dCache-Storage Infrastructure at DESY

Meeting with Colleagues from SESAME

Christian Voß
DESY, 14th August 2025

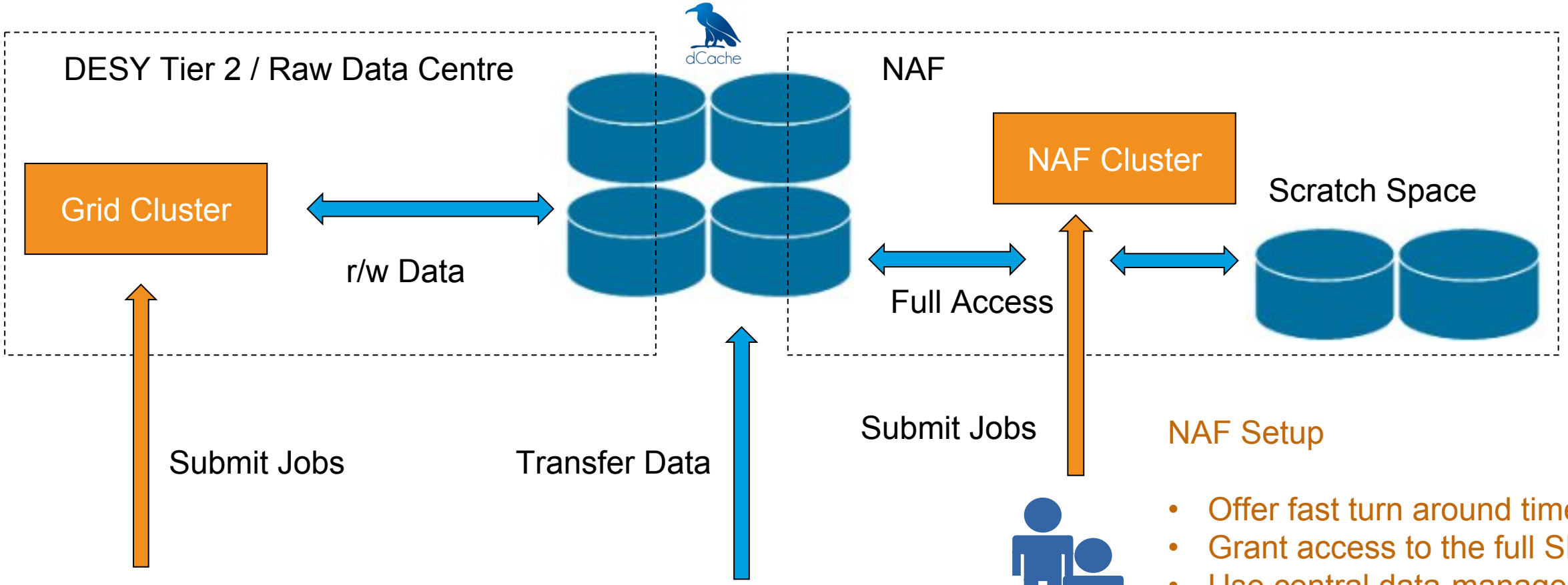
HELMHOLTZ

DESY.

