Event Data Processing Frameworks for the Future

- The Vision
- The Model
- The Guinea pig
- Results

The Problem

- Resources are scarce
  - Process parallelization does not address modern CPU technology
    - Many cores [Intel Many Integrated Core Architecture: 80]
  - Scarce memory / CPU core
  - Number of open files per node ➔ castor, hpms, Oracle
  - …

➔ Minimize resource usage (memory, files)

➔ Let multiple threads use the same resources

-- I/O buffers, detector description, magnetic field map, histograms, static storage, …

~ 1-2 thread per hardware thread

➔ Pipelined Data Processing (PDP)

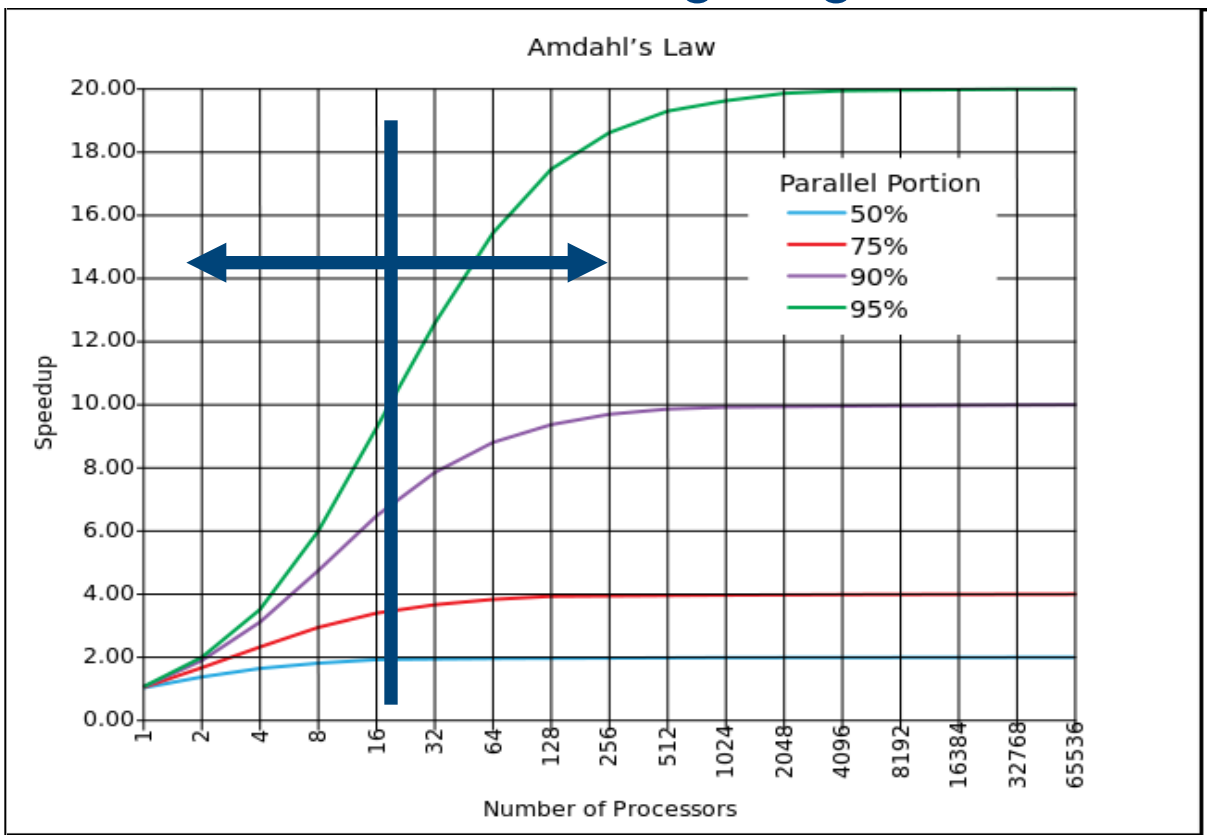
Pipelined Data Processing

- Two parallelization concepts
  - Event parallelization
    simultaneous processing of multiple events
  - Algorithm parallelization for a given event
    simultaneous execution of multiple Algorithms
- Both concepts may coexist
- Additional benefit:
  Processing a given set of events may be faster

- **Glossary** (Gaudi-speak):
  - Event are processed by a sequence of Algorithms
  - An Algorithm is a considerable amount of code
    acting on the data of one event [not just sqrt(x)]

Amdahl's Law

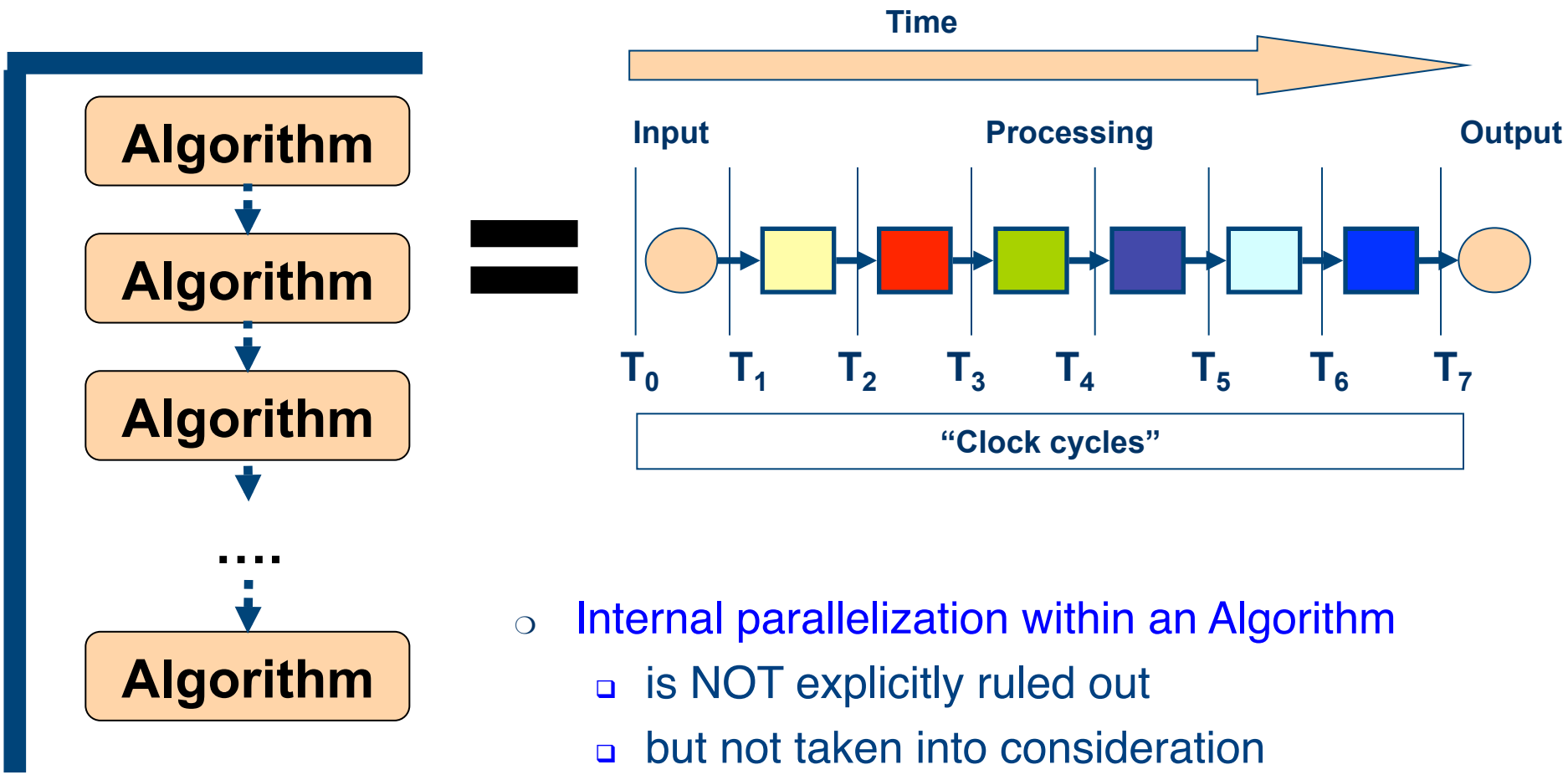
- What is the possible gain that can be achieved ?
  - Speedup = 1 / (serial + parallel / N_{thread})
  - In which area are we navigating?



