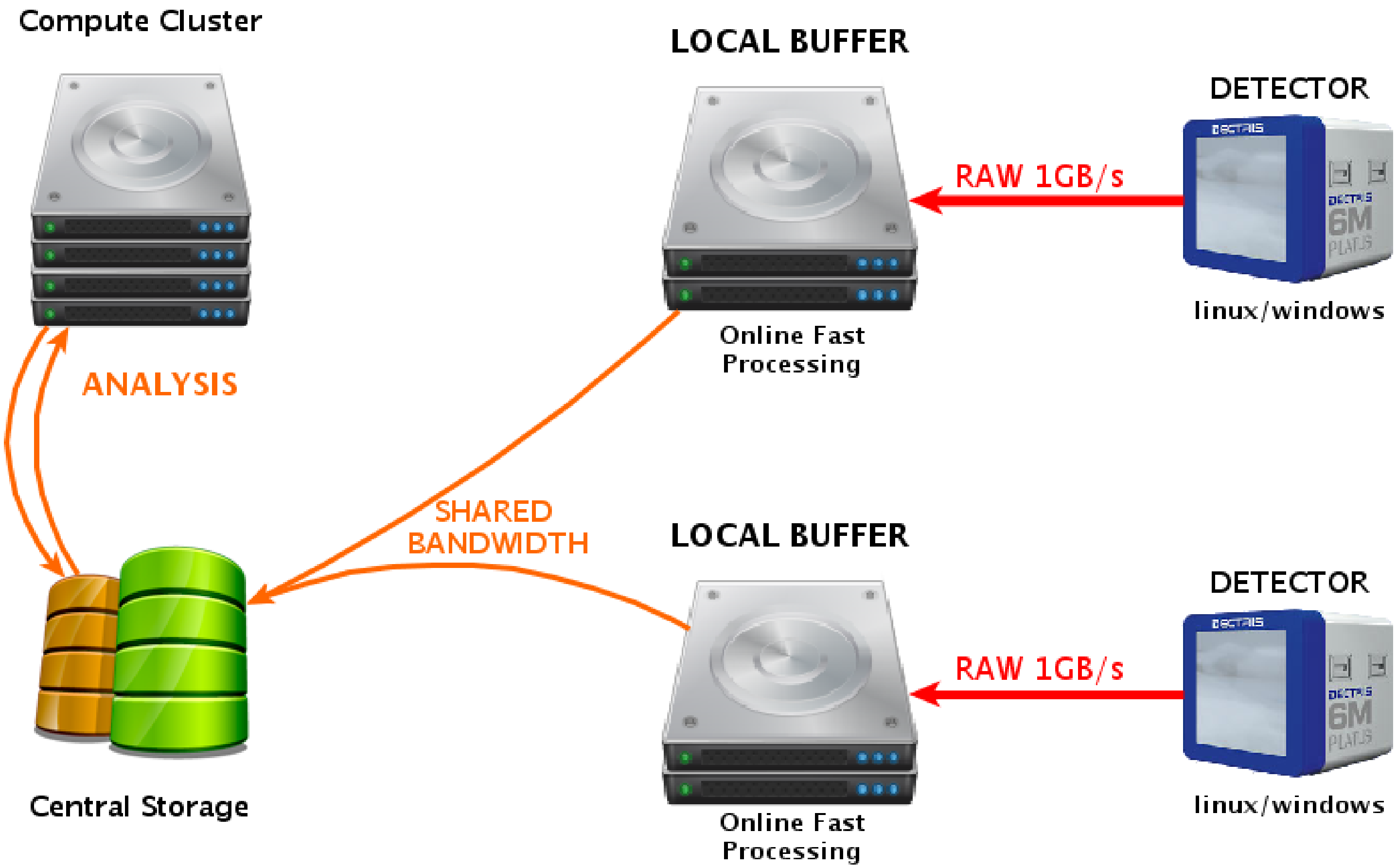
How can a detector saturate a 10Gb link through a remote file system

- The requirements we have focused on:
  - Dedicated machine for buffering detector’s data and for fast online processing
  - Sufficient storage to hold 2 days of experiments data (for the weekend)
  - link from online data processing PC to central storage to write or read results
  - list 10000 files < 3s


Compute Cluster
LOCAL BUFFER
DETECTOR
RAW 1GB/s
linux/windows
Online Fast Processing
ANALYSIS
SHARED BANDWIDTH
LOCAL BUFFER
DETECTOR
RAW 1GB/s
linux/windows
Central Storage
Online Fast Processing

Issues we are facing

- Detectors seen as a single threaded process:
  No Parallelization possible
- Writing and reading from the same disks has high impact on disk performance.
- Difficult to prioritize clients accessing the same remote filesystem.