Paradigm: HEP Analyses are Data Driven

As Underlying Principle for dCache Storage Architecture at DESY

- Almost all HEP data analyses require access to large amounts of data



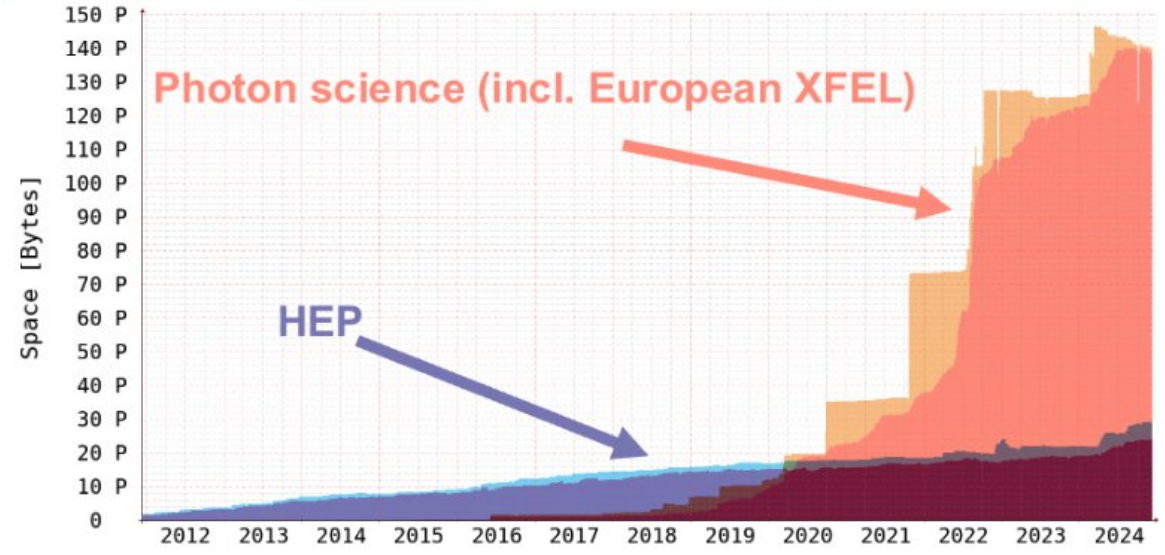
NAF Setup

- Offer fast turn around times
- Grant access to the full SEs
- Use central data-management of experiments to store data at DESY

Mass-Storage for Different Communities

dCache as Central Mass Storage

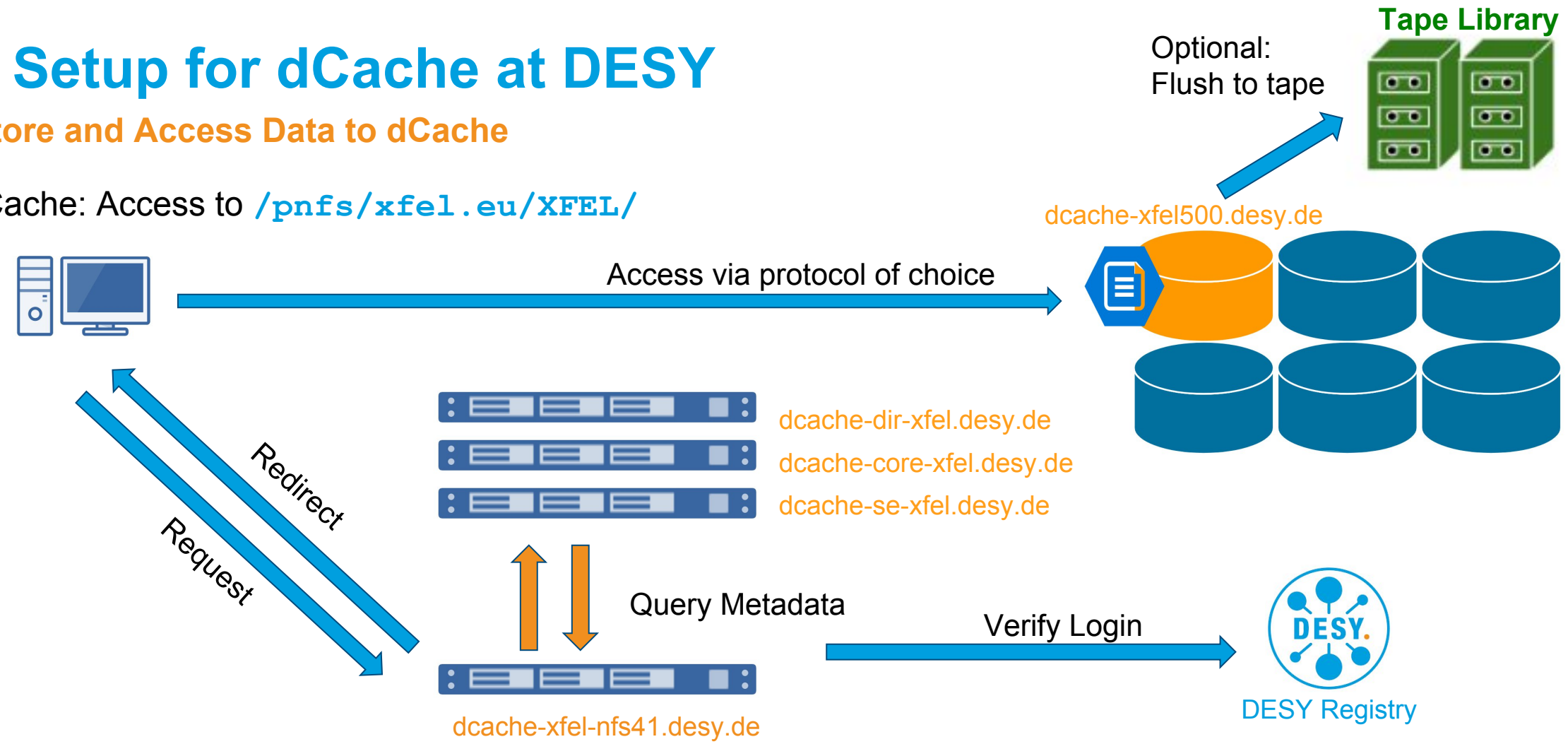
- Central element in overall storage strategy
- Collaborative development under open source licence by
  - DESY
  - Fermilab
  - Nordic E-Infrastructure Collaboration (inoffically NDGF)
- Particle Physics in general
  - In production at 9 of 13 WLCG Tier-1 centres
  - In use at over 60 Tier-2 sites world wide
  - 75% of all remote LHC data stored on dCache
  - In addition: Tevatron and HERA data
  - All smaller DESY experiments store data in dCache
- Photon Science
  - Raw-Data for all DESY light sources
  - Long-term archival



