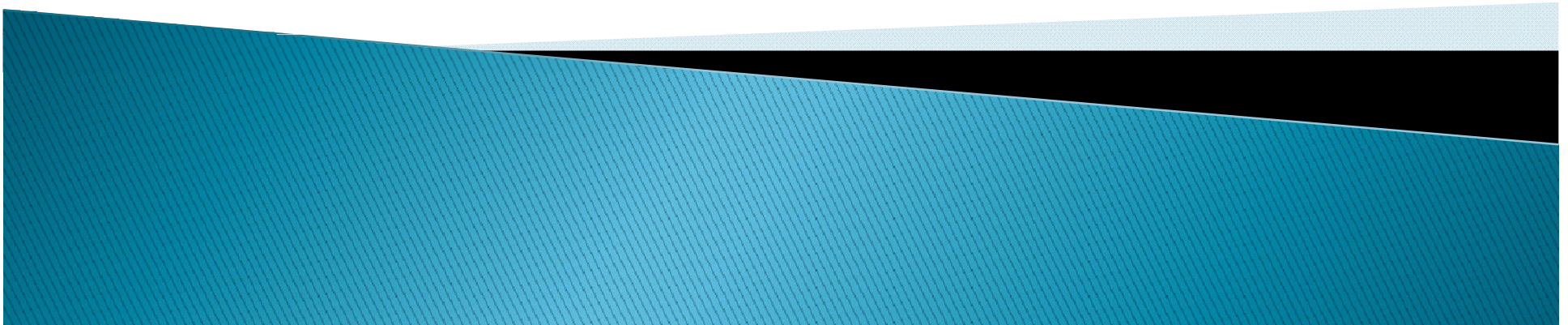


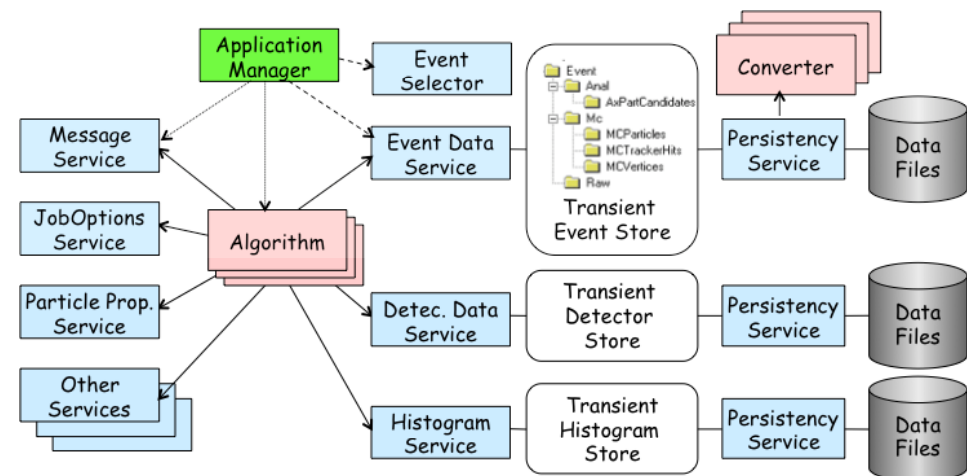
User-friendly Parallelization of GAUDI Applications with Python

CHEP 09, 22–27 March 2009, Prague
Pere Mato (CERN), Eoin Smith (CERN)



GAUDI Introduction

- ▶ GAUDI is a mature software framework for event data processing used by several HEP experiments
 - First presented at CHEP 2000 in Padova
- ▶ A Python interface to configure and run [interactively] GAUDI applications exists
 - GaudiPython
 - Python Configurables



Motivation

- ▶ Work done in the context of the R&D project to exploit multi-core architectures
- ▶ Main goal
 - Get quicker responses when having 4–8 core machines at your disposal
 - Minimal changes (and understandable) to the Python scripts driving the program
- ▶ Leverage from existing third-party Python modules
 - *processing* module, *parallel python* (pp) module
- ▶ Optimization of memory usage