Answers required

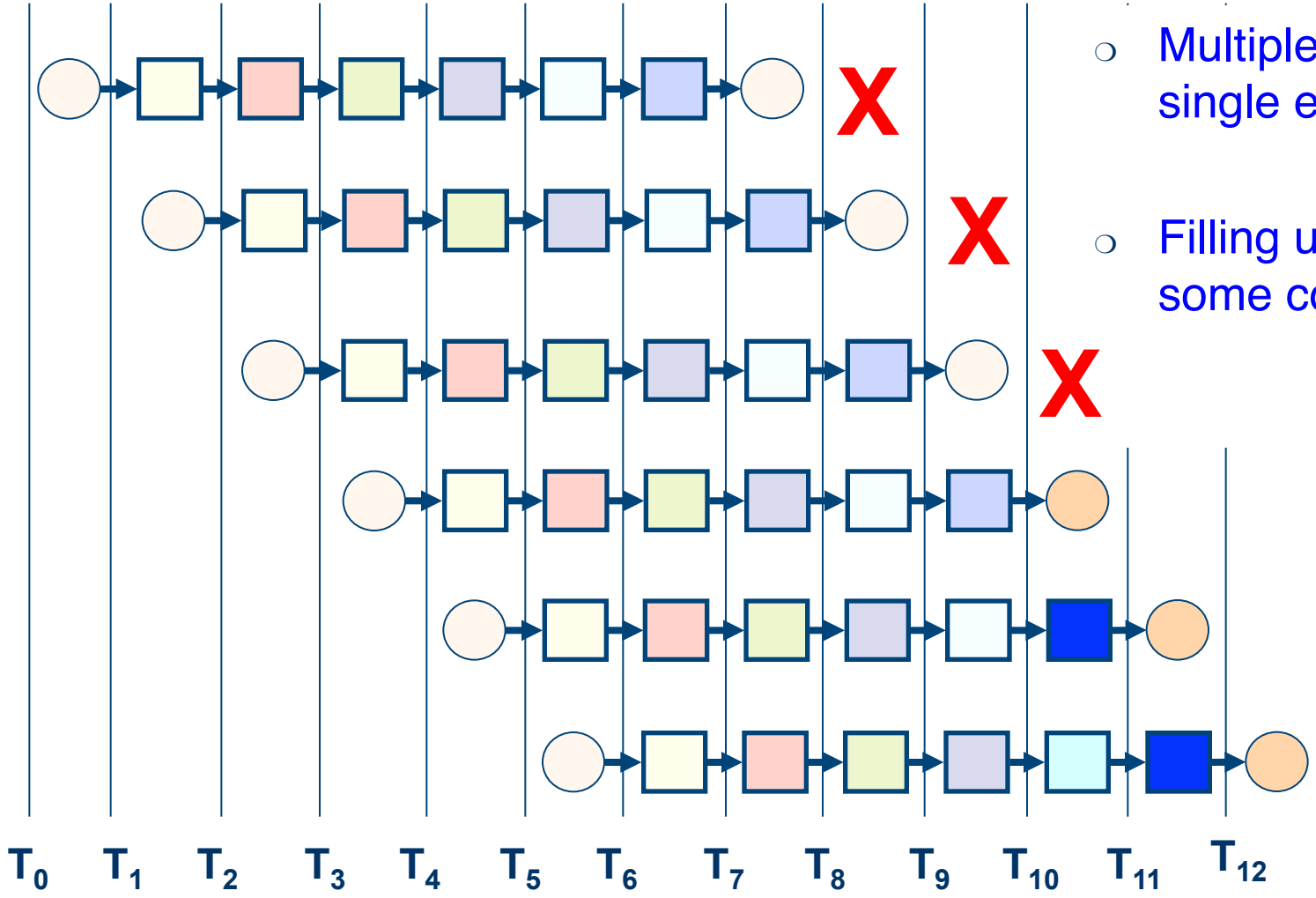
- Using the Pipelined Data Processing paradigm:
  - Which ***speedup*** can be achieved ?
  - Which parameters will the model have ?
  - What amount of work is required to transform an existing program
    - Framework
    - Physics code

Pipelined Data Processing



- Internal parallelization within an Algorithm
  - is NOT explicitly ruled out
  - but not taken into consideration

Pipelined Data Processing: Event Parallelism



- Multiple instances of single event queues
- Filling up threads up to some configurable limit

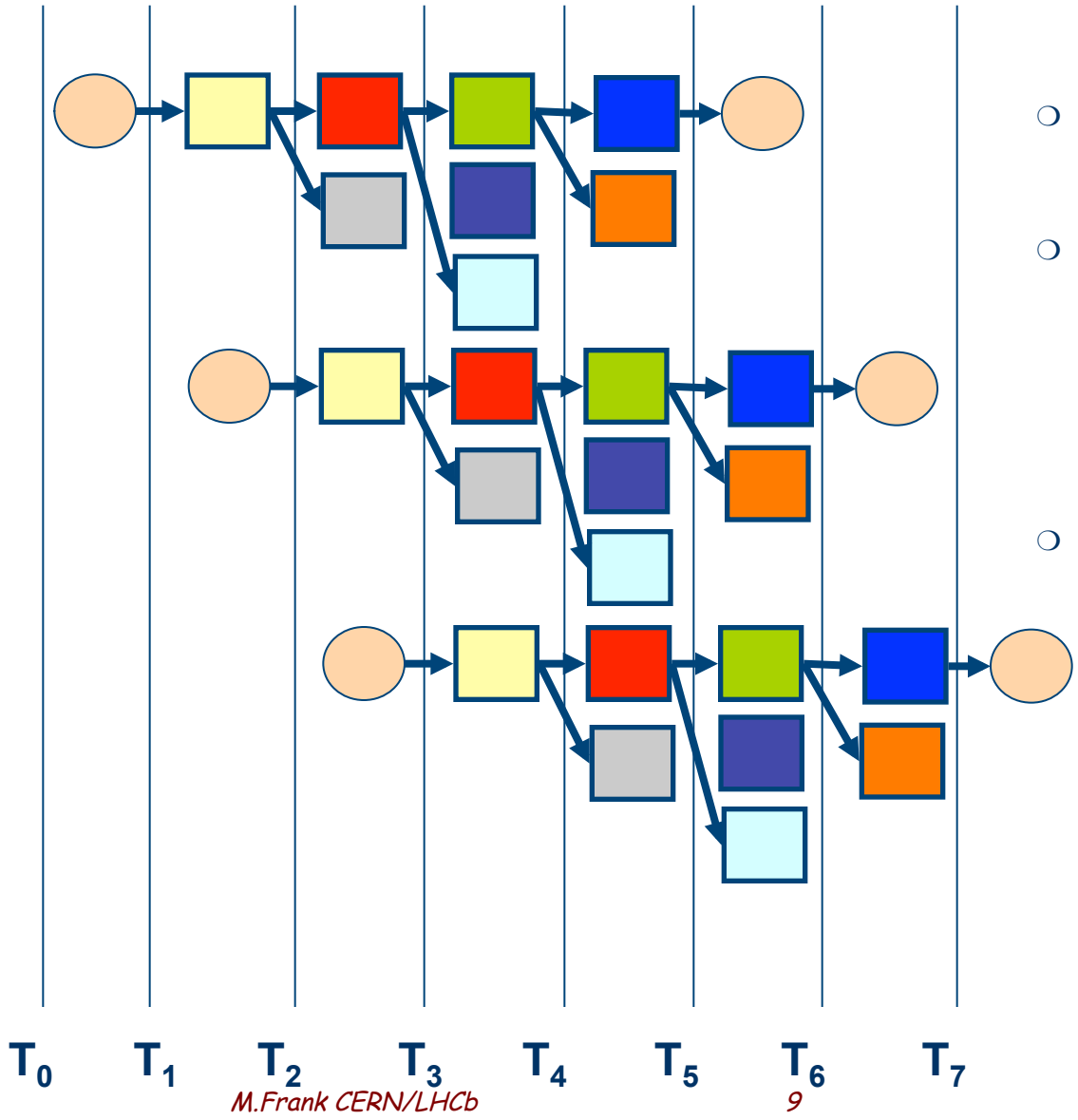
Pipelined Data Processing
Algorithm Parallelization