Basic Setup for dCache at DESY

How to Store and Access Data to dCache

- Use dCache: Access to `/pnfs/xfel.eu/XFEL/`

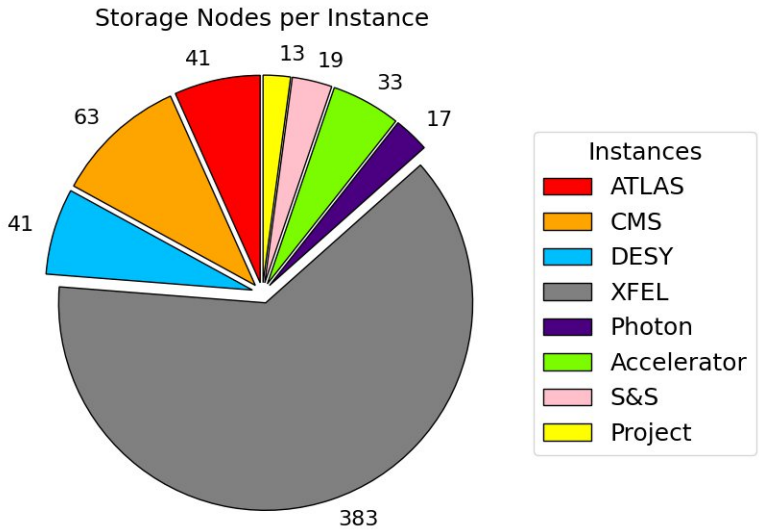


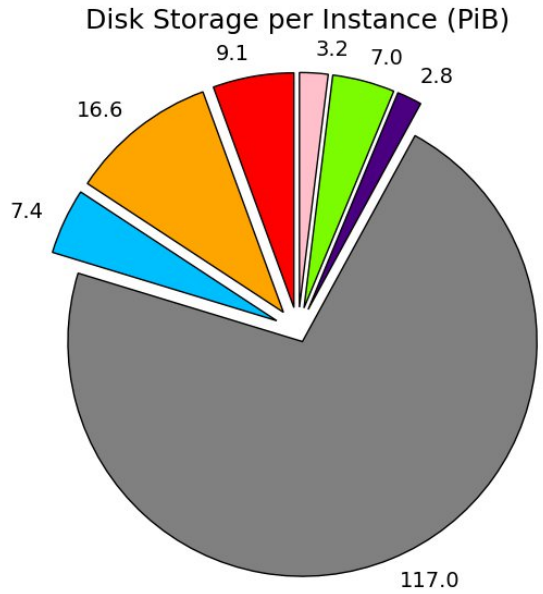
- Access done through doors: several load balanced door for each protocol to ensure availability
- Access controlled via Grid-certificates or by tokens and POSIX (NFS@NAF and Maxwell)
- Data streamed to/from pool, never through doors: allows horizontal scaling
- Namespace is uniform and independent of protocol

