

Experimenting with ROOT task-oriented parallelization and Tools

E. Tejedor, D. Piparo, P. Canal, P. Mató

Concurrency Forum

May 20th 2015



1. ROOT Parallelization

1. Overview
2. TTree iteration & I/O Pipeline
3. Parallel histogram filling

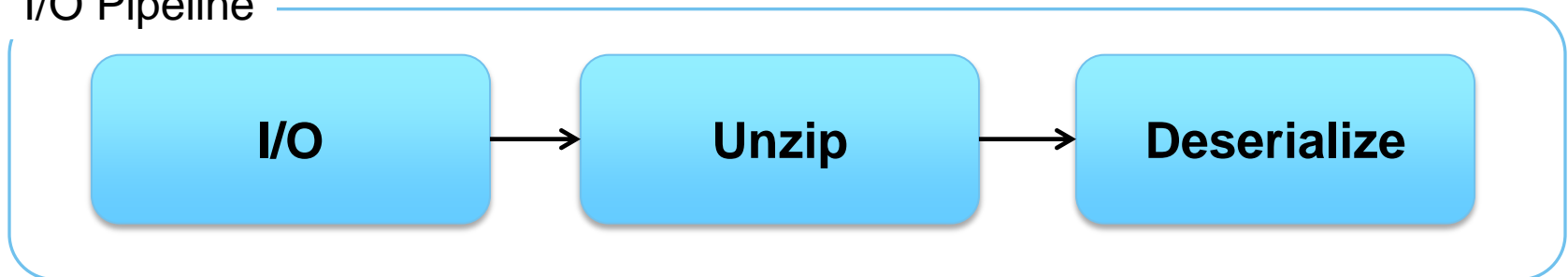
2. Tools

1. VTune
2. IgProf
3. Extrae + Paraver

ROOT Parallelization

- Seek for opportunities to do things in parallel in ROOT
- Prototype solutions for 4 use cases:
 - **I/O Pipeline**
 - Ntuple / **Histogram filling**
 - Ntuple processing (TTreeDraw)
 - Minimization / fitting
- Design the programming model offered to the user

I/O Pipeline



Use case: Sequential code

```
TFile *f = new TFile("tree.root");  
TTree *t1 = (TTree*)f->Get("t1");
```

```
TH1F *hpx = new TH1F("hpx","px distribution",100,-3,3);  
TH2F *hpxpy = new TH2F("hpxpy","py vs px",30,-3,3,30,-3,3);
```

```
Float_t px, py;  
t1->SetBranchAddress("px",&px);  
t1->SetBranchAddress("py",&py);
```

```
for (Long64_t i = 0; i < t1->GetEntries(); i++) {  
    t1->GetEntry(i);
```

1

```
    hpx->Fill(px);  
    hpxpy->Fill(px,py);  
}
```

2

Two problems to solve:

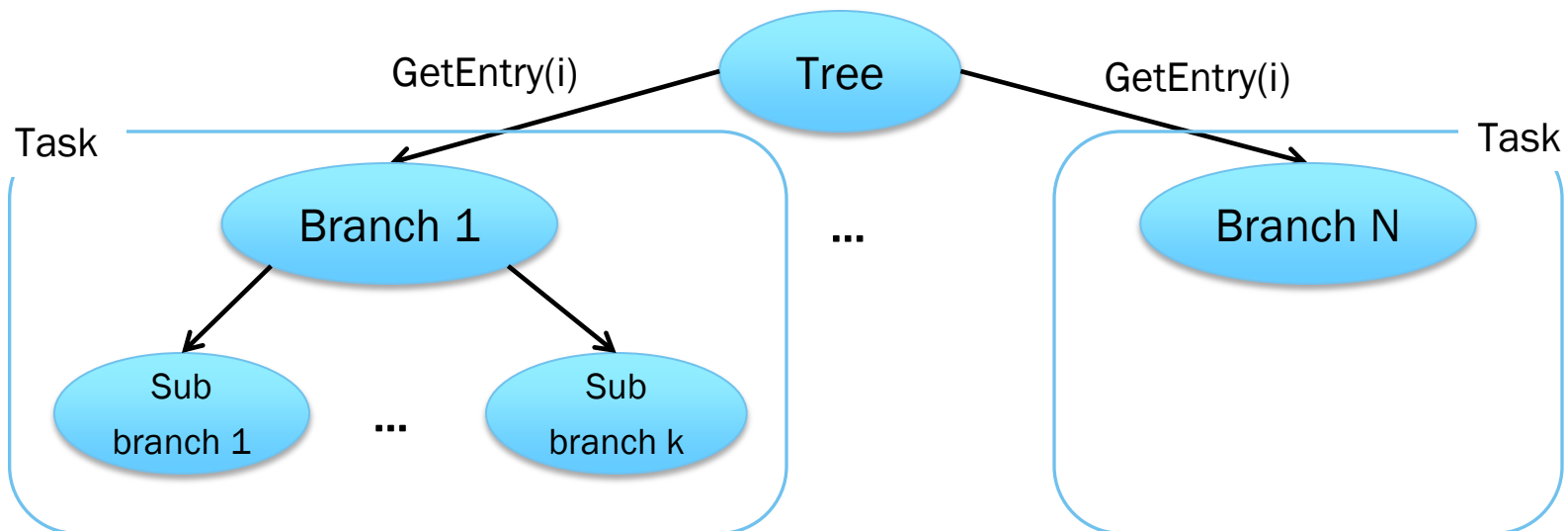
1. How to iterate over a tree
(parallel I/O pipeline)
2. How to fill histograms in parallel

