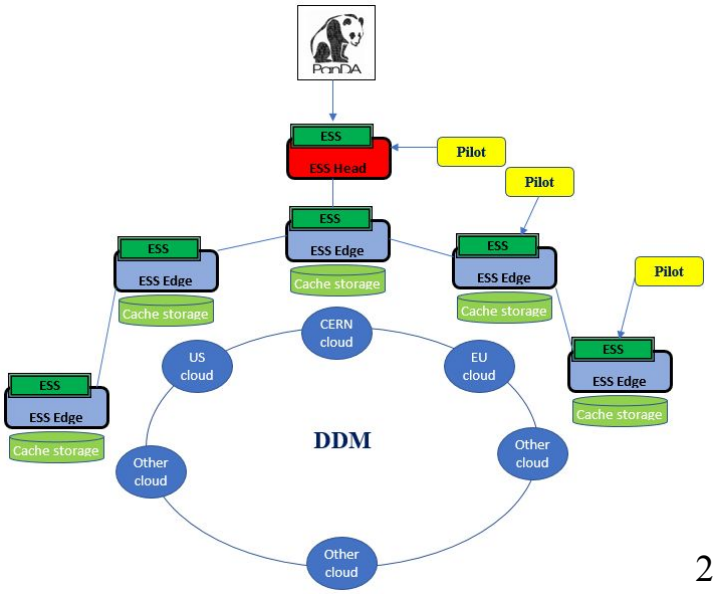
iDDS
– intelligent Data Delivery Service –