dCache Instances at DESY

Overview

- Operate singular instances for specific large communities
- **Three dedicated HEP instances with large central and local user space**
  - ATLAS, CMS, DESY
  - DESY instance serves many smaller HEP experiments: Belle II, ILC, ALPS II,...
  - DESY instance serves IT as well, e.g. storing dCache telemetry data
  - Fully integrated into global HEP computing grid infrastructure
  - Full access on all DESY IT compute clusters
- Dedicated instance for machine DAQ data with special requirements
  - XFEL and FLASH LINACS store DAQ data in a 5PiB ring-buffer during operations
  - Smaller experiments sync DAQ and RAW data close to data taking
  - Analysis done on DESY IT compute clusters as well dedicated resources
- Photon Science
  - Raw-Data for all DESY light sources
  - Large buffer for data analysis, recalibration and reduction campaigns for XFEL
  - Tape only for PETRA III/FLASH therefore very small disk buffer, namespace hidden
  - Store only packed data for PETRA III/FLASH due to small file sizes

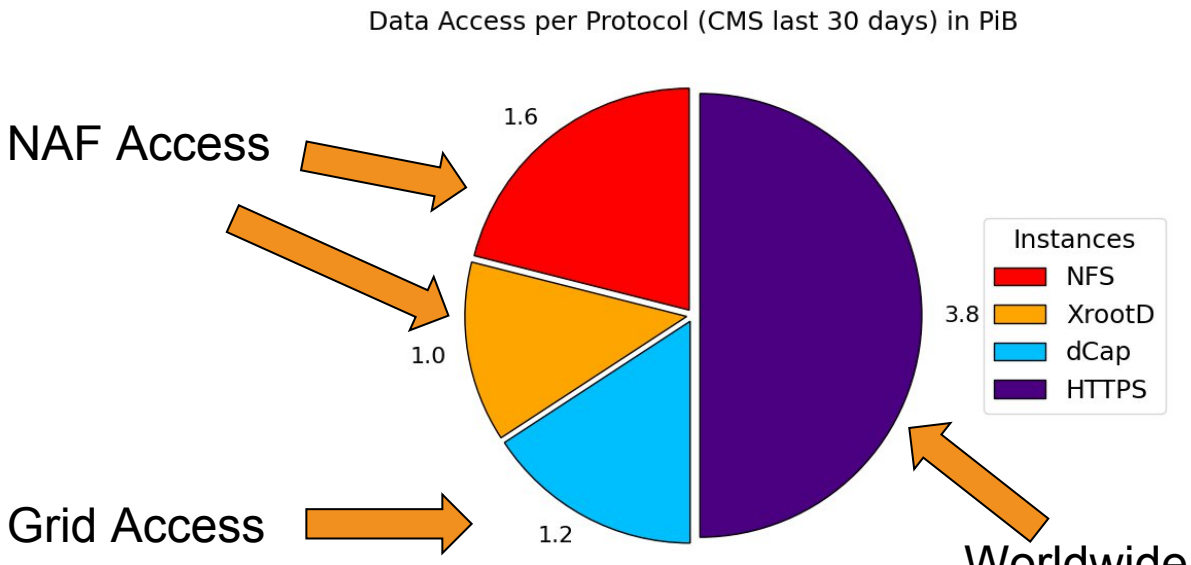




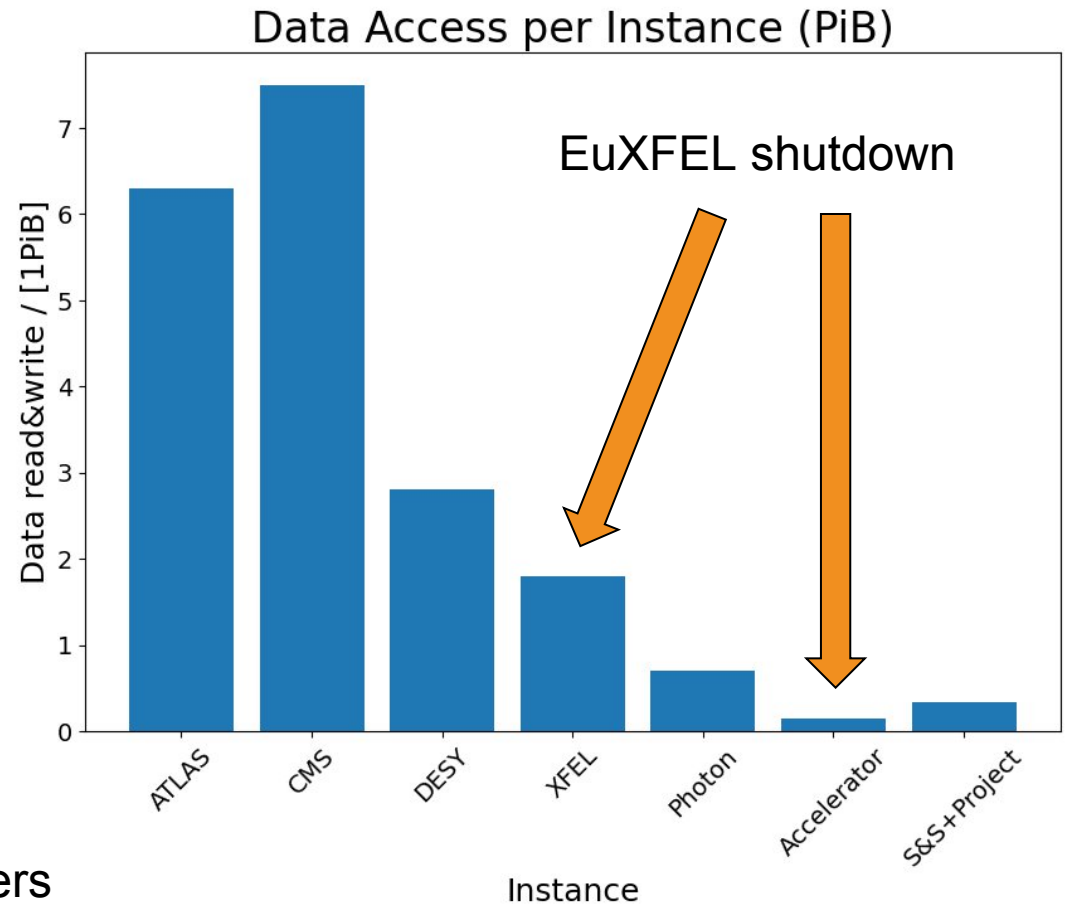
dCache Instances

Data Access

- Full access to stored data from the compute clusters for most instances as well as from other large centres world wide
- Photon instance: namespace mostly hidden from users and PETRA III/FLASH user data is staged from tape to the GPFS
- HEP as the largest user community, rotate the local stored data on average every two months
- Different use cases see different usage of available protocols