A typical GaudiPython script

```
#--- Common configuration -----
from Gaudi.Configuration import *

importOptions('$STDOPTS/LHCbApplication.opts')
importOptions('DoDC06selBs2Jpsi2MuMu_Phi2KK.opts')

#--- modules to be reused in all processes-----
from GaudiPython import AppMgr
from ROOT import TH1F,TFile,gROOT

files = [...]
hbmass = TH1F('hbmass','Mass of B cand',100,5200.,5500.)
appMgr = AppMgr()
appMgr.evtSel().open(files)
evt = appMgr.evtSvc()

while True :
    appMgr.run(1)
    if not evt['Rec/Header'] : break
    cont = evt['Phys/DC06selBs2Jpsi2MuMu_Phi2KK/Particles']
    if cont :
        for b in cont : hbmass.Fill(b.momentum().mass())

hbmass.Draw()
```

Predefined
configuration +
customizations

Preparations
(booking histos,
selecting files, etc.)

Looping over
events

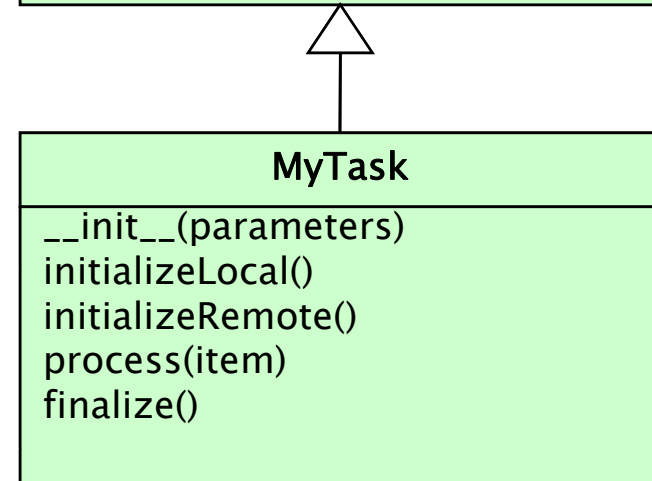
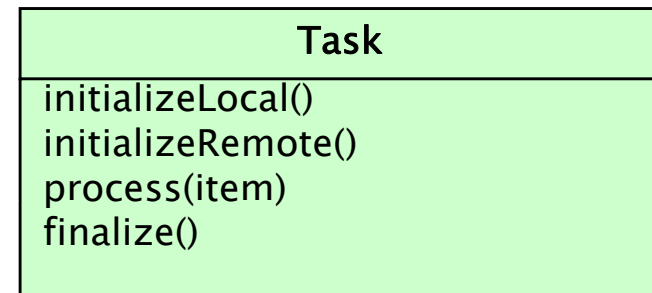
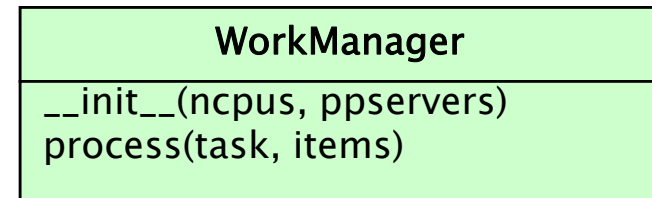
Presenting results

Parallelization Model

- ▶ Introduced a very simple Model for parallel processing of tasks
- ▶ Common model that can be implemented using either *processing* or *pp* modules (or others)
- ▶ The result of processing can be any ‘pickle-able’ Python or C++ object (with a dictionary)
- ▶ Placeholder for additional functionality
 - Setting up the environment (including servers)
 - Monitoring
 - Merging results (summing what can be summed or appending to lists)

The Model

- ▶ User parallelizable task derives from *Task*
 - *initializeLocal()* is executed in parent process
 - *initializeRemote()* is executed once in each remote process
 - *process()* is executed for each work *item* in remote process
 - *finalize()* is executed at the end in the parent process