Tadashi Maeno (BNL)
on behalf of iDDS/ESS project

Introduction

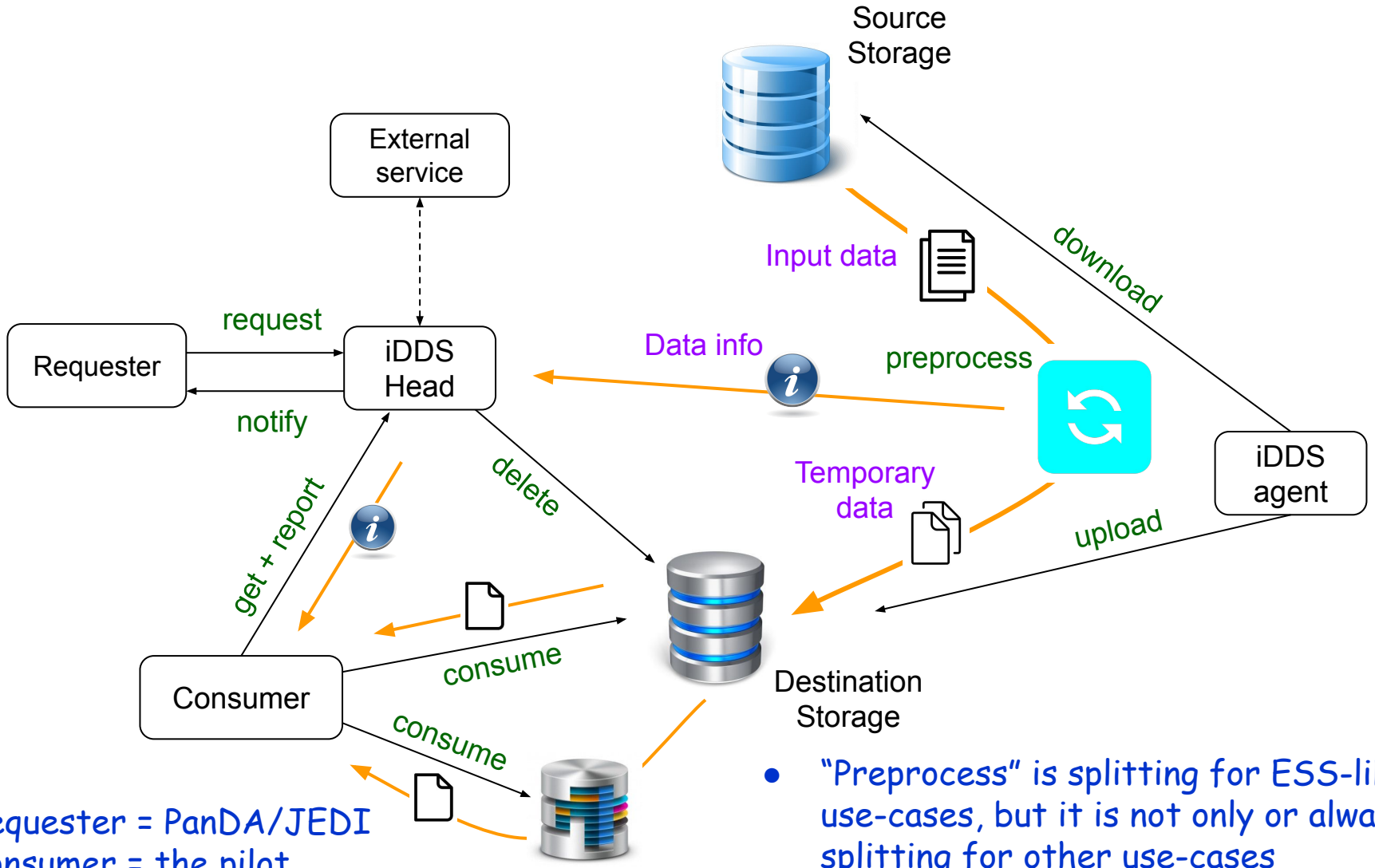
- The first prototype of Event Streaming Service (ESS) reported in ADC Jumboree
  - Rapid development for proof of concept
  - Workflow of ESS
    1. Panda sends a request to ESS Head Svc
    2. ESS Head Svc forwards the request to a regional ESS Edge Svc
    3. The Edge Svc downloads files and splits them to smaller segment files
    4. Segment files are uploaded to a cache storage like object store
    5. ESS Head Svc notifies Panda so that pilots get started
    6. Each pilot gets event ranges from PanDA and resolves them to TURLs of segment files by checking with ESS Head or Edge Svc
    7. The pilot or payload downloads segment files from the cache storage
  - Issues in the prototype
    - Only fine-grained processing is considered
      - Enhancement for other use-cases could be cumbersome
    - Too ATLAS/PanDA specific
    - A dedicated CPU cluster for each regional ESS Edge Svc would be required
      - 1 sec to produce 1 segment file per event → ~11 days for 1M event input data on SCORE
      - Could be expensive to scale



Introduction (cntd)

- Requirements
  - Experiment agnostic
  - Flexibility to support more use-cases and backend systems
  - Easy and cheaper deployment
- iDDS : intelligent Data Delivery Service
  - A joint project between IRIS-HEP and ATLAS
    - CMS use-cases should be taken into account from the beginning of the initial design stage
  - An intelligent service to preprocess and deliver data to consumers
    - Data = files, file fragments, file information, or sets of files
    - Not a storage, WFMS, or DDMS
  - Generalization of ESS concept/workflow
    - Fine grained processing is one of major use-cases
    - More use-cases even in ATLAS (to be shown later)
  - Workflow with iDDS
    1. A requester sends a request to iDDS Head
    2. iDDS agent downloads data from a source storage and preprocesses them to produce temporary data
    3. Temporary data are uploaded to a destination storage
    4. iDDS Head notifies the requester, so that consumers get started
    5. Consumers get information of temporary data from iDDS and consume those data from the destination storage or via a cache storage
    6. Temporary data are deleted if no further usage is foreseen

Generalized Workflow with iDDS



- Requester = PanDA/JEDI
  Consumer = the pilot
  in ATLAS context
- Source and destination storages can be the same

- "Preprocess" is splitting for ESS-like use-cases, but it is not only or always splitting for other use-cases
- "Preprocess" and Consumer could asynchronously run in parallel for urgent processing

