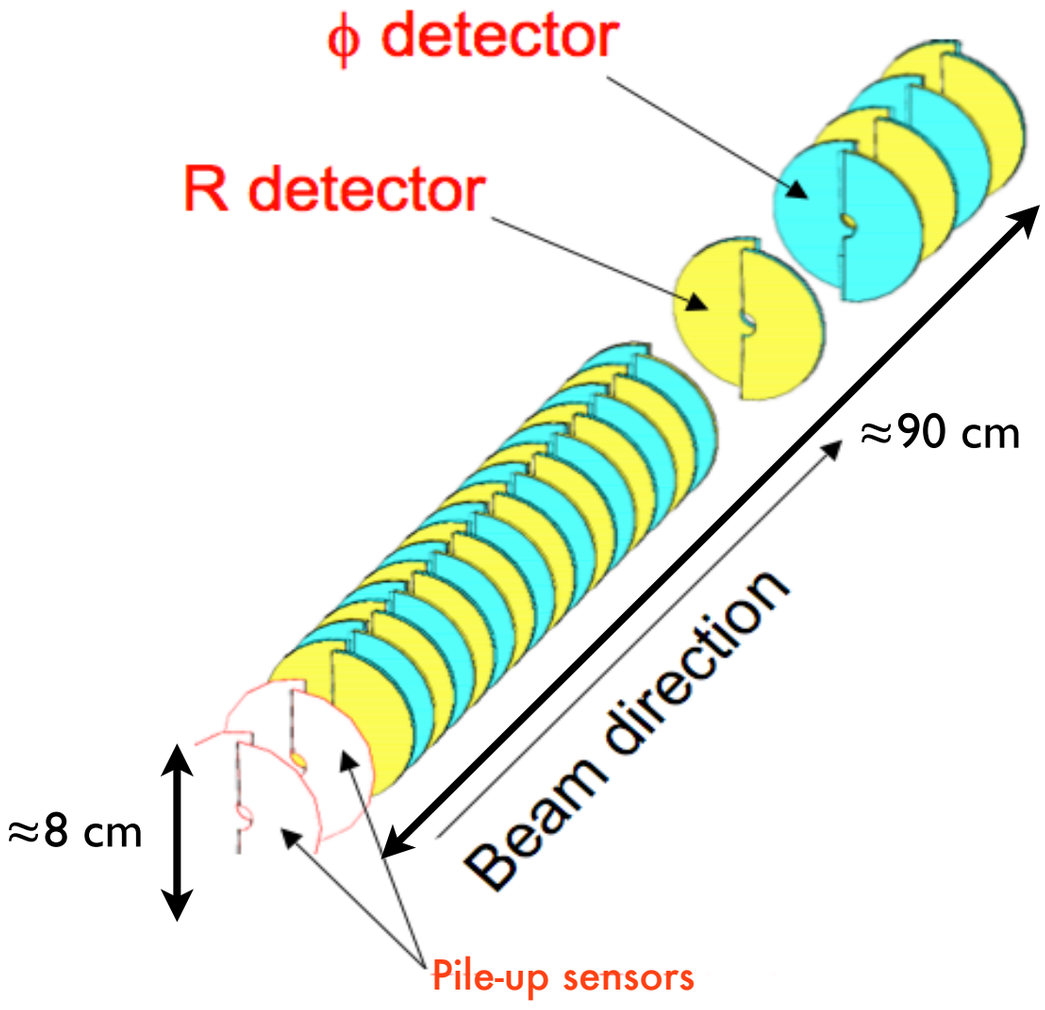
```
// compute total size and allocate memory
uint8_t * dst = (uint8_t *)&buffer[0];
const size_t noSensorsSize       = sizeof(int);
const size_t noHitsSize          = sizeof(int);
const size_t sensorZsSize        = m_event.sensorZs.size() * sizeof(int);
const size_t sensorHitStartsSize = m_event.sensorHitStarts.size() * sizeof(int);
const size_t sensorHitsNumsSize  = m_event.sensorHitsNums.size() * sizeof(int);
const size_t hitIDsSize          = m_event.hitIDs.size() * sizeof(int);
const size_t hitXsSize           = m_event.hitXs.size() * sizeof(float);
const size_t hitYsSize           = m_event.hitYs.size() * sizeof(float);
const size_t hitZsSize           = m_event.hitZs.size() * sizeof(float);

// serialize container contents
dst = copy(m_event.sensorZs,         dst);
dst = copy(m_event.sensorHitStarts, dst);
dst = copy(m_event.sensorHitsNums,  dst);
dst = copy(m_event.hitIDs,           dst);
dst = copy(m_event.hitXs,            dst);
dst = copy(m_event.hitYs,            dst);
dst = copy(m_event.hitZs,            dst);
```