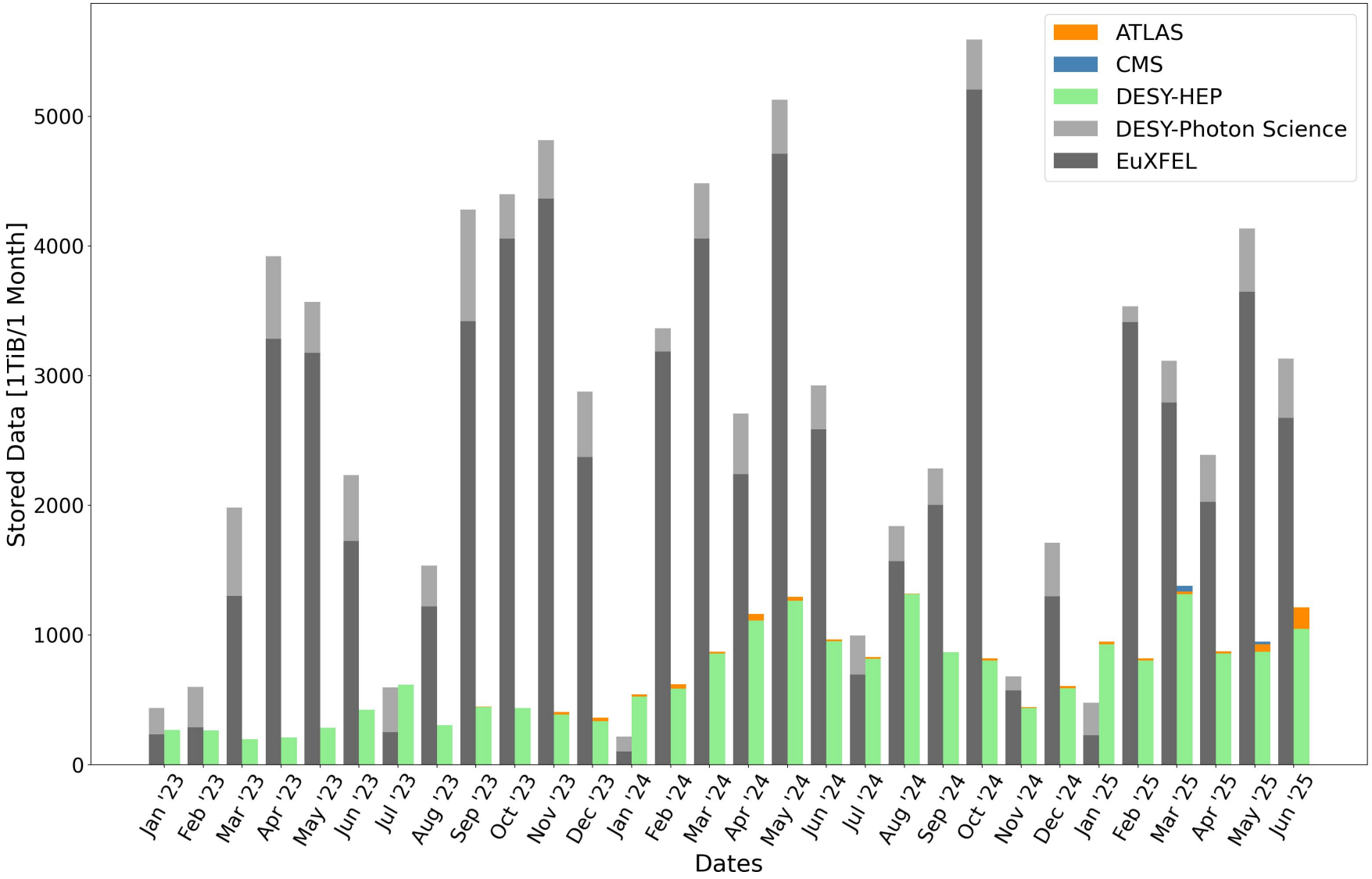


Traffic for last 30 days



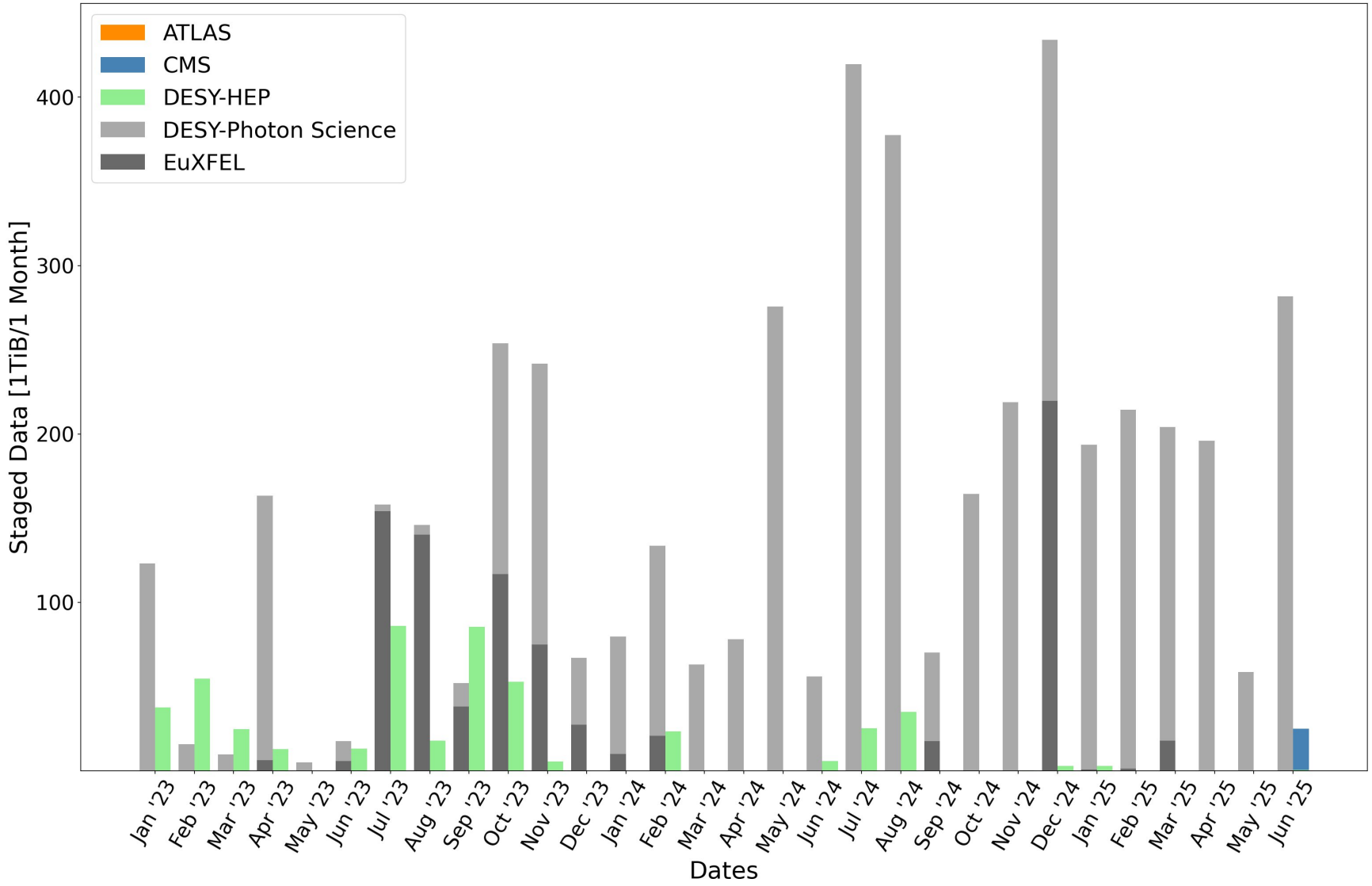
dCache And Tape

Stores for all Communities



dCache And Tape

Re-Stores for all Communities



Interaction between Tape and dCache

Staging and Quality of Service

Staging and Pinning

- By default: accessing a file triggers a stage
- Convenient: user needs no knowledge about locality
- Historically data was stages through
  - Dedicated NFS based interface (using touch commands)
  - Storage Resource Manager (SRM) using Grid authentication and authorisation
  - **Today: Staging through REST-API with flexible logins**
- Accessing the file via NFS leads to hanging nodes
- Random, accidental access can overload tape library
- **Put in place stage protection allowing only a selected few to stage**
- **Staging comes with a disk lifetime (pin)**

- Define desired state for any file (can be very general)
  - File should be flushed to tape, but not remain on disk
  - File should be flushed to tape and remain on disk
  - File should only be on disk, but with n copies
- These states define the quality of service
- Service inside dCache that ensures this state
- Use REST-API to modify the state triggering flushes or stages, cleaning disk locations: stays until changed again
- Requires no tracking of pin lifetimes
- **Policy engine in place that can take complicated workflows**: 2 copies on disk for 6 months, 1 disk copy for three years, tape only after three years

Thank you, any Questions

Managing Heterogenic Pools