Programming model: Overview

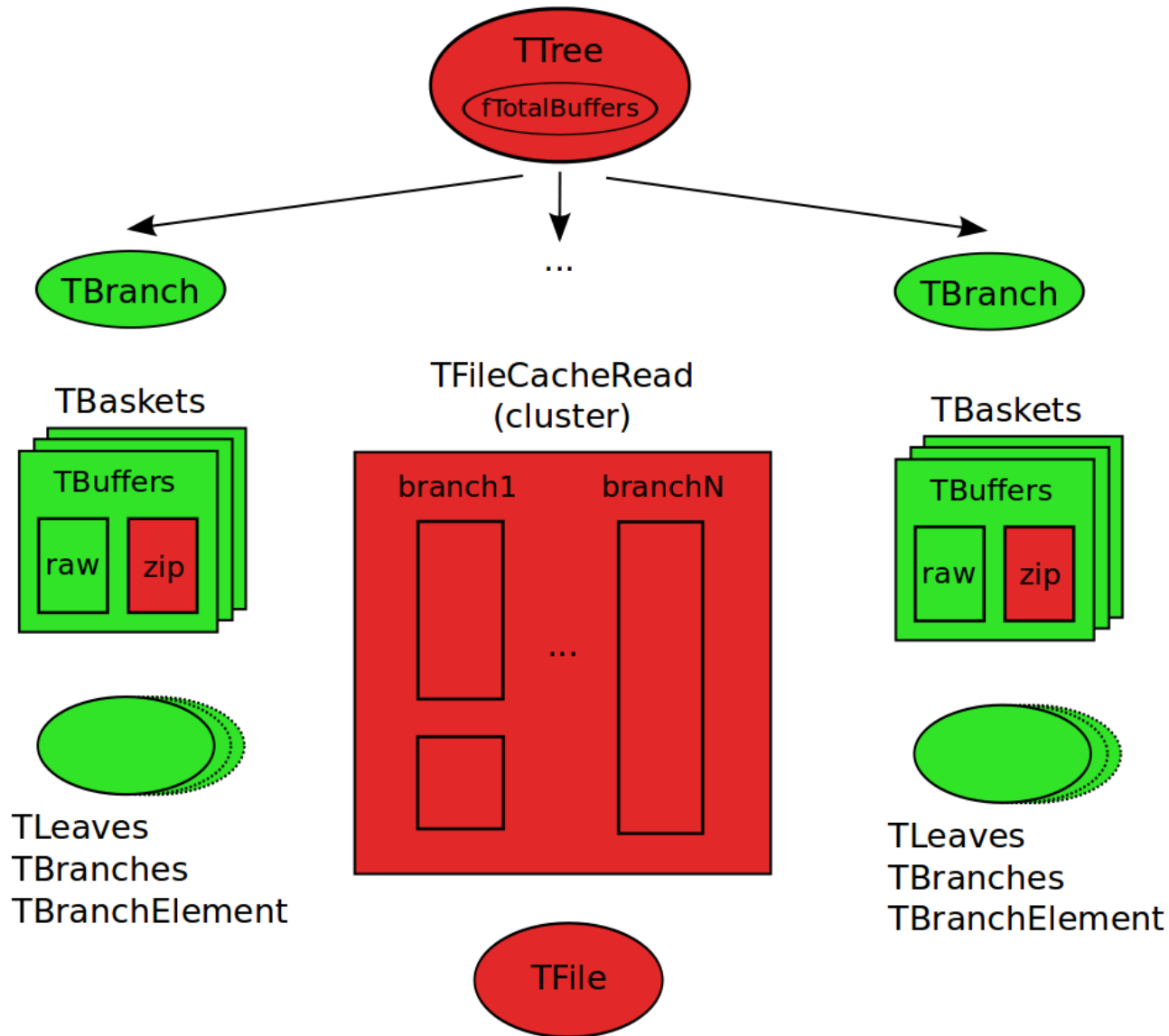
- Make parallelization transparent to the user in some cases
 - e.g. TTreeDraw, minimization
- Otherwise, offer a user-friendly programming model
 - Hide parallelization complexity
 - Provide simple means for concurrency
- TTree iteration
 - I/O pipeline parallelization happens behind the scenes
- Histogram filling
 - Provide a MultiObject mechanism to fill in parallel and merge the partial results