Very simple example

```
from ROOT import TH1F, TRandom, TCanvas, gROOT
from GaudiPython.Parallel import Task, WorkManager
```

```
class HistTask(Task):
    def __init__(self, nHist=4):
        self.nHist = nHist
        self.canvas = None

    def initializeLocal(self):
        self.output = [TH1F('%d%i', '%d%i', 100, -3., 3.) for i in range(self.nHist)]
        self.random = TRandom()
```

```
>>> from GaudiPython.Parallel import WorkManager
>>> from HistTask import HistTask

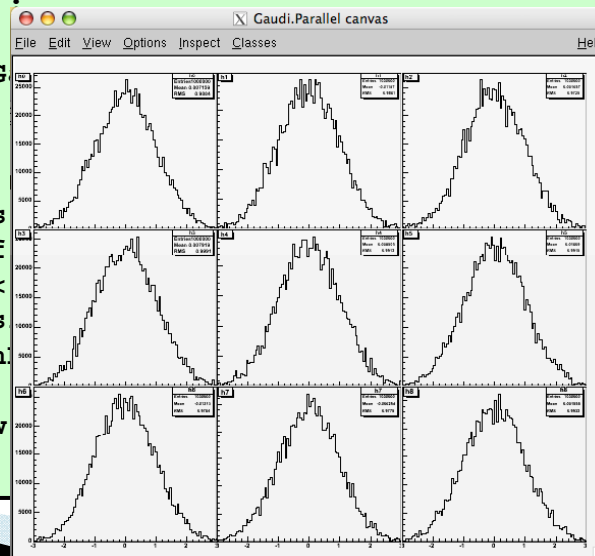
>>> task = HistTask(nHist=9)
>>> wmgr = WorkManager()
>>> wmgr.process(task, [100000 for i in range(100)])
```

```
def in Job execution statistics:
    job count | % of all jobs | job time sum | time per job | job server
    100 | 100.00 | 378.296 | 3.783 | lxbuild114.cern.ch

def pr Time elapsed since server creation 60.805200
```

```
for h in self.output:
    for i in range(n):
        x = self.random.G
        h.Fill(x)
```

```
def finalize(self):
    self.canvas = TCanvas
    nside = int(sqrt(self
    nside = nside*nside <
    self.canvas.Divide(ns
    for i in range(self.n
        self.canvas.cd(i+1)
        self.output[i].Draw
```



```
0, 10, 700, 700)
```