- Algorithms consume data from the TES
  (transient event data store – blackboard for event data)
- Algorithms post data to the TES

Basic assumptions:

- The execution order of any 2 algorithms with the same input data does not matter
- They can be executed in parallel

Consequence



- Can keep more threads busy at a time
- Hence:
  - Less events in memory
  - Less memory used
- Example
  - First massage raw data for each subdetector (parallel)
  - Then fit track…

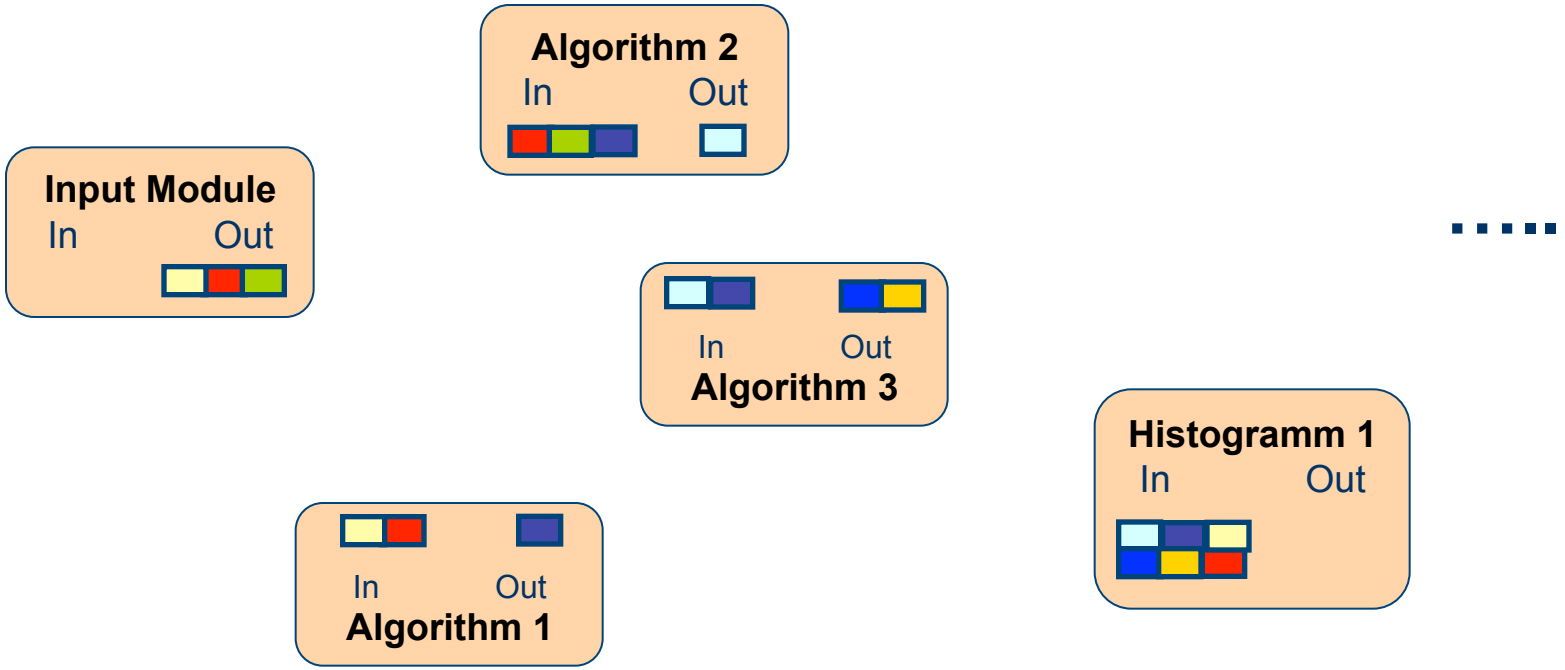
The Guinea Pig Model

- Paragon: LHCb reconstruction program “Brunel”
- Implement **Pipelined Data Processing** model
- With input from real event execution
  - Which algorithms are executed
  - Average wall time each algorithm requires
  - List of required input data items for each algorithm

- The Model
  - Replace execution with “sleep”
    Not entirely accurate, but a reasonable approximation

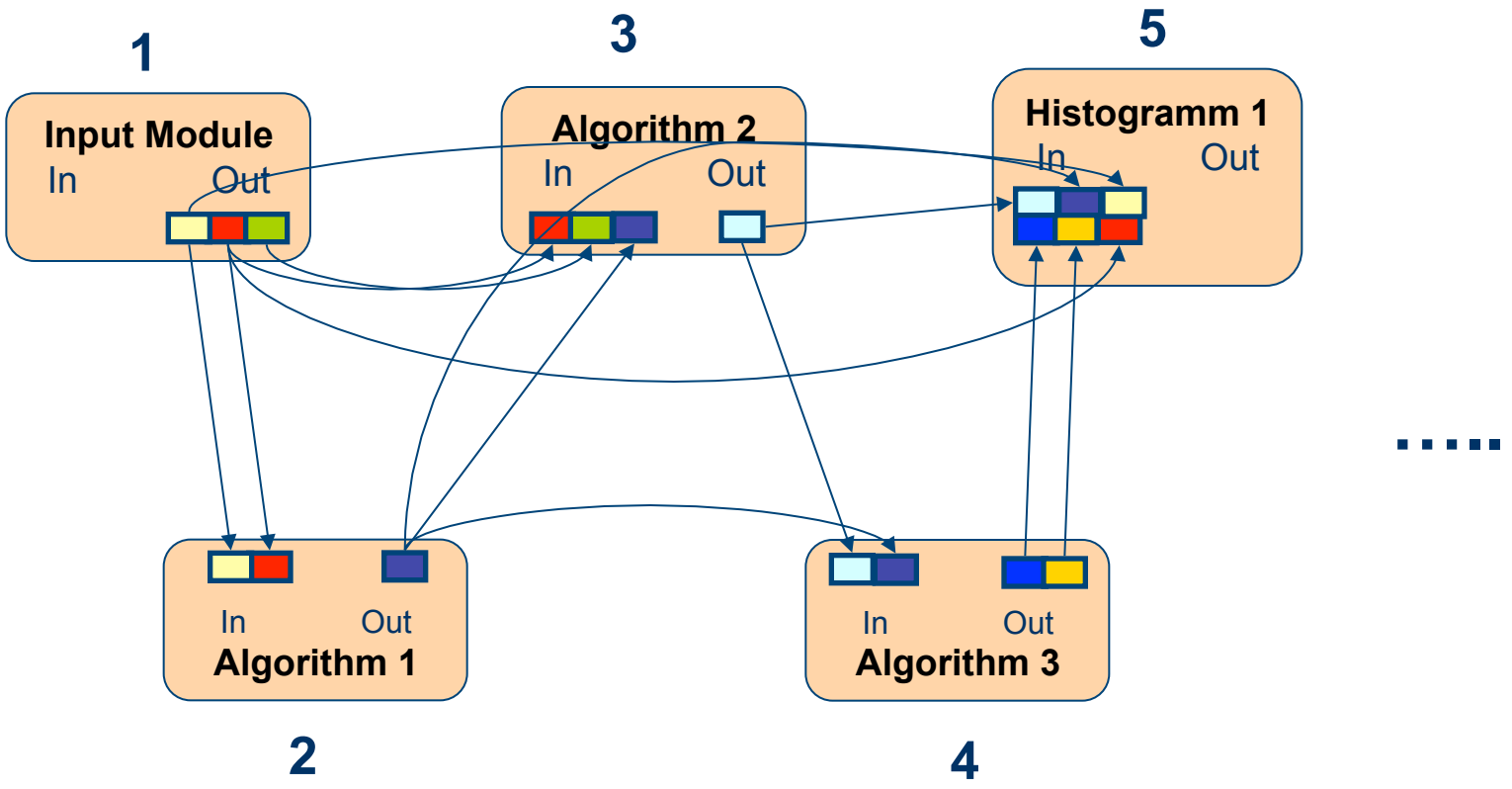
Pipelined Data Processing: Configuration