TTree Iter: Parallel Branch Processing

- Use case: iteration of a TTree
- First parallelization approach based on parallel branch processing
 - Goal: reduce GetEntry latency
 - Create one asynchronous task per TBranch::GetEntry
- Protect access to shared objects (tree, cache)

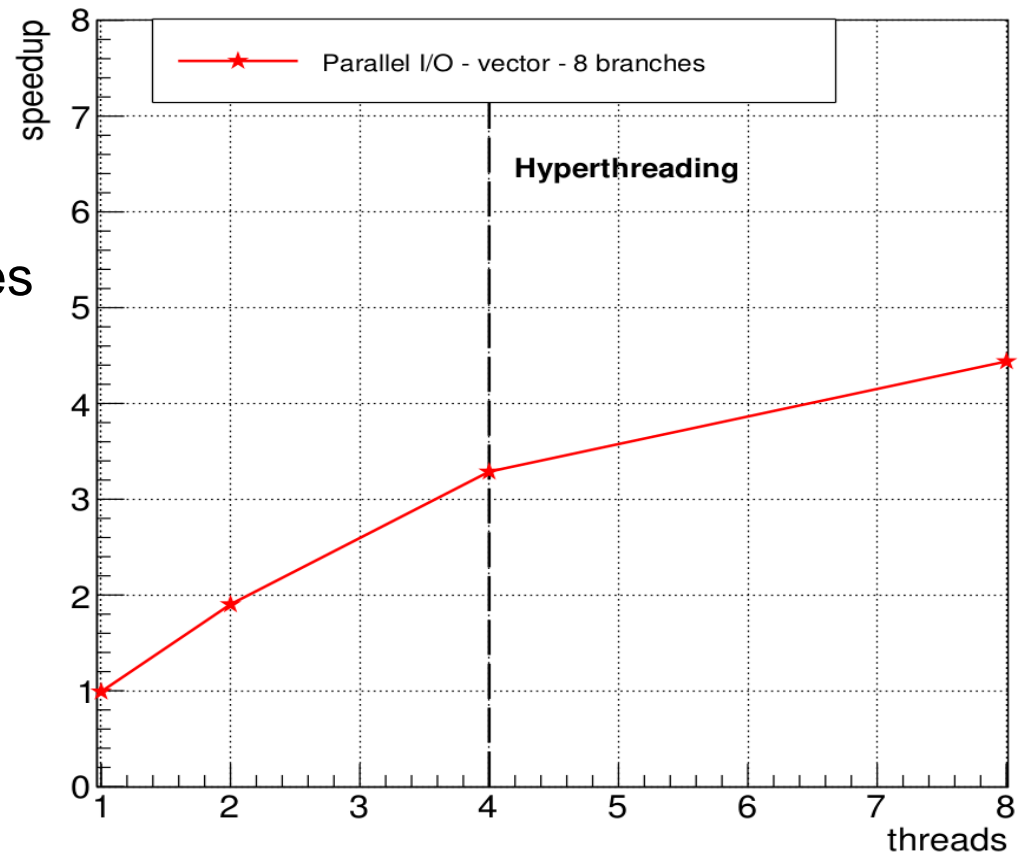


TTree Iter: Parallelization details



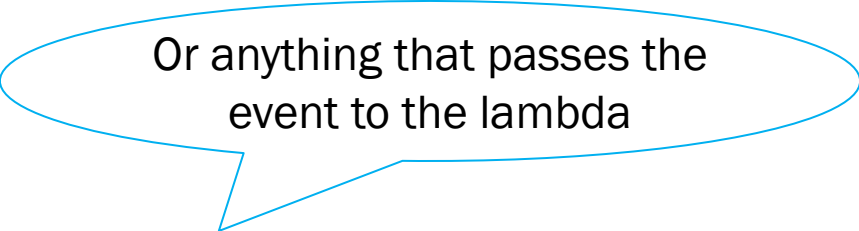
TTree Iter: Preliminary Results

- Preliminary performance results for first PBP prototype
- TTree iteration
 - Complex branches
 - Homogeneous
- Next step: real use case
 - Different types of branches



TTree Iter: Programming Model

- The programmer could provide a lambda that processes a single event
 - Parallel Branch Processing would be hidden