Types of Preprocessing

Not exclusive. Can be combined

- Splitting
  - To split files into small segment files
- Concatenation
  - To concatenate or merge small files into large files
- Transformation
  - To convert from a cold storage optimized format to a warm data format, change compression level on the fly, marshal internal data structure to a simplified GPU/ML friendly format, etc
- Thinning
  - To remove objects from data which are unnecessary for subsequent processing
- Extraction
  - To obtain information from files which are used in subsequent processing. For example, mapping from event numbers to offsets + data chunk sizes in files
- Filterling
  - To select a subset of data used in subsequent processing
- Pre-staging
  - To pre-stage files from slow disk/tape systems to warm storages
- Mixing
  - To combine multiple source data to a single data

Analogy to Media Streaming

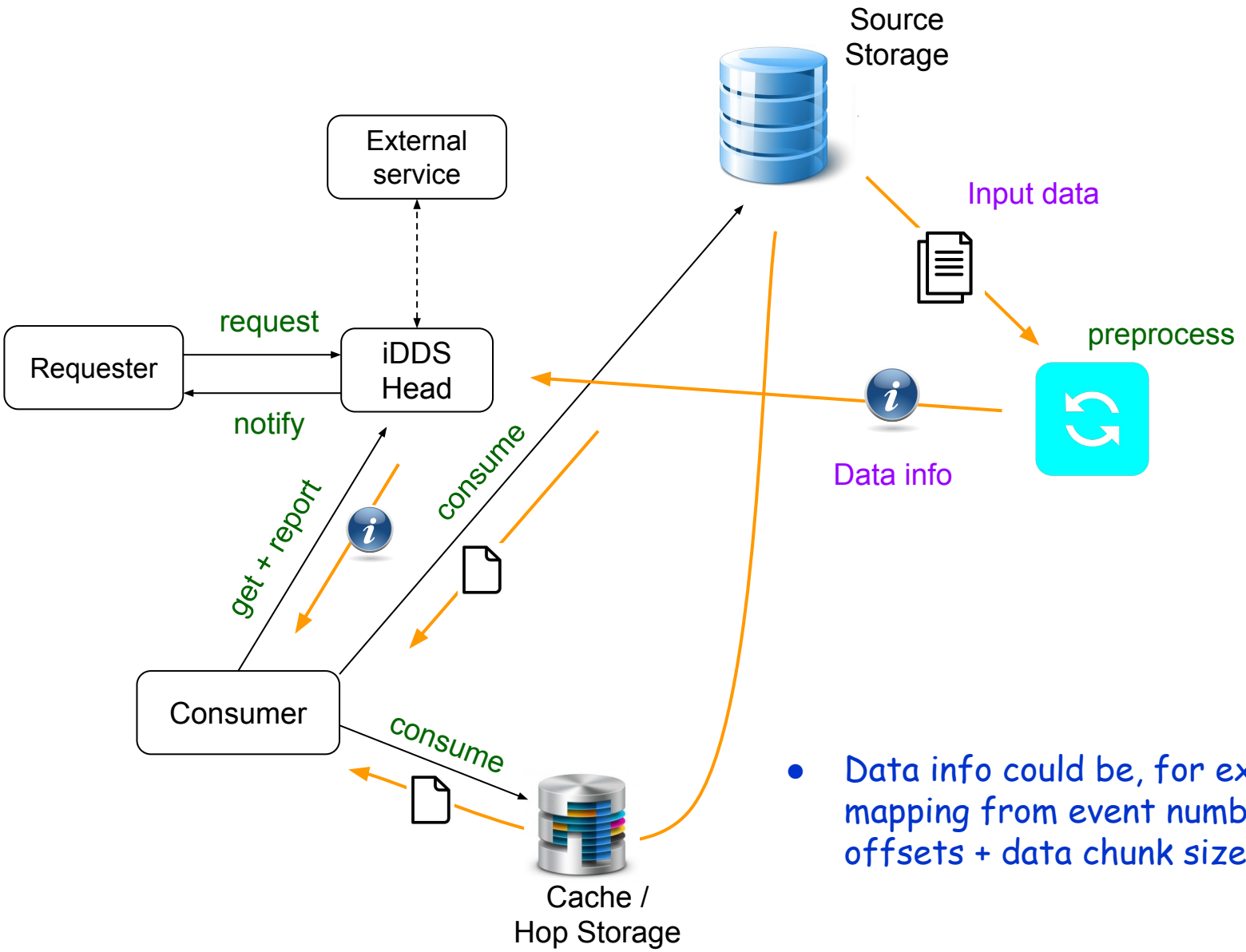
1. An influencer uploads a RAW file to Amazon S3 and sends a request to iDDS
   - Original data = the RAW file
   - Source storage = Amazon S3
2. iDDS runs an agent close to Amazon S3 to convert the RAW file to many small TS files
   - Preprocessing = encoding with H.264 + AAC (transformation) + MPEG-2 transport stream segmentation (splitting)
   - Temporary data = TS files
3. TS files are uploaded to a storage behind an HTTP server
   - Destination storage = the storage behind HTTP
4. The influencer gets notified, so that he/she invites followers
5. Each follower launches a web browser and gets an m3u8 file from iDDS
   - Information of temporary data = the m3u8 file
   - Consumer = web browser
6. The web browser asynchronously downloads TS files through HTTP server + CDN/proxy to play video and audio
   - Cache = CDN/proxy
   - Subsequent processing = playing video and audio