More Realistic Example

```
from ROOT import TFile, TCanvas, TH1F, TH2F
from Gaudi.Configuration import *
from GaudiPython.Parallel import Task, WorkManager
from GaudiPython import AppMgr

class MyTask(Task):
    def initializeLocal(self):
        self.output = {'h_bmass': TH1F('h_bmass', 'Mass of B candidate', 100, 5200., 5500.)}

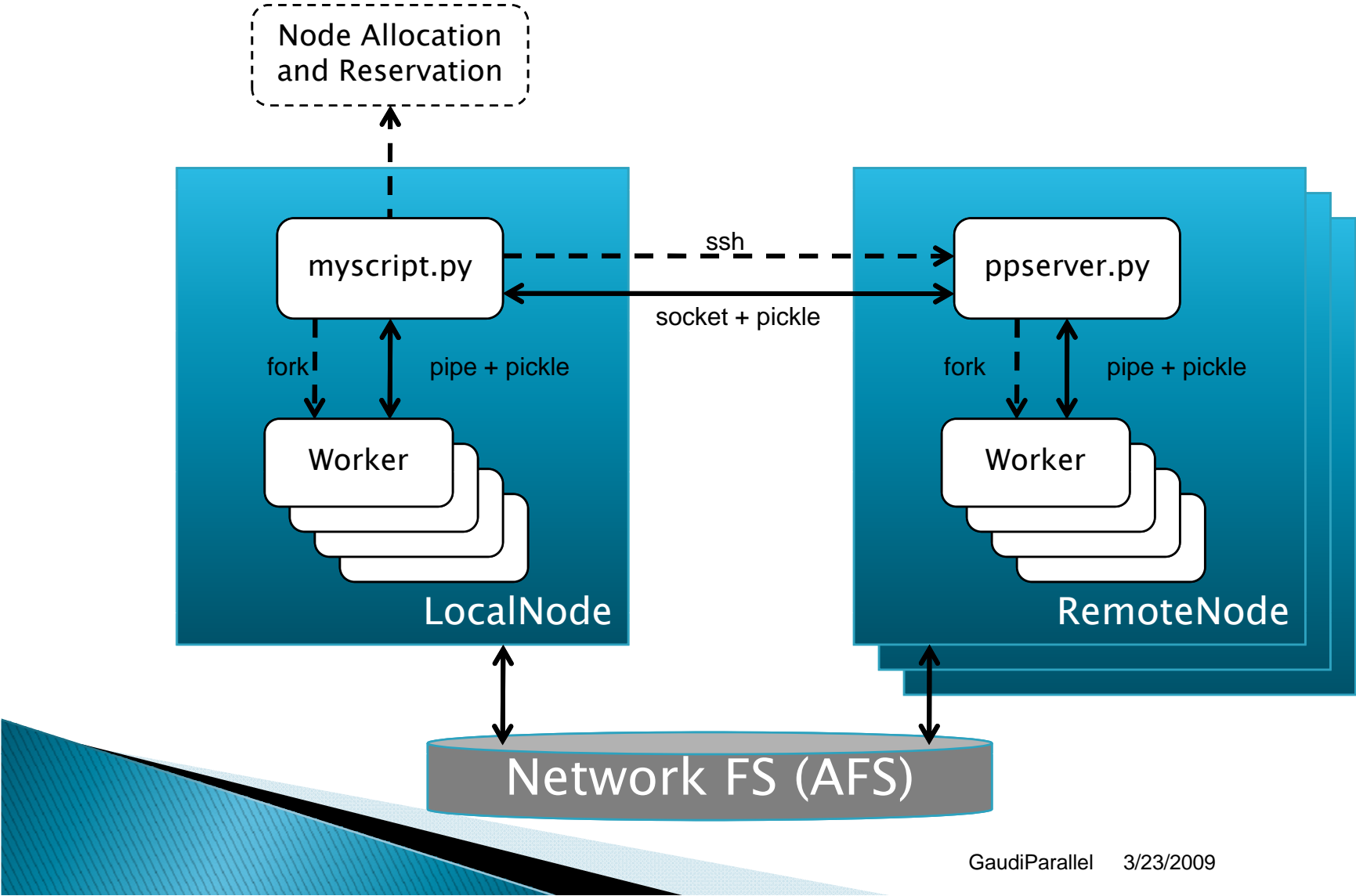
    def process(self, file):
        appMgr = AppMgr()
        appMgr.evtSel().open([file])
        evt = appMgr.evtSvc()
        while 0 < 1:
            appMgr.run(1)
            # check if there are still valid events
            if evt['Rec/Header'] == None : break

from GaudiPython.Parallel import WorkManager
from MyTask import MyTask
from inputdata import bs2jpsipimm_files as files

ppservers = ('lxbuild113.cern.ch', 'lxbuild114.cern.ch')

if __name__ == '__main__':
    task = MyTask()
    wmgr = WorkManager(ppservers=ppservers)
    wmgr.process( task, files[:50])
```


Parallelization on a Cluster



Cluster Parallelization Details

- ▶ The [pp]server is created/destroyed by the user script
 - Requires ssh properly setup (no password required)
- ▶ Ultra-lightweight setup
- ▶ Single user, proper authentication, etc.
- ▶ The ‘parent’ user environment is cloned to the server
 - Requires a shared file system
- ▶ Automatic cleanup on crashes/aborts