Determine Pool Sizes During Configuration Runs

- DESY IT uses Puppet as central configuration management tool (managed via Git repositories)
- Try to automate as much as possible especially on pools
- Puppet allows custom facts to me calculated on specific hosts
- Read out the mounted file systems (consistent rules on nameing and local mount points on all pools)

```
pools => {
  xfel500-01 => {
    MOUNTPOINT => "/dcache/dcache-xfel500-01",
    SIZE => 572009121579008
  }
}
```

- Use the information to generat pool layout and setup files
- Configure NFS port depending on pool ‘number’

```
[dcache-xfel500-01Domain]
[dcache-xfel500-01Domain/pool]
pool.name=dcache-xfel500-01
pool.path=/dcache/dcache-xfel500-01
pool.wait-for-files=${pool.path}/data
pool.mover.nfs.port.min=24001
pool.mover.nfs.port.max=24001
pool.tags=hostname=dcache-xfel500
```

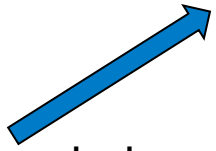

```
Facter.add(:pools) do
  confine :kernel => "Linux"
  confine :hostgroup_0 => "dot"
  confine :hostgroup_1 => "pool"

  setcode do
    pools = {}

    Facter.value(:blockdevices_array).each do |device|
      lsblk_command = 'df --output=source,target,size | grep ' + device
      pool = {}
      pool_info = Facter::Core::Execution.execute(lsblk_command)
      pool_info = pool_info.split(' ')
      if pool_info[0]
        pool_name = pool_info[1].split("/")[2]
        pool_name.slice! "dcache-"
        pool['MOUNTPOINT'] = pool_info[1]
        pool['SIZE'] = pool_info[2].to_i * 1024
        pools[pool_name] = pool
      end
    end
    pools
  end
end
```