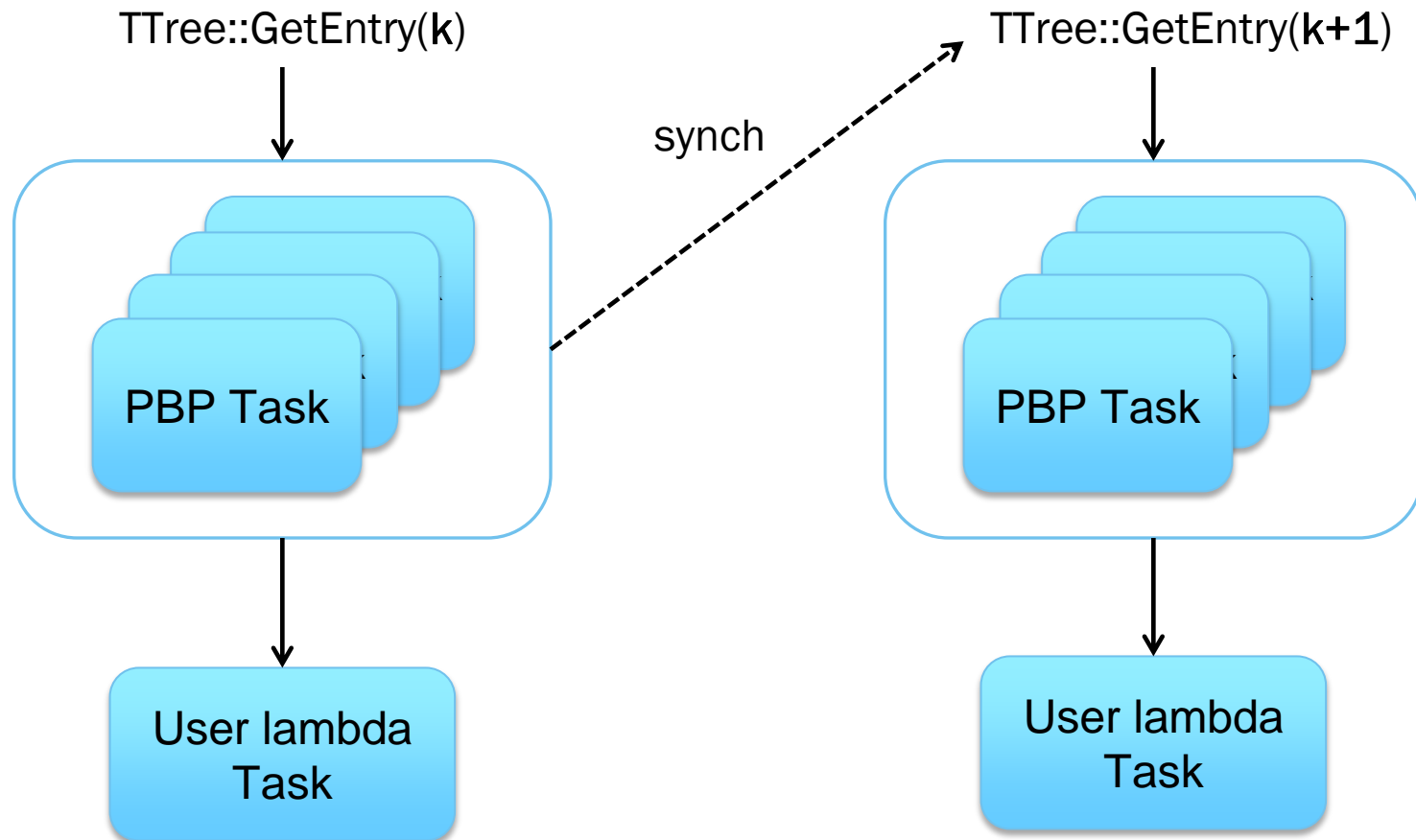


Or anything that passes the event to the lambda

```
tree.process([&](Float_t px, Float_t py) {  
    hpx->Fill(px);  
    hpypy->Fill(px,py);  
});
```