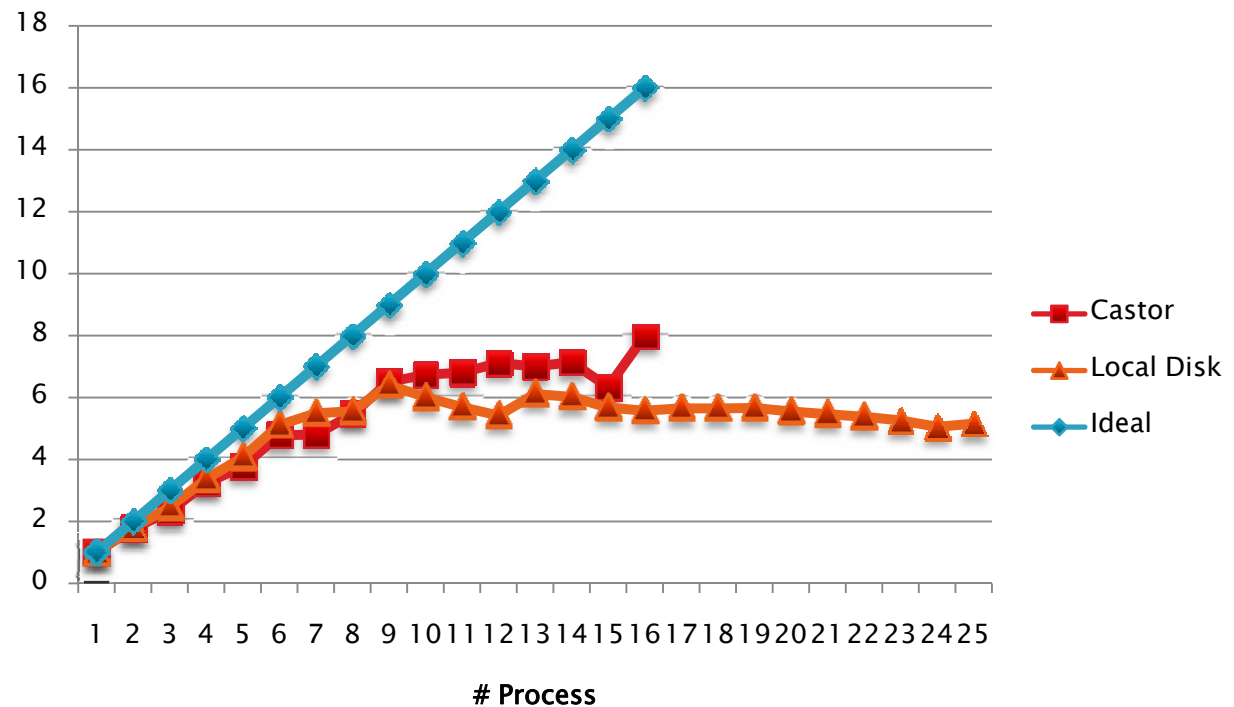
Speedup in Multi-core systems

Example reading
50 DST files
from castor (rfio)
or local disk and
filling some
histogram

$$S_p = \frac{T_1}{T_p}$$

With 9 process a
speedup of 7 can
be expected

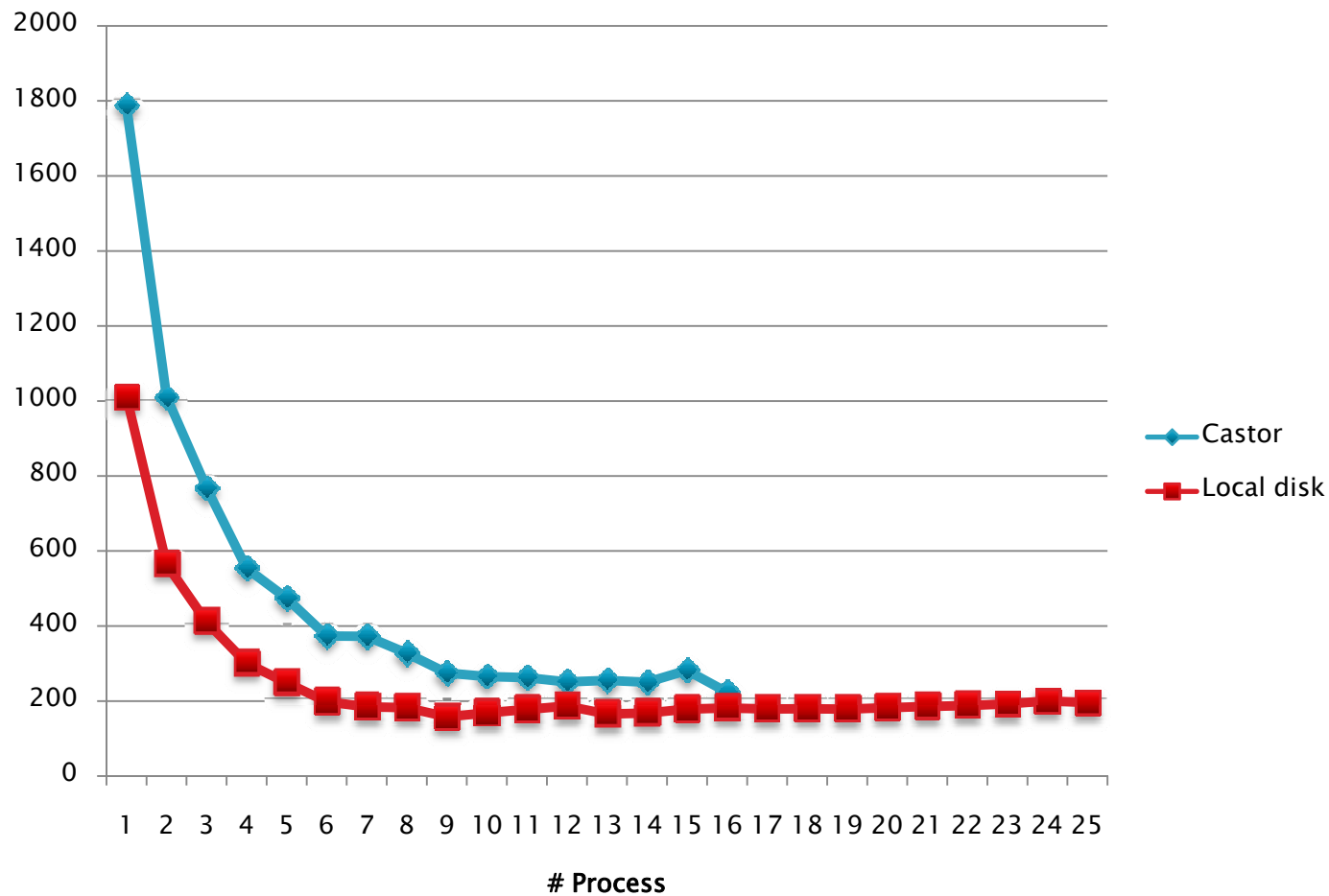
Speedup (8 cores)