Remote file systems tested and fine tuning

- On the hardware side:
    - Raids cards
    - Network cards
- On the system side:
    - Linux flavors: centos, redhat, debian, ubuntu.
    - Virtual memory or tcp fine tuning
- On the block side:
    - File system types: xfs, ext4
    - Network block device or iscsi
- On the file side:
    - CIFS, NFS (2,3,4)

The hardware choosen

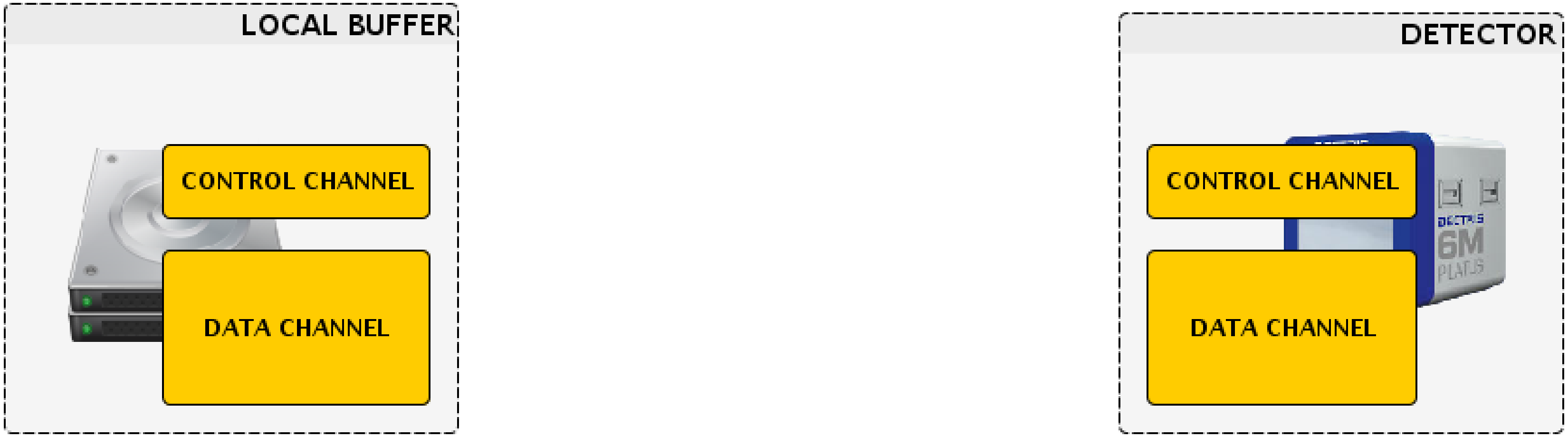
- The machine choosen (DELL R720xd) has:
  - 24 drives for a total capacity of 20T
  - 2x6 cores
  - RAM will depend on the needed transfer rate (ramdisks)
  - Up to 6x10Gb optical links
  - Local write speed with ext4 raid6: **> 1200 MB/s**

Results

- They are bad !!
  - Block devices (ndb and iscsi) gives the best results
    Around **500 or 600 MB/s** with 6MB files
    But not as flexible as NFS and still under 1GB/s
  - CIFS, NFS reach **400 MB/s** in best cases …
  - Parallel file system are more complex to deal with.
    Not because of their own complexity, but because we deal with a wide variety of detectors, and most of the time installing a heavy client like the GPFS one **is a problem**.
    Moreover we have been told that performance for a single threaded application is not that good (below 1GB/s) (not tested)