TTree Iter: PBP + User lambda

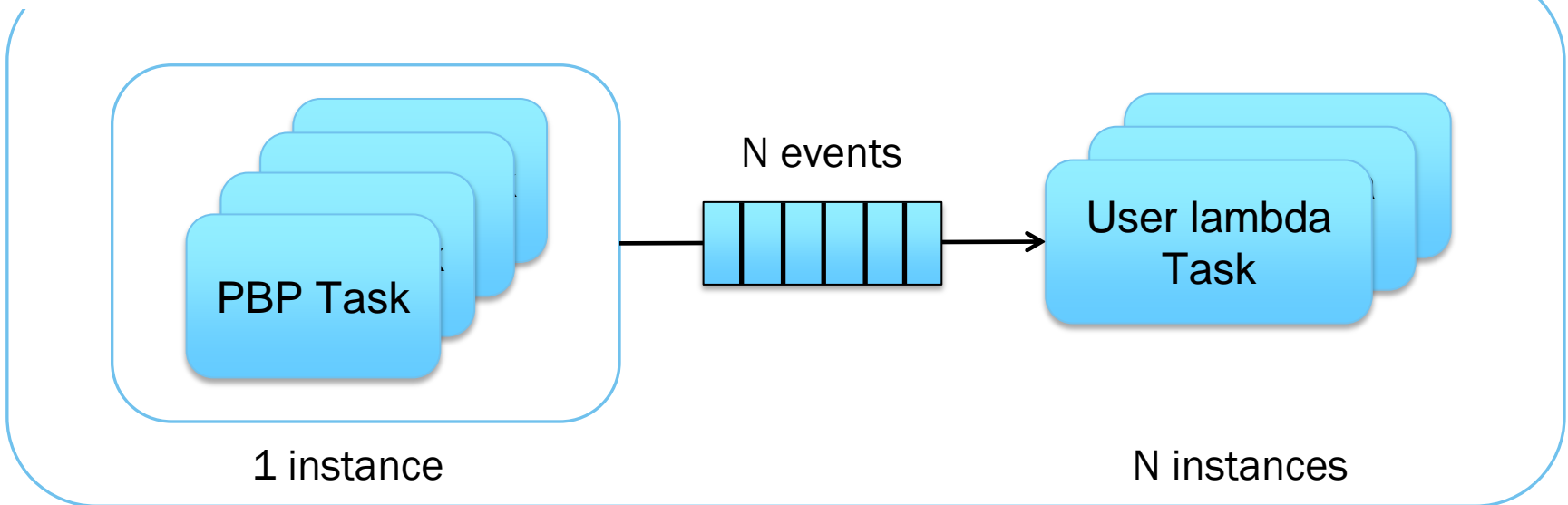
- Concatenate PBP tasks with user lambda tasks
 - Keep results for multiple entries in memory
 - Heterogeneous sub-branches, scheduling, clustering



TTree Iter: PBP + User lambda (II)

- PBP + User lambda could be implemented as a TBB pipeline with two stages
 - PBP stage is serial but creates parallel tasks internally
 - User lambda stage is parallel (limited by N)

TBB pipeline



Histogram filling: Programming Model

- Create a container class for thread-local histograms
- Each thread fills its own partial histogram
- Partial histograms are merged at the end

```
MultiObject<TH1F> hpx(TH1F(...));  
MultiObject<TH2F> hpxpy(TH2F(...));
```

→ Multi-histogram objects

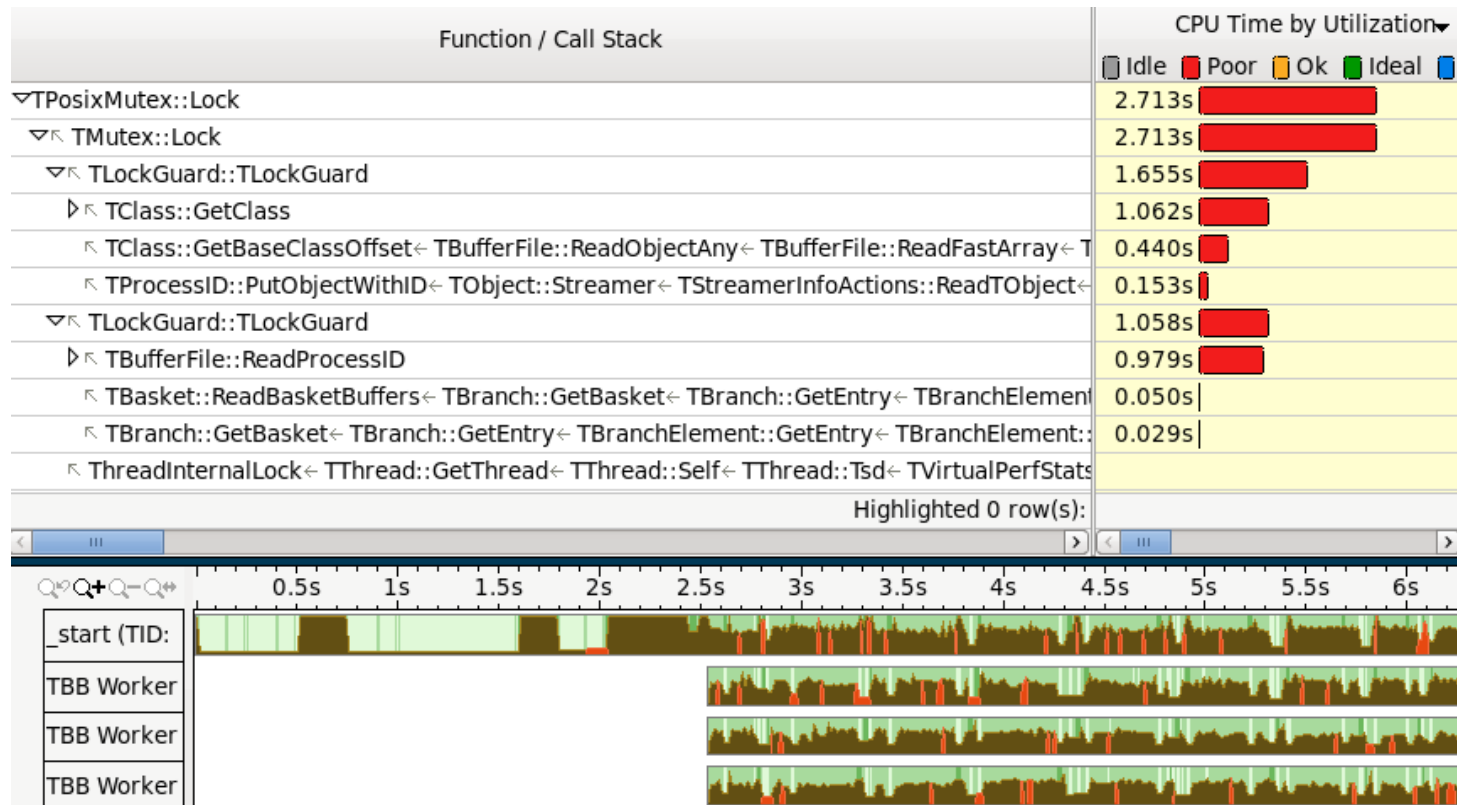
```
t1.process([&](Float_t px, Float_t py) {  
    hpx->Fill(px);  
    hpxpy->Fill(px,py);  
});
```