Elapsed Time

Total Execution Time (8 cores)

Same example reading 50 DST files. The castor case is probably protocol bound (~70 % CPU utilization/processes)



Shared Memory Measurements

- ▶ Measuring memory usage with smaps
 - Data lives in `/proc/$pid/smaps`
 - Developed python script to digest it

```
3a5c100000-3a5c115000 r-xp 00000000 08:03 621533 /lib64/ld-2.3.4.so
Size:                84 kB
Rss:                 76 kB
Pss:                 1 kB
Shared_Clean:        76 kB
Shared_Dirty:         0 kB
Private_Clean:        0 kB
Private_Dirty:        0 kB
Referenced:          76 kB
```

- ▶ Parallelism achieved using Python *processing* module
 - Uses *fork()* internally
- ▶ Tested with LHCb reconstruction program *Brunel*
 - Version (v33r1)

Parallel Brunel (reconstruction)

```
from Gaudi.Configuration import *
from GaudiPython.Parallel import Task, WorkManager
from GaudiPython import AppMgr, PyAlgorithm

class Brunel(Task):
    def initializeLocal(self): pass

    def initializeRemote(self):
        importOptions('$BRUNELROOT/options/Brunel-Default.py')
        importOptions('$BRUNELROOT/options/DC06-Files.py')
        ApplicationMgr( OutputLevel = ERROR, AppName = 'PyBrunel')
        appMgr = AppMgr()
        appMgr.initialize()

    def process(self, file):
        appMgr.evtSel().open([file])
        appMgr.run(10)

    def finalize(self): pass

from Files import digi_files as files
if __name__ == '__main__':
    task = Brunel()
    wmgr = WorkManager(ncpus=4)
    wmgr.process( task, files[:4])
```

} Initialization on
each sub-
process

} Process 10
events in each
file

} Main program

With Common Initialization

```
from Gaudi.Configuration import *
from GaudiPython.Parallel import Task, WorkManager
from GaudiPython import AppMgr, PyAlgorithm

importOptions('$BRUNELROOT/options/Brunel-Default.py')
importOptions('$BRUNELROOT/options/DC06-Files.py')
ApplicationMgr( OutputLevel = ERROR, AppName = 'PyBrunel')
appMgr = AppMgr()
appMgr.initialize()

class Brunel(Task):
    def initializeLocal(self): pass

    def initializeRemote(self): pass

    def process(self, file):
        appMgr.evtsel().open([file])
        appMgr.run(10)

    def finalize(self): pass

from Files import digi_files as files
if __name__ == '__main__':
    task = Brunel()
    wmgr = WorkManager(ncpus=4)
    wmgr.process( task, files[:4])
```