≃ Fine-grained processing with temporary data

Keys of iDDS

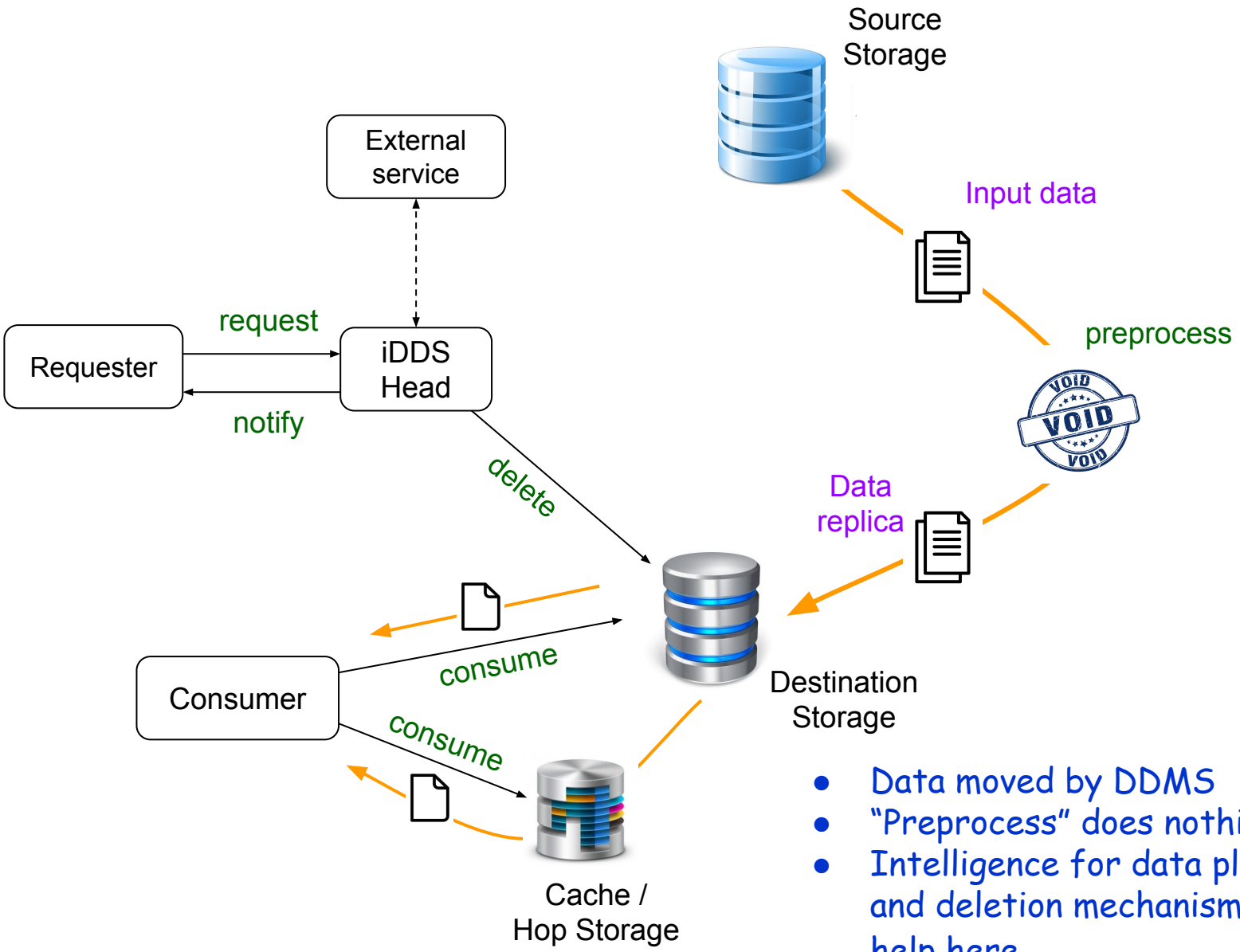
- Plugin structure
  - To be experiment agnostic and to be easily extended to newly emerging use-cases
- Preprocessing agents
  - Flexibility
    - To support various (new) types of preprocessing
  - Potentially very resource intensive
    - The ESS prototype ran agents locally on the same node where ESS Head Svc was running
      - FTS like deployment model : A dedicated resource pool is required per ESS node
  - One possibility is to delegate execution of preprocessing to WFMS
    - Resources, data transfer mechanism, and workload scheduling for free
      - PandaMover like deployment model
  - Data reduction by preprocessing is important to decrease data traffic over WAN and improve overall data processing throughput
- Scheduling of preprocessing agents
  - Mostly rely on WFMS
  - Additional hints for smarter brokerage, such as consumer's location, reduction factor of preprocessing, priorities of subsequent processing, and so on
- Placement of temporary data
  - Two obvious placement policies: Close to the original data or to consumers
  - Intelligent data placement based on scientific data contents, current network/storage metrics and prediction, and subsequent processing
  - To leverage cache hierarchy in data pulls