- Start with a sea of algorithms
  - Match inputs with outputs
    - ➔ Algorithm dependencies
    - ➔ Execution order
  - Model dependencies obtained by snooping on TES

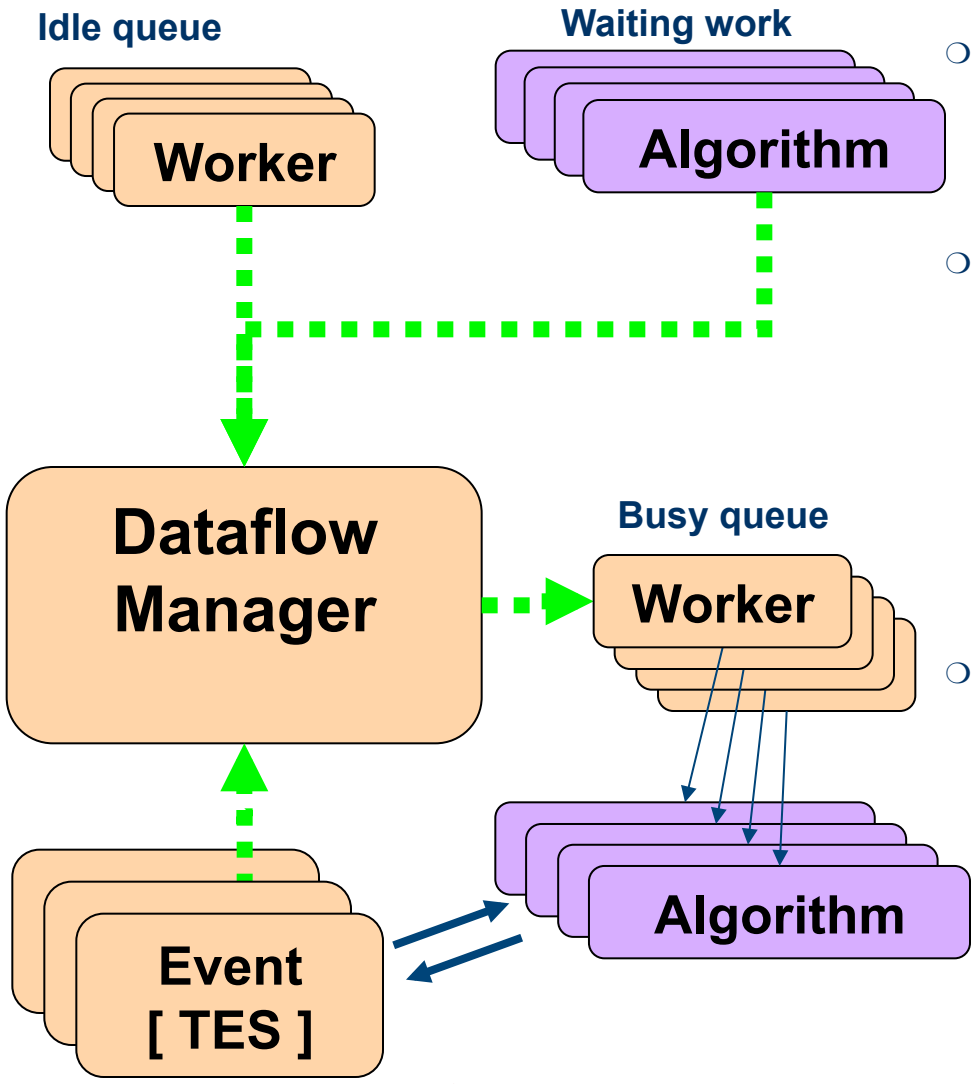


Pipelined Data Processing: Configuration

- Resolved Algorithm queue after snooping

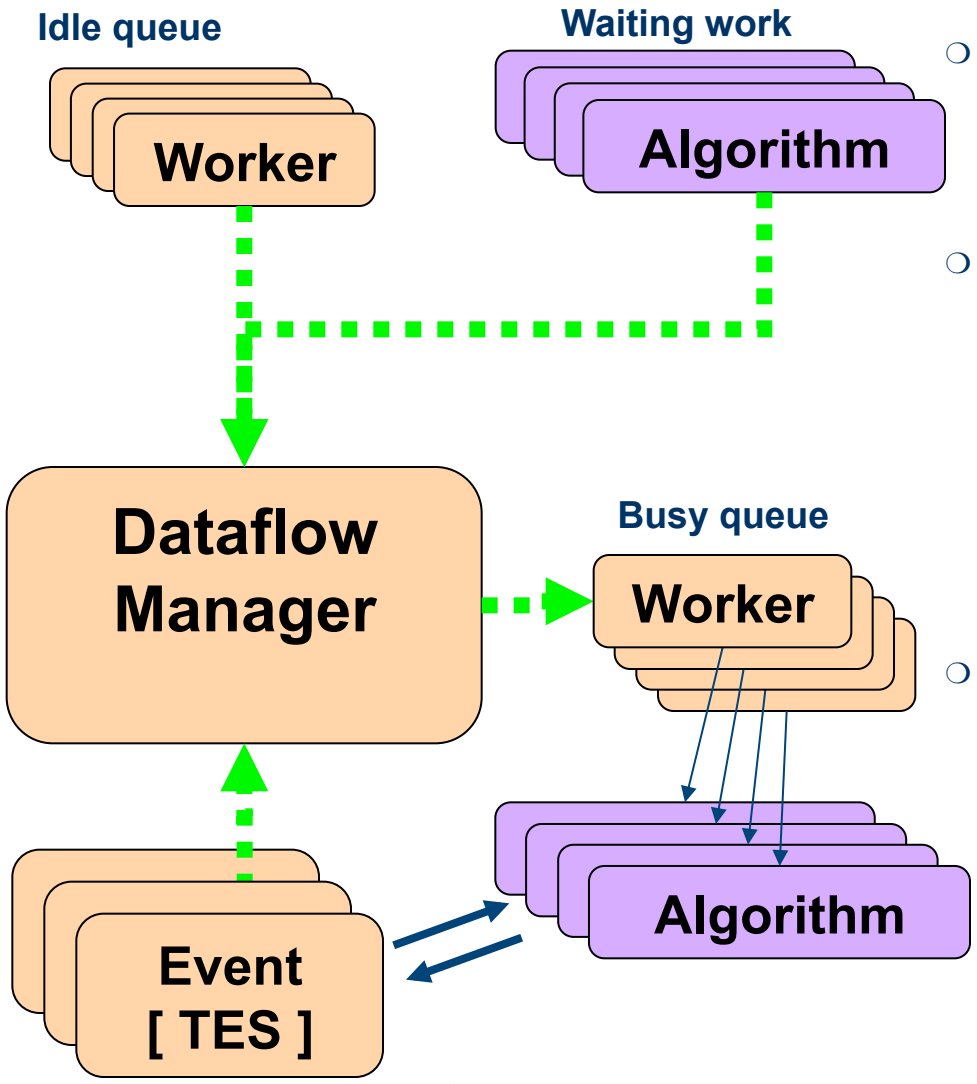


Conceptual Model: Executors, Workers and Manager



- Formal workload given to a worker
- As long as work and idle workers:
  ➔ schedule an algorithm
  - acquire worker from idle queue
  - attach algorithm to worker
  - submit worker
- Once Worker is finished
  - put worker back to idle queue
  - Algorithm back to “sea”
  - Evaluate TES content to reschedule workers