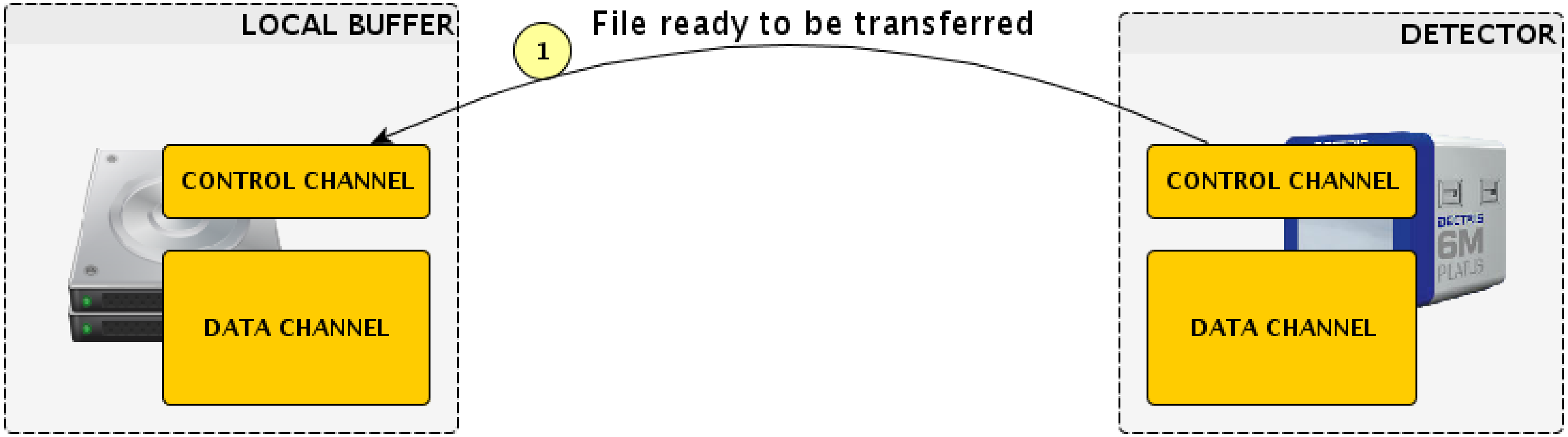
The HTTP case

- We have implemented our own solution.
- 2 Channels as this is done in FTP



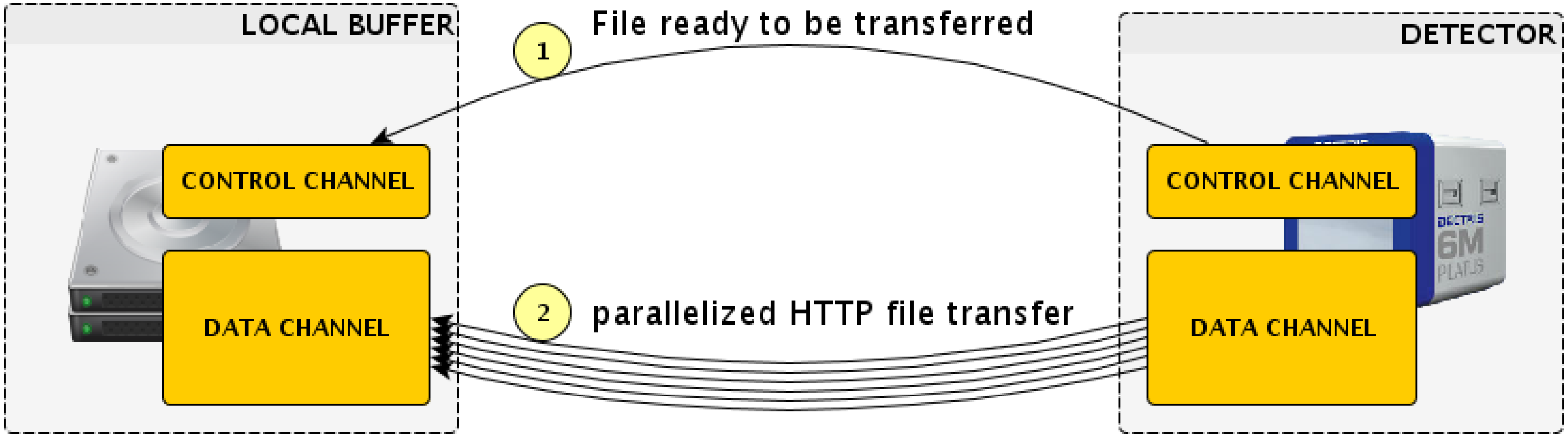
The HTTP case

- Step1: Detector tells when a file is finished to be written and ready to be transferred.