Common
initialization in
parent process

Process 10
events in each
file

Main program

Memory Usage[¶]

	#	code		data		sum	total [§]	total/N
		private	share	private	share			
Single	1	157	5	336	0	498	498	498
Single	2	4	167	339	0	510	850	425
Single	4	0	171	321	0	492	1462	365
Fork ^a	4	0	149 [†]	313	16	478	1454	363
Fork ^b	2	0	116 [†]	180	162	458	742	371
Fork ^b	4	0	117 [†]	178	158	453	1105	276
Fork ^c	4	0	86 [†]	118	196	400	852	213
Fork ^{c,d}	8	0	86 [†]	118	196	400	1320	165

[¶] Physical memory usage in Mbytes, [§] including parent process

^a no initialization in parent process, ^b all services initialized in parent,

^c one event run in parent, ^d only estimated

[†] part of the code only mapped in parent process

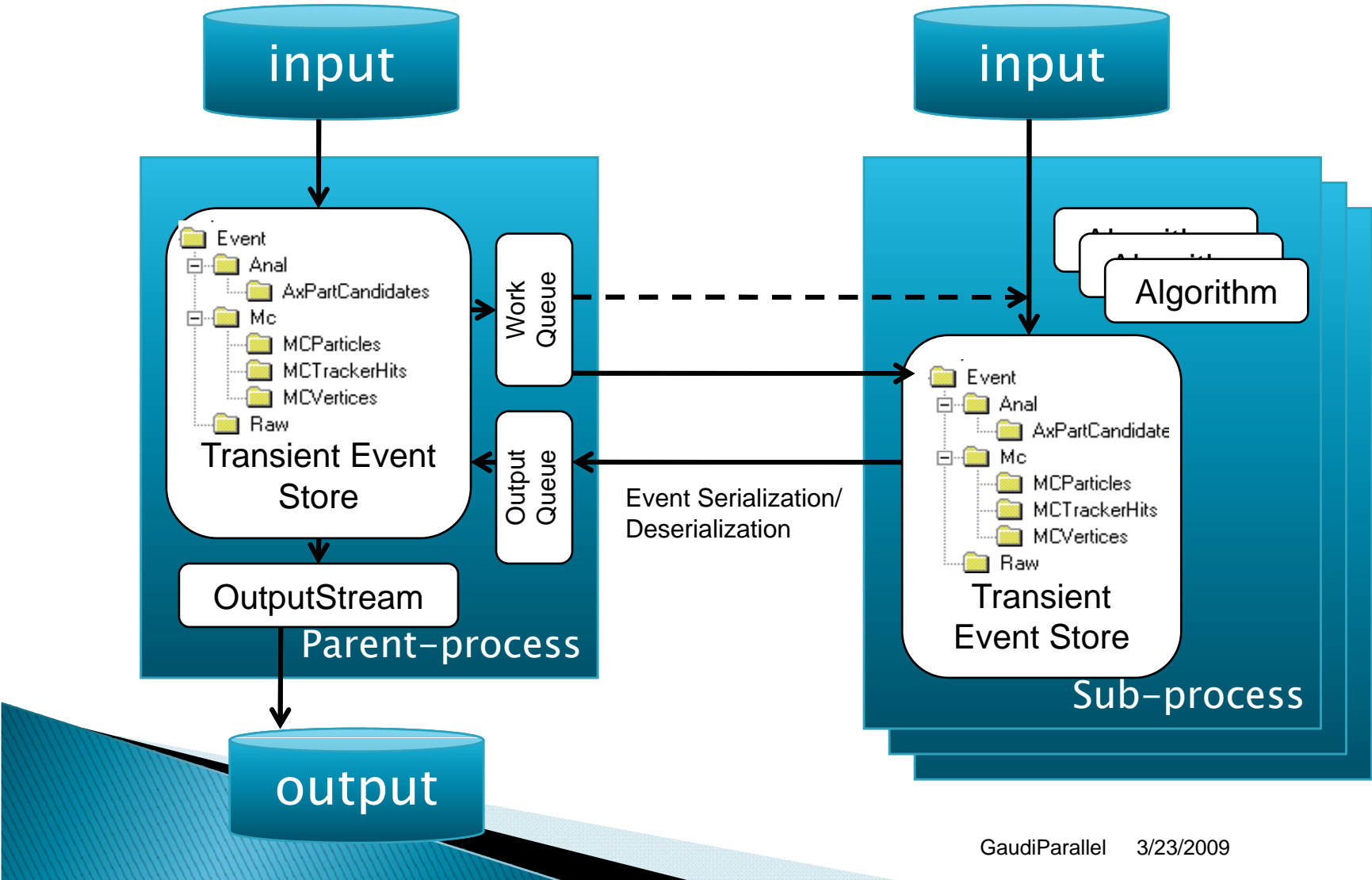
Shared Memory Comments

- ▶ The shared memory pages after processing the 10th event on different files can be as much as 70% (code+data) and 62% (data only)
 - Does not get degraded after processing many events (measurements from J. Arnold, CERN/Openlab)
- ▶ On a 8-core computer running 8 independent reconstruction instances requires 2.7 GB, while running them with *fork()* requires 1.4 GB
- ▶ In this measurements the output file(s) not taken into account

Model fails for “Event” output

- ▶ In case of a program producing large output data the current model is not adequate
 - Data production programs: simulation, reconstruction, etc.
- ▶ Two options:
 - Each sub-process writing a different file, parent process merges the files