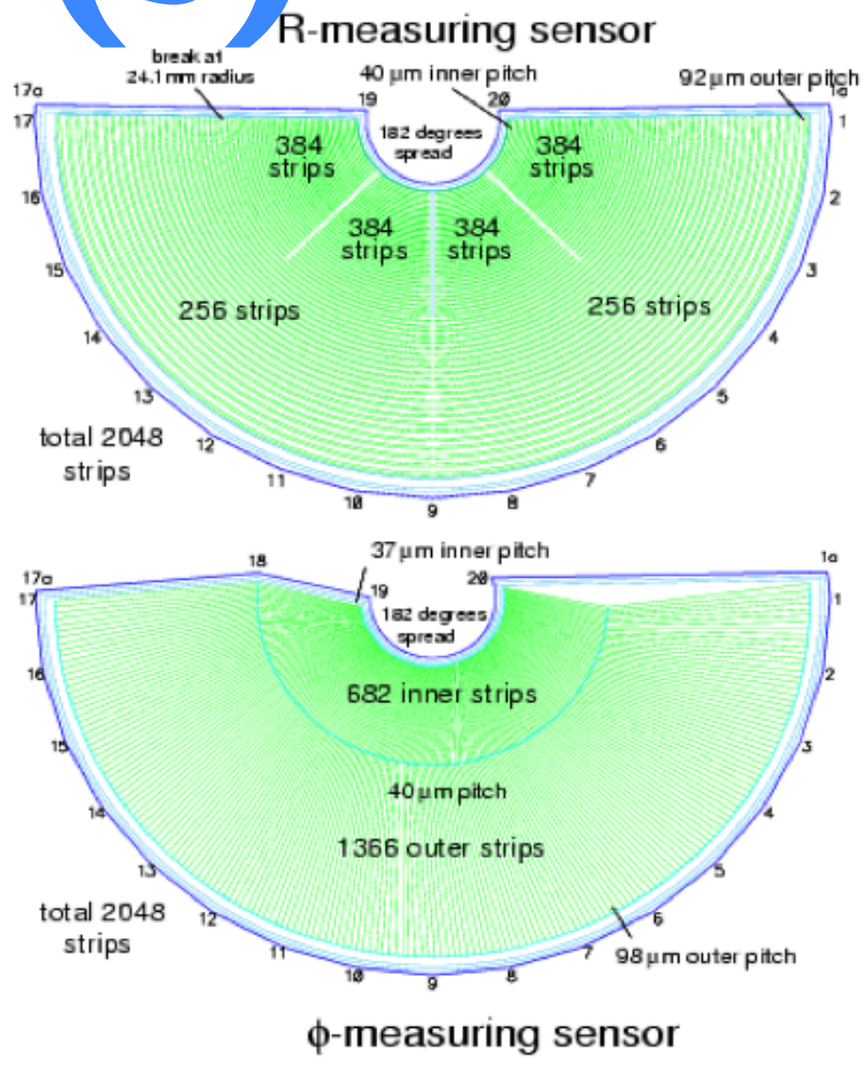
Il client puo' quindi sottomettere i dati al server specificando il kernel:

```
gpuService->submitData("PrPixelCudaHandler", &m_serializedEvent[0], m_serializedEvent.size());
```


16

Tracking del VELO su GPU (3)