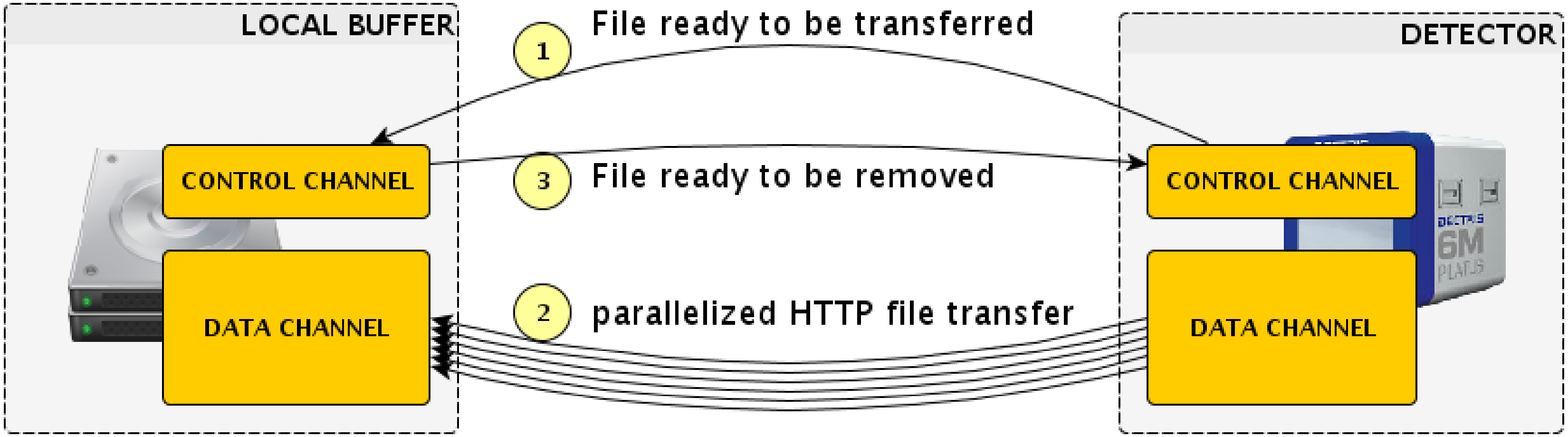
The HTTP case

- Step2: we **parallelize** file transfers with persistent http connections



The HTTP case

- Step3: once transfer is done, we remove files from the detector.



Optimisations (Linux)

- On the detector side:
  - Ramdisk to have a minimum impact on disk perf.
  - Lighttpd for efficient http transfer with minimal memory footprint
  - Inotify to know when a file is fully written (sadly not available in windows world).
  - Daemontools to monitor all this.
- On the Local Buffer side:
  - Unix named pipes to implement FIFO queues.
  - Libcurl to get files through http and keep connection opened.
  - Ionice to prioritize down sync to central storage.
  - Daemontools to monitor all this.

Optimisations

- Python “twisted” : an event-driven networking engine.
  - Pretty fast engine
  - Used on the control channel (linux **AND** windows)
  - Thread safe, and can handle multiple clients

Results and advantages

- With 1 link we almost reach the limit: **900MB/s**.
  we need to keep some bandwith for recovery purpose.
- With 2 links we reach the local buffer raid card limit: **1200MB/s**.

- **Advantages:**
  - As this is an asynchronous transfer, we can break the link and reboot the local buffer while acquisition is on going.
    Ramdisk should be big enough on detector !!
  - Compared to NFS client it is much lighter !
    40 to 50% of 1 cpu at 900MB/s whereas NFS consumes more CPU and generates more IOwaits at much lower data rates.

Interoperability / Road map

- Windows 7 (32/64 bits) version for the client.
  - Local buffer machine will still be on Linux
  - Local buffer machine should be able to talk to linux/windows detectors without modification
- Easy switch in case of local buffer failure.
- Online data analysis on local buffer machine
  - Data flow inside the machine so we can use ramdisks
  - Keep central storage in sync
  - Strategy for Raw/Temporary/Computed data
  - Backup Strategy
- Ability to gather data from 2 or more detectors.

Road map

- Requirements not implemented yet:
    - mount user storage on online data processing PC (probably impossible: LBS must insure aquisition !)
    - automatic export of analysed data to user's export medium. (need to compare the 20MB/s of a USB2 drive, and 1GB/s of the detector ...)

THANKS !