 - Merging files non-trivial when “references” exists
 - Event data is collected by parent process and written into a single output stream
 - Some additional communication overhead

Handling Event Input/Output

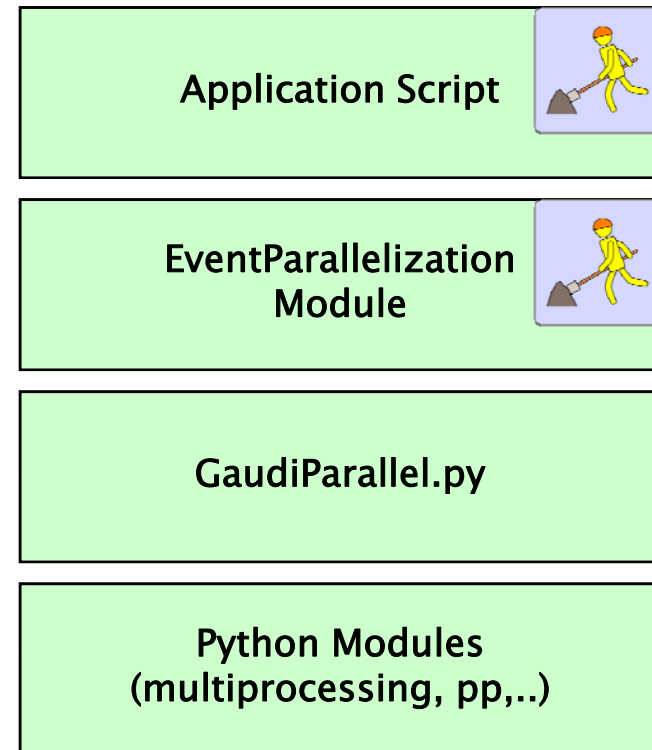


Handling Input/Output

- ▶ Developed event serialization/deserialization (TESSerializer)
 - The entire transient event store (or parts) can be moved from/to sub-processes
 - Implemented by pickling a TBufferFile object
 - Links and references must be maintained
- ▶ Modifying 'processing' python module to add input and output queues with additional thread
- ▶ Extending 'Task' with new method *processLocal()* to process in parent process each event [fragment] coming from workers

Layered Solution

- ▶ GaudiParallel.py offers a generic module for Gaudi parallelization (multi-core + cluster)
- ▶ EventParallelization.py will specialize for events
- ▶ The Application Scripts will be customized to each program type (simulation, reconstruction, analysis)



Summary

- ▶ Python offers a very high-level control of the applications based on GAUDI
- ▶ Very simple adaptation of user scripts may led to huge gains in latency (multi-core)
- ▶ Gains in memory usage are also important
- ▶ Several schemas and scheduling strategies can be tested without changes in the C++ code
 - E.g. local/remote initialization, task granularity (file, event, chunk), sync or async mode, queues, etc.