→ Fill partial histogram

```
TH1F& hpxsum = hpx.sum();  
TH2F& hpxpysum = hpxpy.sum();
```

→ Reduction

- Intel VTune Amplifier
 - Profiler
 - Concurrency analysis, thread display

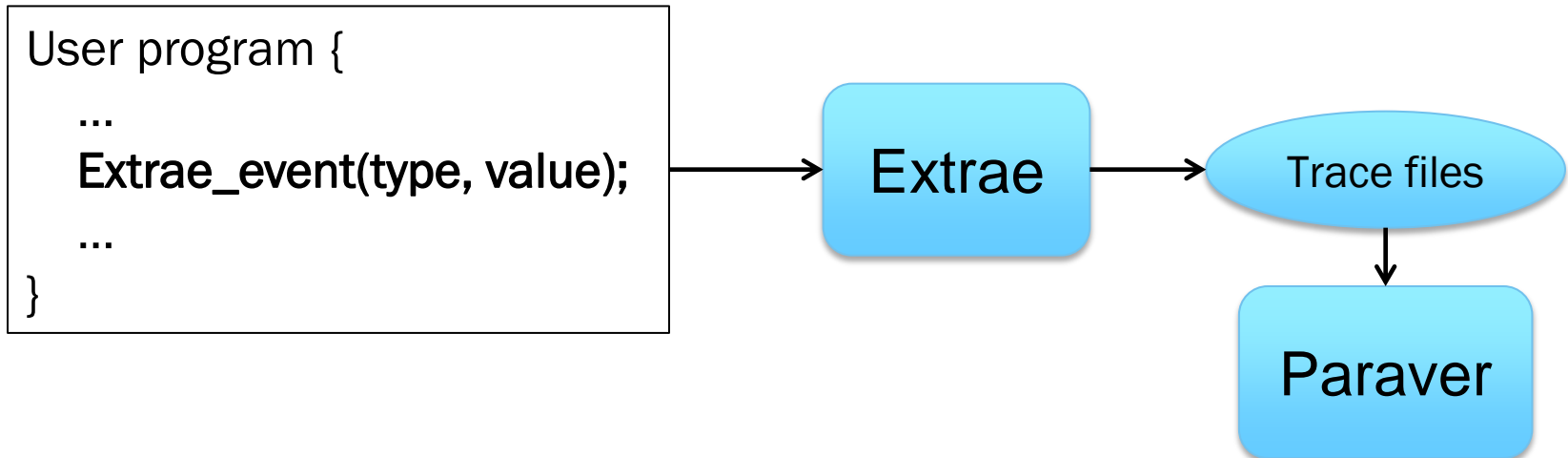


- Profiling tool
 - Double check for VTune
 - Less profiling overhead

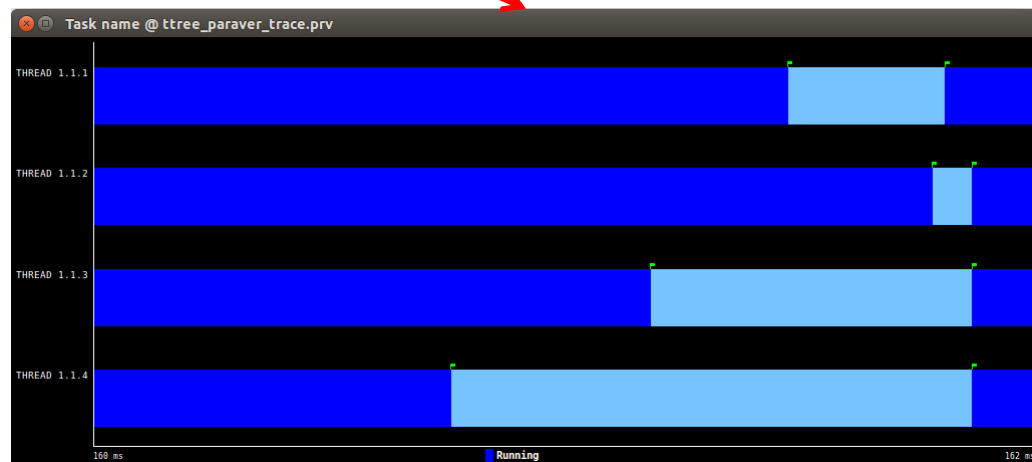
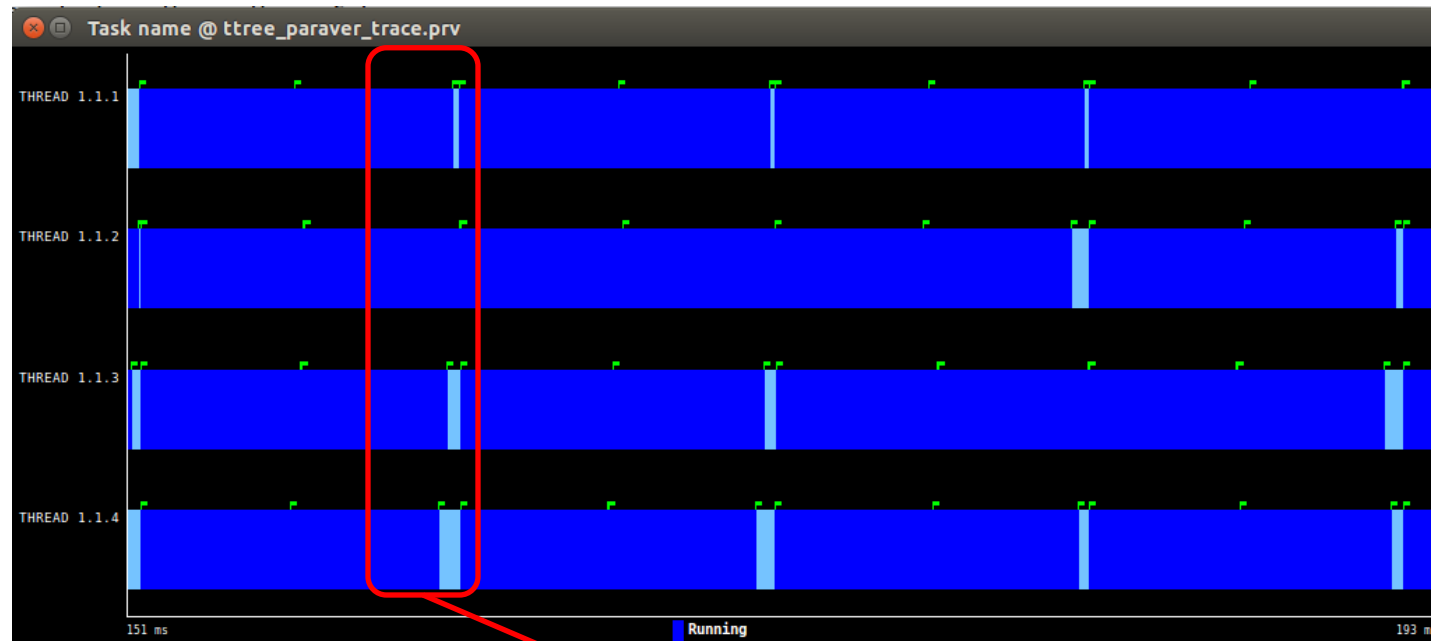
Rank	Total %	Self	Symbol name
<u>21</u>	59.79	37.42	<u>inflate_fast</u>
<u>44</u>	17.34	10.85	<u>frombuf(char*&, double*)</u>
<u>50</u>	4.67	2.92	<u>adler32</u>
<u>51</u>	3.12	1.95	<u>__read_nocancel</u>
<u>55</u>	2.43	1.52	<u>memcpy</u>
<u>43</u>	2.19	1.37	<u>TBufferFile::ReadFastArray(double*, int)</u>
<u>56</u>	1.84	1.15	<u>inflate_table</u>
<u>20</u>	1.10	0.69	<u>inflate</u>
<u>57</u>	0.94	0.59	<u>tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::receive_or_steal_task(long&)</u>
<u>72</u>	0.63	0.40	<u>__memset_sse2</u>
<u>74</u>	0.59	0.37	<u>sched_yield</u>
<u>137</u>	0.19	0.12	<u>__lll_unlock_wake</u>