- Il VELO è un detector a strips di silicio, situato vicino alla regione di interazione
- R-ϕ layout, 21 stazioni, ognuna con 2 sensori R e 2 sensori ϕ

17

Tracking del VELO su GPU (4)

- La ricostruzione delle tracce nel VELO è suddivisa in due parti:

1. **RZ tracking:** ricerca di tracce nel piano R-Z (solo sensori R)
2. **Space tracking:** le tracce spaziali sono ricostruite a partire dalle tracce in RZ aggiungendo l'informazione delle hits nei sensori ϕ

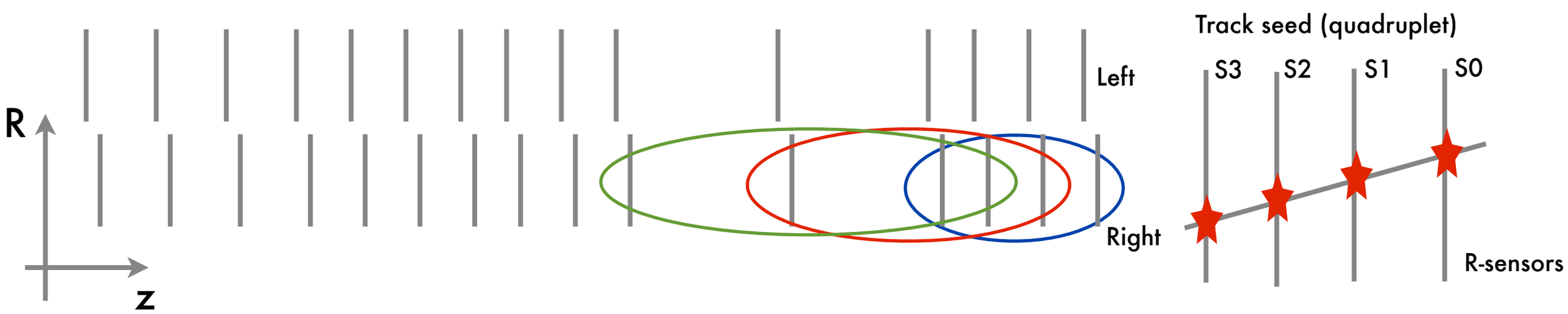
- **Strategia di parallelizzazione:**

1. Processare più eventi in parallelo (ovvio)
2. Ricerca in parallello delle tracce RZ (seeding + track following)
3. Space tracking in parallelo per ogni traccia RZ (più dettagli in backup)

18

Tracking del VELO su GPU (5)

- **RZ tracking:**
- Only R-sensors are used
- The algorithm looks for quadruplets of hits in four contiguous R-sensors (seed) on both halves.
  - Each thread works on a set of four contiguous R-sensors and find all quadruplets.
- Then each quadruplet is extended in parallel as much as possible adding the remaining R-sensors



19

Tracking del VELO su GPU (5)

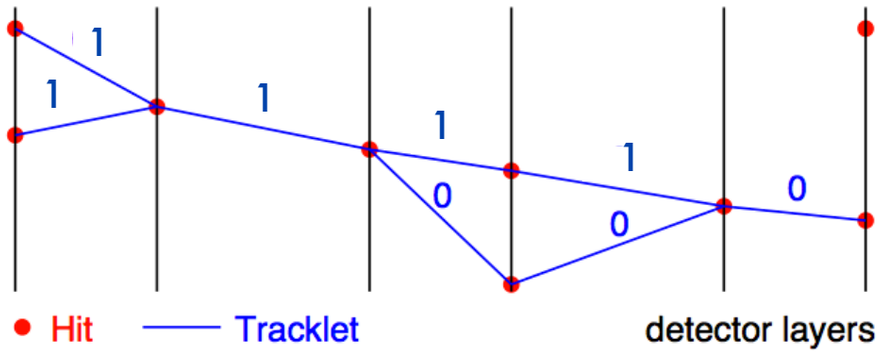
- **Space tracking:**
- Add hits on ϕ-sensors
- Each RZ track is processed concurrently by assigning a space-tracking algorithm to each thread:
  - Search for a triplet of hits: for each hit in the first two ϕ-sensors, the candidate hit in the third sensor is the one most compatible with predicted position (best χ^2)
  - The track is extended and its parameters are found by minimizing $\Sigma_{points}\ \chi^2$ (linear system solved by substitution)
  - This part is almost a re-writing in CUDA of the original space-tracking code