Conceptual Model: Executors, Workers and Manager



- Formal workload given to a worker
- As l… …rs:
  - ➔ s…
    - a… …eue
    - a…
    - s…
- Onc…
  - p… …ue
  - A…
  - E… reschedule workers

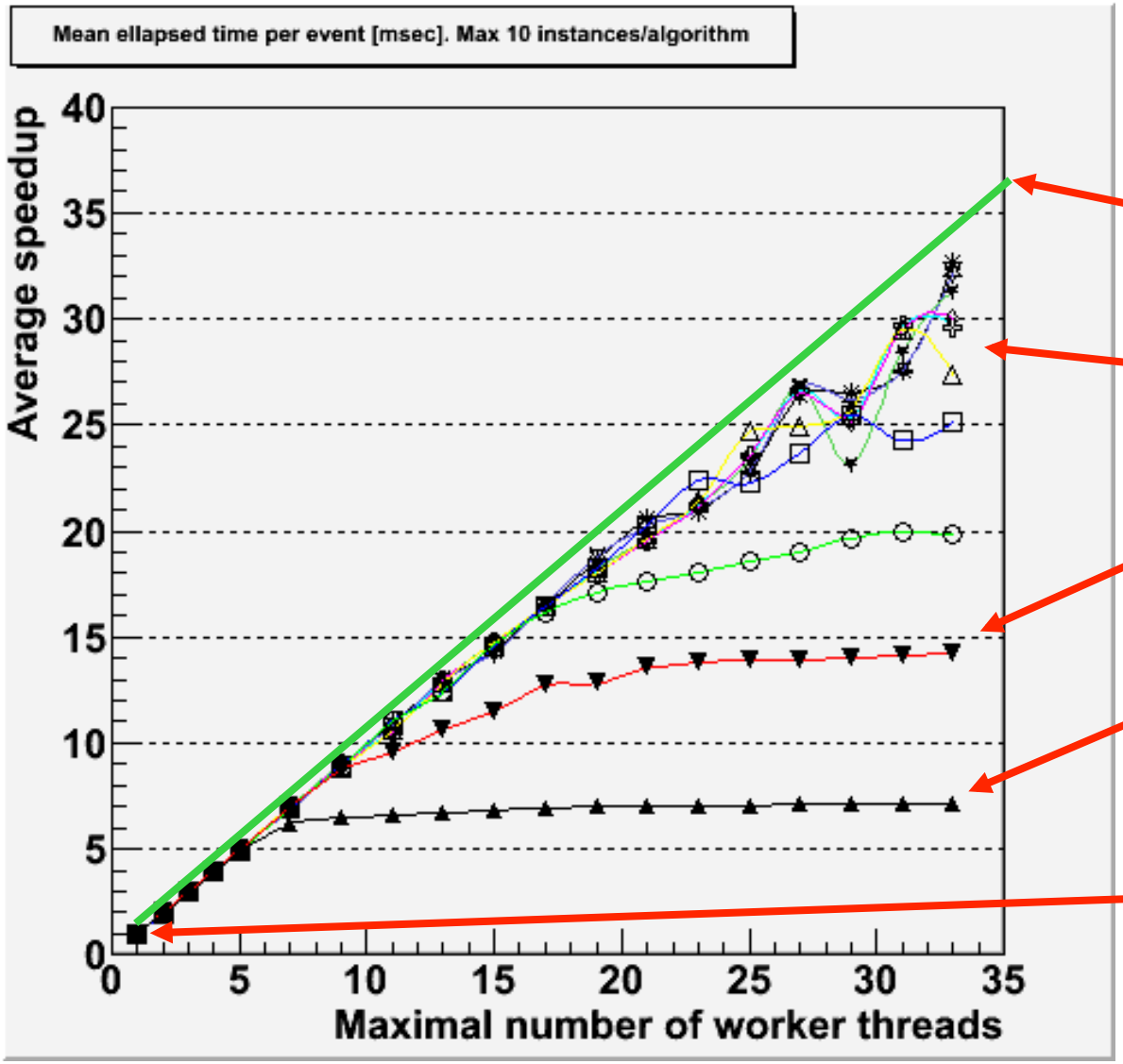
Machinery implemented using GCD
(Grand Central Dispatch)

but: Standalone implementation simple (was predecessor)

The Guinea Pig Model: Parameter Space

- All parameters “within reason”
- Global model parameters
  - Maximal number of threads allowed. Max ~ 40
- Event parallelization parameters
  - Maximal number of events processed in parallel
  - Maximal 10 events
- Algorithmic parallelization parameters
  - Maximal number of instances of a given Algorithm
  - By definition <= number of parallel events

Model Result: Assuming full reentrancy



- **Max 10 events in parallel**
- **Max 10 instances/algorithm**
- **All algorithms reentrant**

Theoretical limit
$t = t_1 / N_{thread}$

Max evts > 3
Speedup up to ~30

Max 2 events
1 event * 2

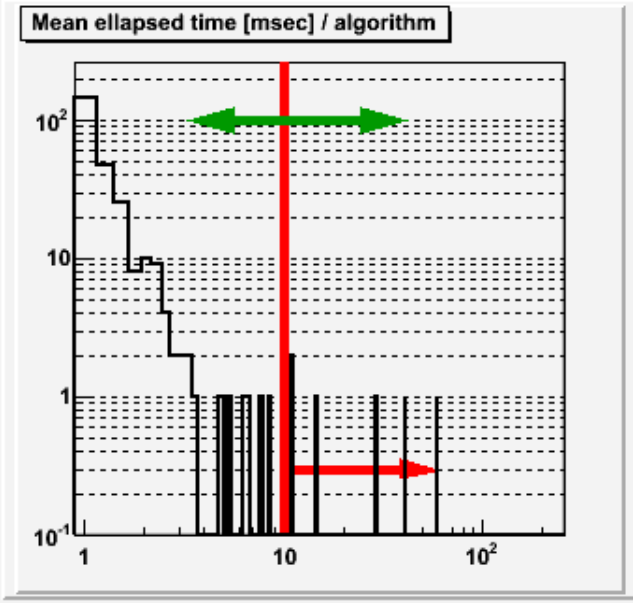
Max 1 event
Algorithmic parallel limit
Speedup: ~7