Tools: Extrae + Paraver

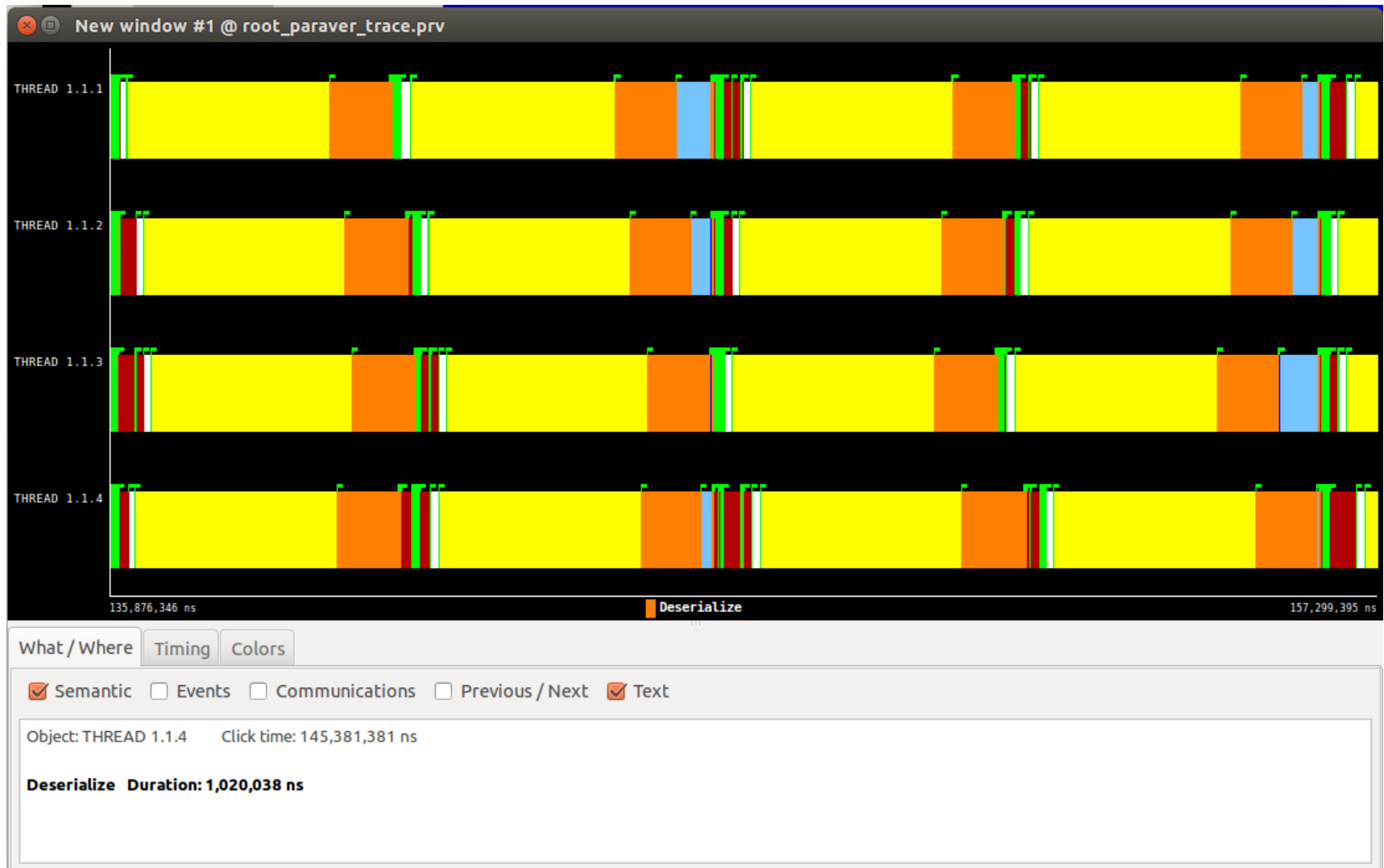
- Extrae
 - Instrumentation tool
 - Can be used to emit user events
 - Generates trace files
- Paraver
 - Graphical performance analysis
 - Displays trace files



Tools: Paraver trace



Tools: Paraver trace (II)



- VTune
 - ✓ Thread graphical display
 - ✓ Analysis of waits, TBB overhead
 - ✗ Lock analysis not accurate
 - ✗ Significant overhead
- IgProf
 - ✓ Low overhead
 - ✗ Underestimation of lock cost
- Extrae + Paraver
 - ✓ Graphical analysis of the application's behavior
 - ✓ Event-level detail
 - ✗ Limit of events?

- Implement the union of PBP + User lambda
 - Combine ROOT and user tasks for a better thread utilization
- Apply parallelization to a real use case
 - CMS analysis application
 - Verify results
- If possible, apply similar strategies to the rest of parallelization use cases
 - Validate approach before including it into a ROOT release