Fine-grained Processing without Temporary Files



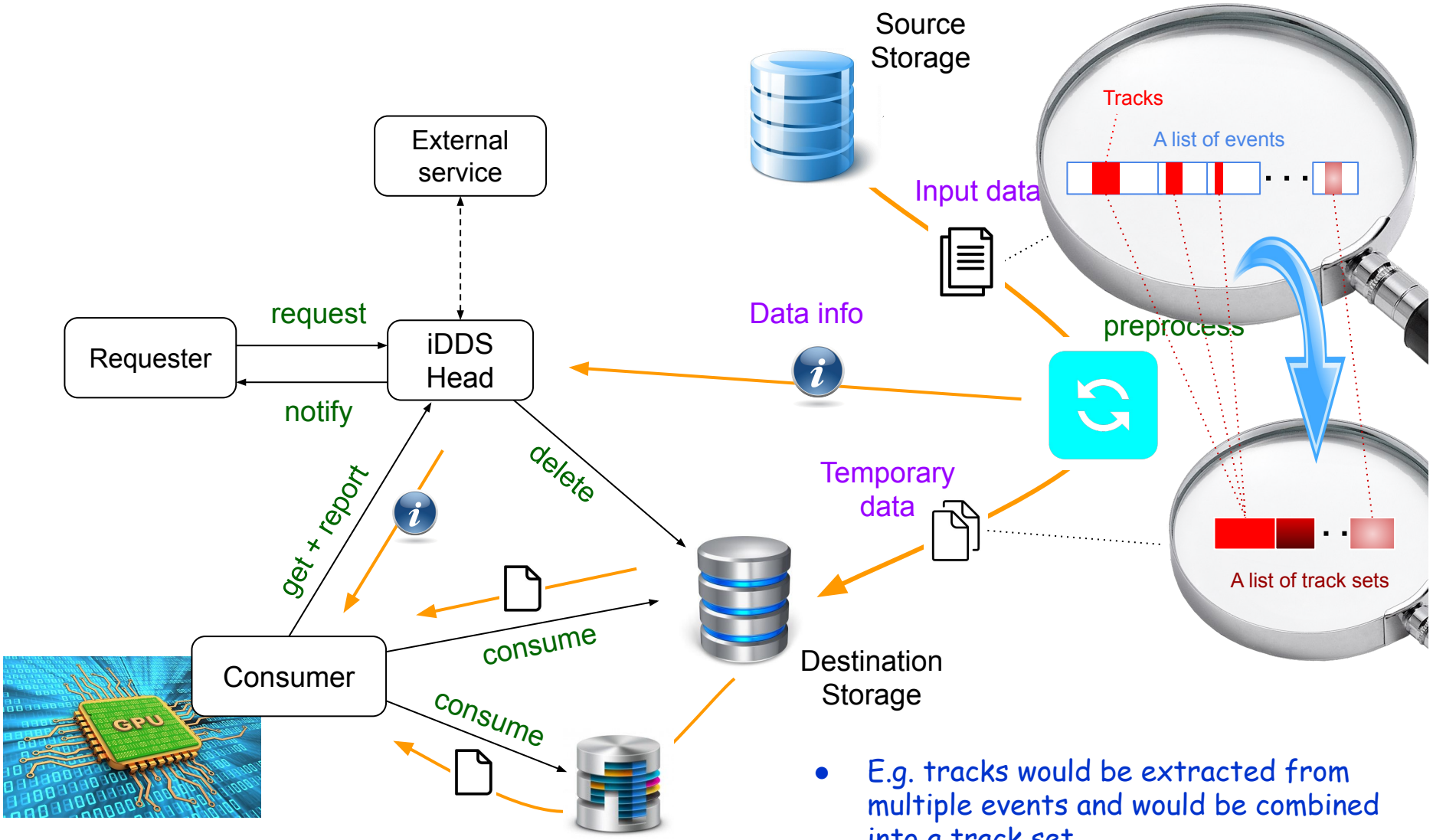
- Data info could be, for example, mapping from event numbers to offsets + data chunk sizes in files

Dynamic Data Placement (PD2P) or Tape Carousel



- Data moved by DDMS
- "Preprocess" does nothing
- Intelligence for data placement and deletion mechanism could help here