20

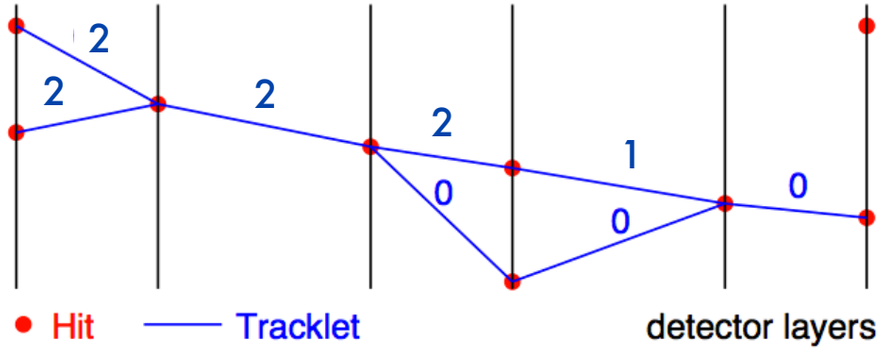
Tracking delle T-stations

- L' Automa Cellulare (AC) procede in 3 passi:
  - Generazione di tutti i segmenti tra layers adiacenti ("le cellule"). Ogni segmento possiede un contatore ("stato della cellula") inizialmente inizializzato a zero
  - Ricerca dei segmenti vicini (segmenti adiacenti compatibili con il modello di propagazione della traccia)
  - Evoluzione (in step temporali). Il contatore (c1) di ogni segmento è incrementato di +1 se uno dei suoi vicini ha il contatore c2 ≥ c1

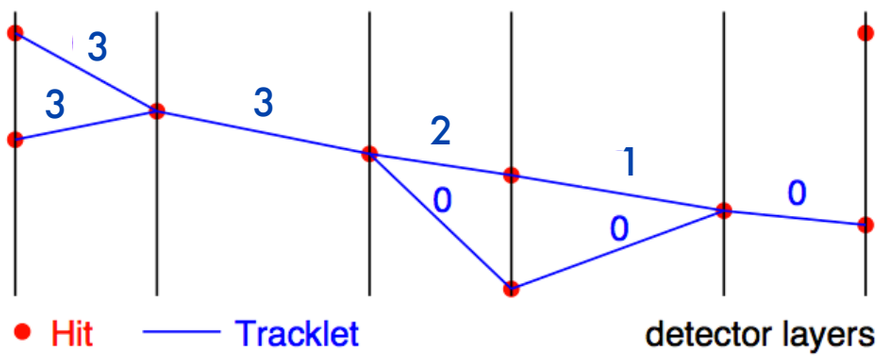
Step1



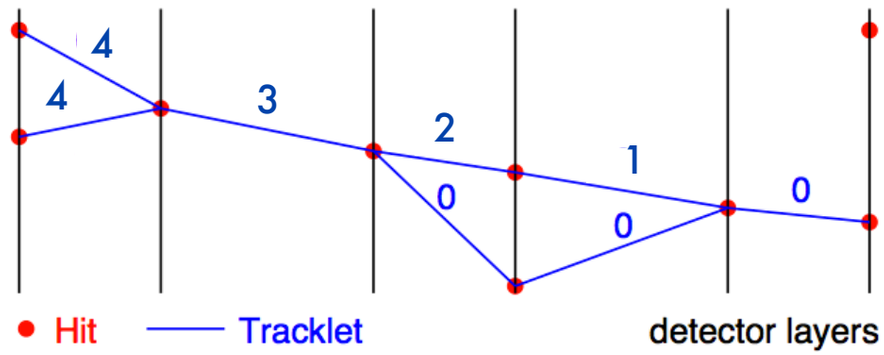
Step2



Step3



Step4



21