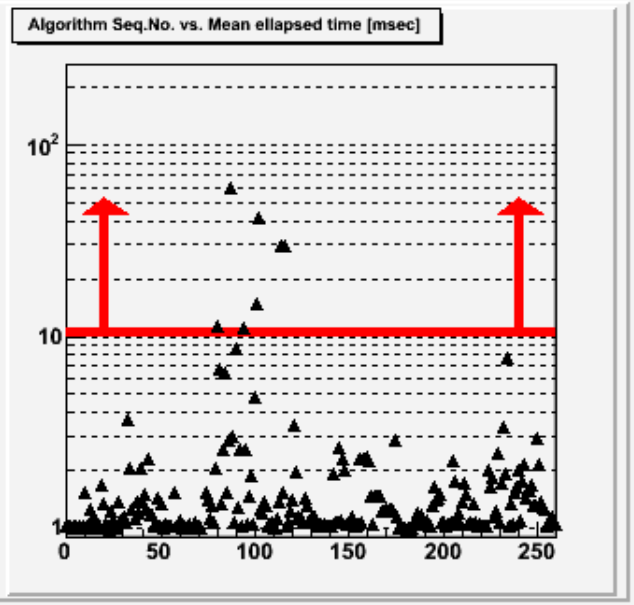
One thread
= classic processing (t_1)

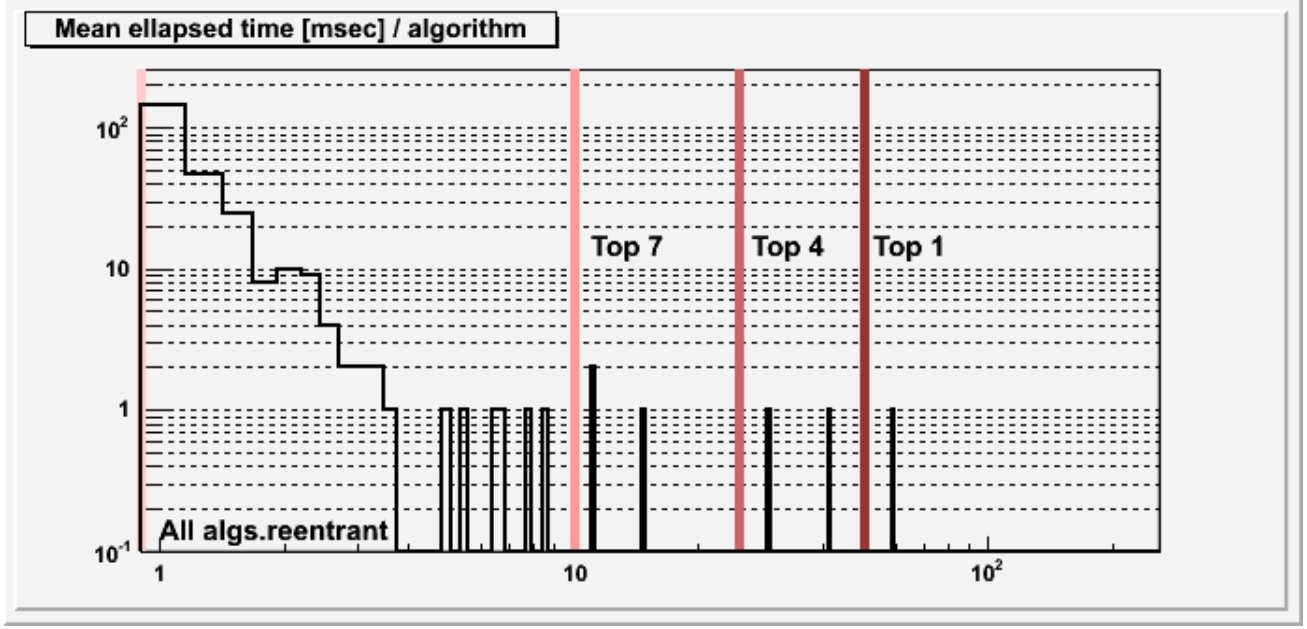
Model Result: Assuming full reentrancy

- The result only shows that the model works
- However, such an implementation would be
  - Not practical in the presence of (a lot of) existing code since all of it must be reentrant
  - Hell of a work – if possible at all
- Measures are necessary
  - Not only for a transition phase
  - Some algorithms cannot be made reentrant
  - Exercise: Only make top N algorithms reentrant

What does this really mean?







Vary a cutoff, which defined, which algorithms **must** be reentrant

Model Result:
The top 7 time consuming algorithms

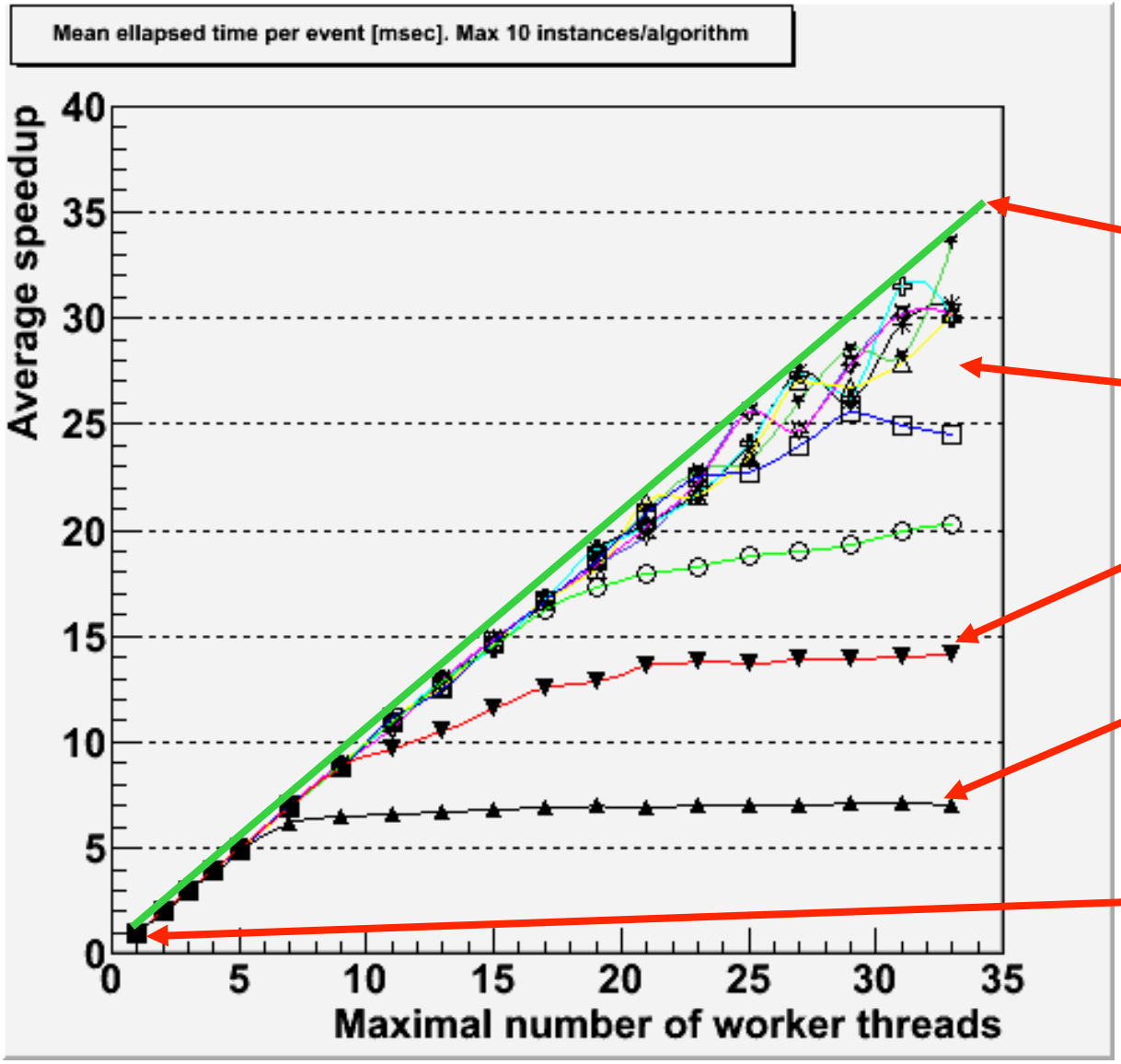
Average proc. time/event	**580 msec**	**100 %**
FitBest	58 msec	10.0 %
CreateOfflinePhotons	40 msec	6.8 %
RichOfflineGPIDLLIt0	28 msec	5.0 %
RichOfflineGPIDLLIt1	29 msec	4.8 %
CreateOfflineTracks	14 msec	2.4 %
PatForward	10 msec	1.7 %
TrackAddLikelihood	10 msec	1.7%
Top 7:	**189 msec**	**32.6 %**

top 1

top 4

top 7

Model Result Top 7: Max. 10 instances of top 7 algorithms



- **Max 10 events in parallel**
- **TOP 7 algorithms reentrant with max. 10 instances**
- **Cut 10 msec [1.7 %]**