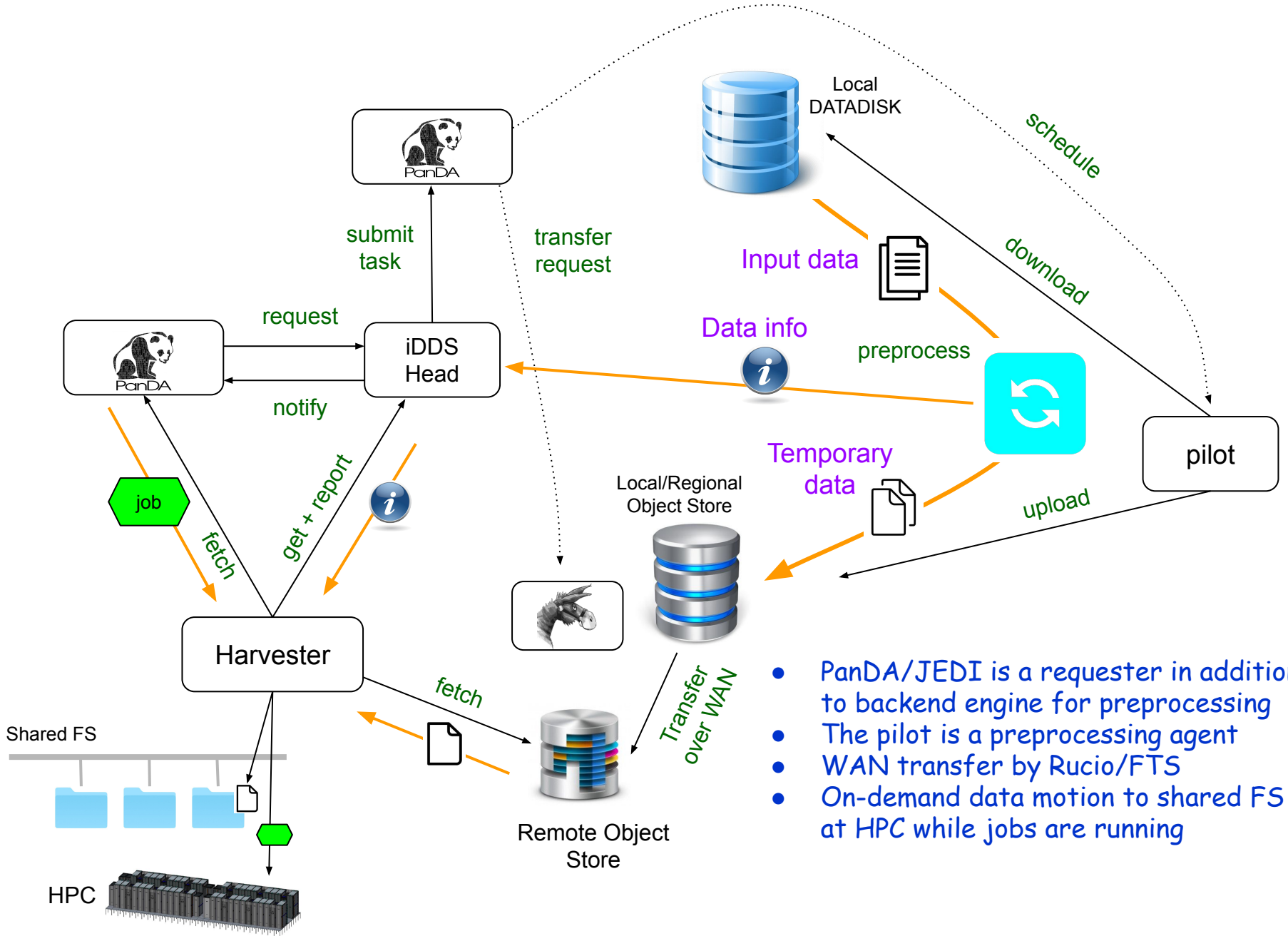
Marshalling to GPU/ML-friendly Format



- One-time marshalling by iDDS rather than repeating real-time marshalling in each consumer per iteration