Theoretical limit

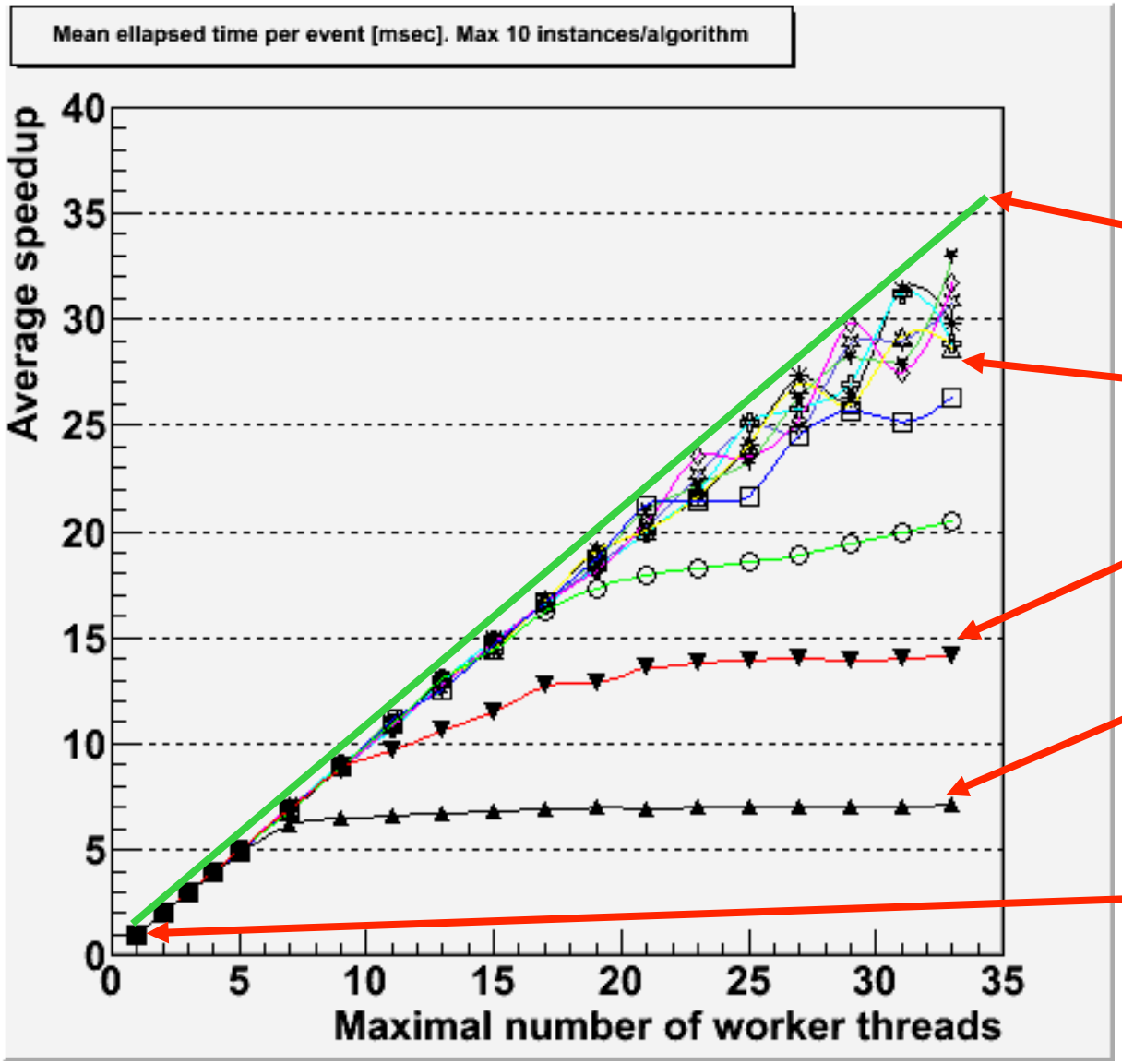
Max evts > 3
Speedup up to ~30

Max 2 events
1 event * 2

Max 1 event
Algorithmic parallel limit
Speedup: ~7

One thread
= classic processing (t_1)

Model Result Top 4: Max. 10 instances of top 4 algorithms



- Max 10 events in parallel
- TOP 4 algorithms reentrant with max 10 instances
- Cut 25 msec [4.3 %]

Theoretical limit

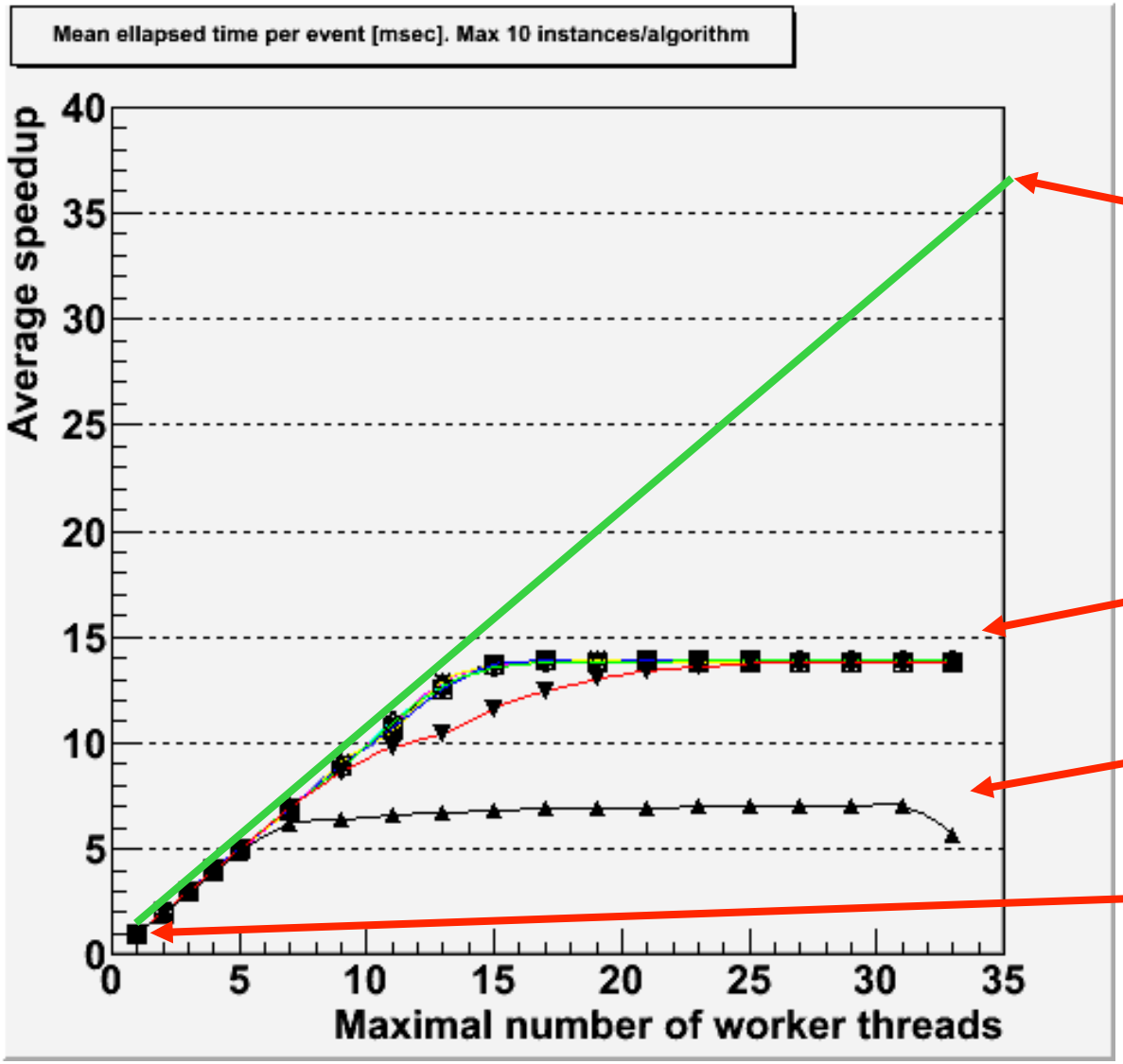
Max evts > 3
Speedup up to ~30

Max 2 events
1 event * 2

Max 1 event
Algorithmic parallel limit
Speedup: ~7

One thread
= classic processing (t_1)

Model Result Top 1: Max. 10 instances of top algorithm



- Max 10 events in parallel
- TOP 1 algorithm reentrant with max 10 instances
- Cut 50 msec [10 %]

Theoretical limit

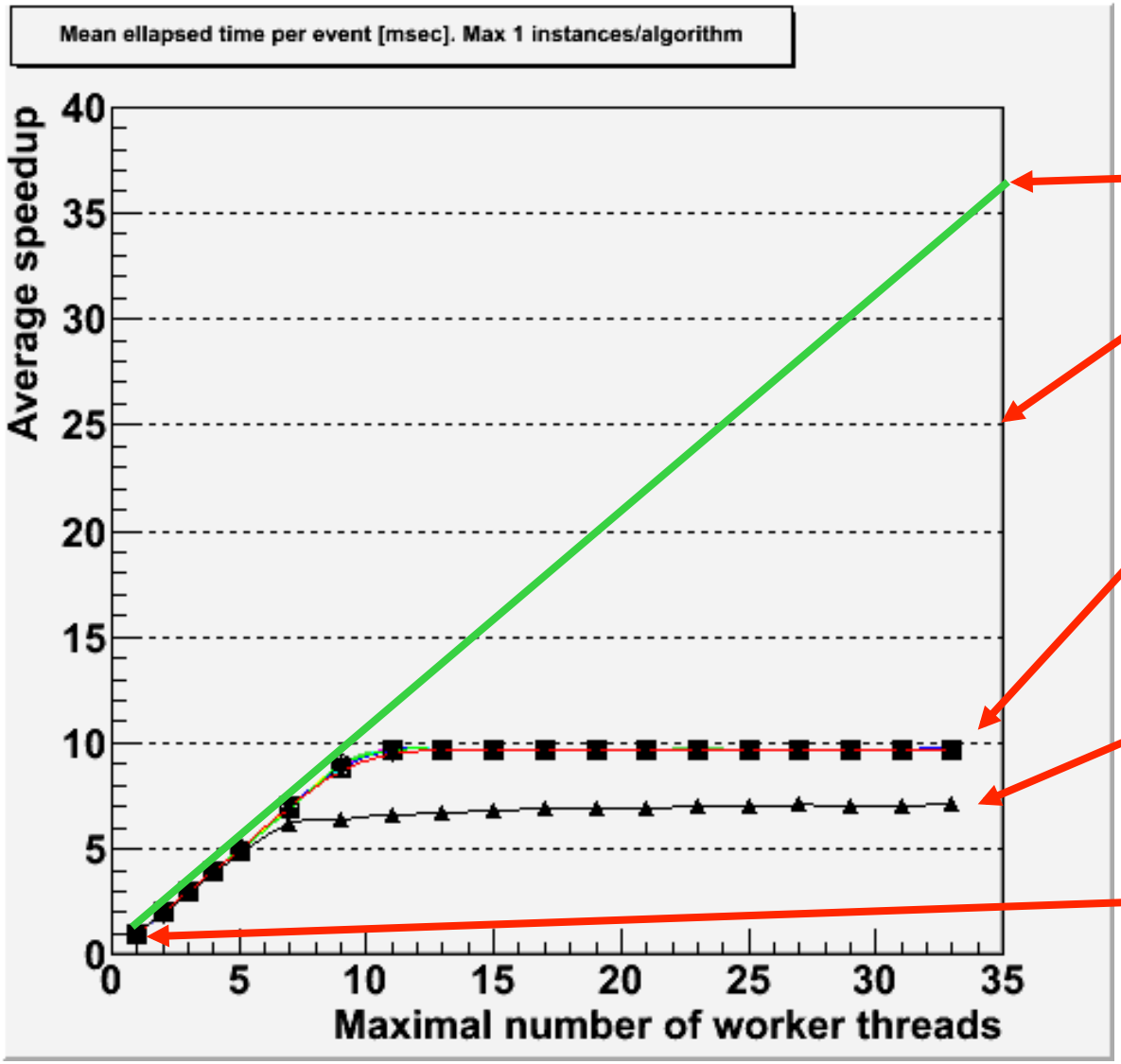
Max evts > 3
No improvement
Not sufficient

Max 2 events
Speedup ~ 1 event * 2

Max 1 event
Algorithmic parallel limit
Speedup: ~7

One thread
= classic processing (t_1)

Model Result: Importance of Algorithm Reentrancy



- Max 10 events in parallel
- Max 1 instance/algorithm

Theoretical limit

Allowing for more events will not improve things anymore

Dominated by execution time of slowest algorithm

Max 1 event
Algorithmic parallel limit
Speedup: ~7

One thread
= classic processing (t_1)

Results: Summary

- Provided **both** parallelization mechanisms are applied
  - Large wall time gains could be achieved
  - Factor 30 not out of reach
  - Framework infrastructure resources reduced by this factor
- Many changes are only internal to the framework
  - Multiple event processing
  - Thread safe data access
- Only top time consuming algorithms must be closely watched and made reentrant

Implications to existing frameworks

- Can the implementation of such an a processing framework be applied to existing code?
  - depends…
  - Algorithms and framework components must be able to deal with several events in parallel
    - e.g. Single “blackboard” would not do
    - The state of Algorithm instances may not depend on the current event
  - The Algorithm chain must be divisible into “self-contained” units
    - Locking typically not supported by existing frameworks
    - Spaghetti code is a killer…
  - Otherwise: Yes; this can be applied to existing frameworks

Program of Work

- V Build a prototype
- V Test with dummy Algorithms
- V Measure possible gains
- X Get more physics code and core software developers in the boat
- X Develop framework changes to support parallelization
- X Apply to existing reconstruction program
  - X Start to convert existing physics code base
    1rst. goal: 7 algorithm reentrant
    => workout mechanisms
- X Measure performance

Conclusions

- Only both, event and algorithm parallelization shows the full potential of many core Algorithms
- Not all of the physics code base must be changed at once
- Smooth transition phase is provided
- If most of the implications can be hidden by the framework
- Still: a lot of work coming up
- Migration cannot be transparent
  - has to be agreed / prepared / scheduled by the code developers in of the whole collaboration

Backup Slides

Dataflow Manager V2
GCD implementation