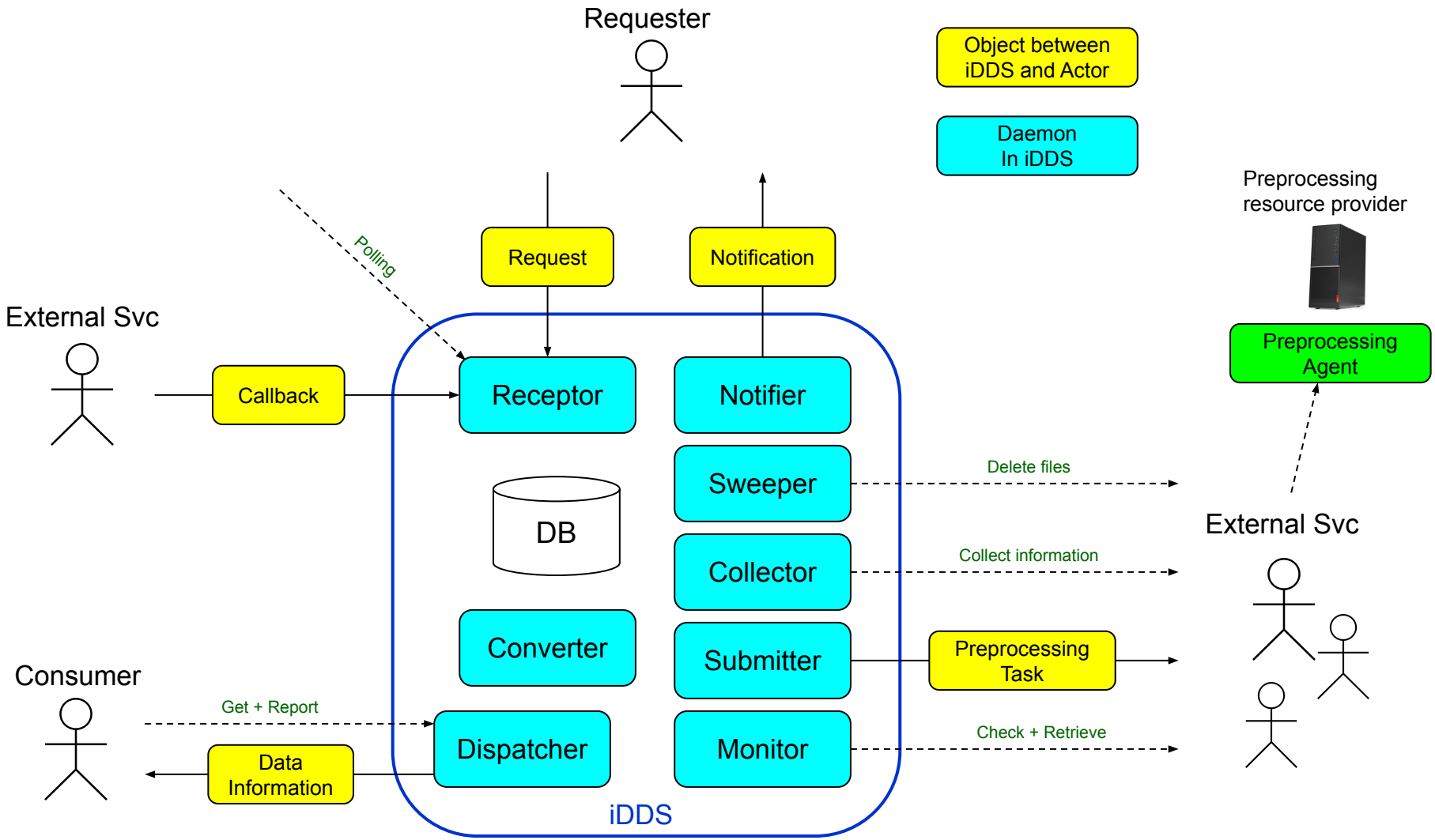
- E.g. tracks would be extracted from multiple events and would be combined into a track set
- GPU would process per track set instead of per event

iDDS + Harvester for ATLAS HPC



- PanDA/JEDI is a requester in addition to backend engine for preprocessing
- The pilot is a preprocessing agent
- WAN transfer by Rucio/FTS
- On-demand data motion to shared FS at HPC while jobs are running

iDDS Architecture (Preliminary)



Current Status and Plans

- Conceptual modeling of iDDS has been done based on the assessment for ESS prototype
- Requirements document to be finished by the end of the month
  - Full architecture design
  - Database structure
  - Definition of daemons, agents, and API
- Development model to be established
  - Coding rule
  - HSF hosted github repository
  - Regular meeting
  - Mailing list hsf-event-processing-wg@googlegroup.com
  - Manpower
  - Milestones
  - …
- Details will be discussed in WFMS TIM in Ljubljana next month

Appendix

- ESS prototype
  https://docs.google.com/presentation/d/1mysBcl0CA69Q7mgYLs9xcs-qiL0i96amNLq2oSCYSno/edit?usp=sharing
- ESS git repository
  https://github.com/PanDAWMS/ESS
- JLab meeting notes
  https://docs.google.com/document/d/1S-BUncPBZmbTCffeI3BVevtorLOGm_o8riAbc_Y_S08/edit#heading=h.g4t3nsmnzgch
- iDDS requirements document (to be finished)
  https://docs.google.com/document/d/1asIefhqvGfkD6aiWH9QgGN7np_tIC_3m7CfDaocGeJw/edit?usp=sharing