Managing dCache Services Through Puppet

Keep Most of the Configuration in Puppet

- Use Foreman host groups to control the roles in our instances
- Written Puppet code so that configuration can be written in Hiera (YAML) as straight forward as possible
- Use inheritance as much as possibe ➜ Pool share 90% of configs

```
dcache::dcache_conf:
  dcache:
    dcache.enable.kafka : true
    dcache.kafka.bootstrap-servers: 'it-kafka-broker04.desy.de:9092,it-kafka-broker05.desy.de:9092
    dcache.log.kafka.topic : 'alarm-cms'
    dcache.kafka.topic: 'billing-cms'
    chimera.db.host : 'dcache-dir-cms.desy.de,dcache-core-cms.desy.de'
    dcache.java.memory.direct : '512m'
    dcache.java.memory.heap : '2048m'
```



```
dcache::nfs_exports:
  '/' :
    'dcache-dir-desy'            : '(rw,no_root_squash,no_dcap,no_dcap)'
    'dcache-core-desy'           : '(rw,no_root_squash,no_dcap,no_dcap)'
    'dcache-se-desy'             : '(rw,no_root_squash,no_dcap,no_dcap)'
    'dcache-pack-desy03.desy.de' : '(rw,no_root_squash,no_dcap,no_dcap)'
  '/pnfs/desy.de/belle/belle1' :
    'nafhh-belle0*.desy.de' : '(ro,no_dcap,acl)'
    'htc-belle01.desy.de'   : '(ro,no_dcap,acl)'
    'naf-belle*.desy.de'    : '(ro,no_dcap,acl)'
    'grid-vm05'             : '(ro,no_dcap,acl)'
  '/pnfs/desy.de/belle/belle2':
    'nafhh-belle0*.desy.de'  : '(ro,no_dcap,acl)'
    'htc-belle01.desy.de'    : '(ro,no_dcap,acl)'
    'naf-belle*.desy.de'     : '(ro,no_dcap,acl)'
```

- Idea e.g. simply adding a new export without further knowledge of Puppet or the host setup (reduce need for expert knowledge)
- Hope of reduced load on DOT team did not materialise

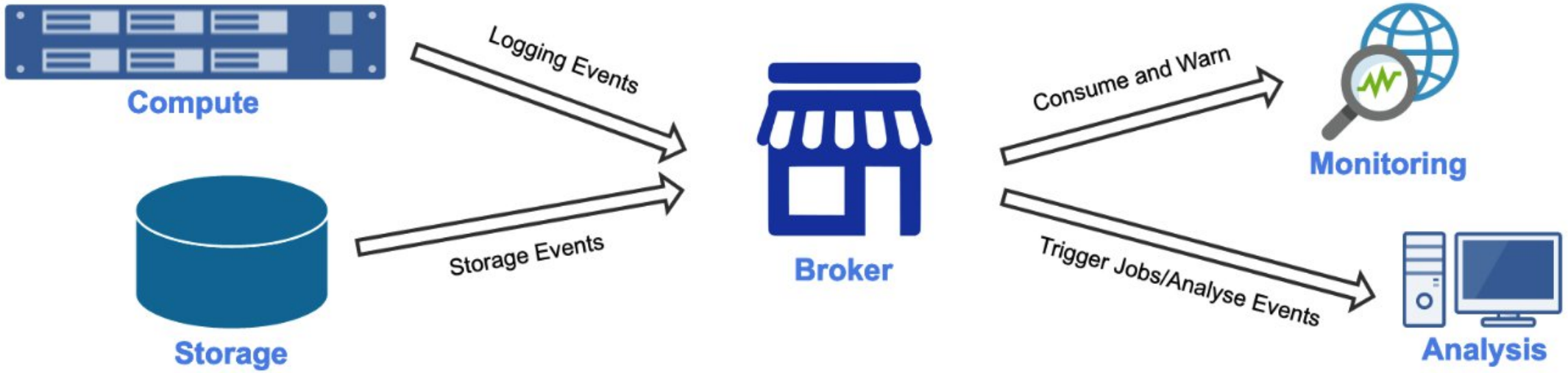
Monitoring for dCache

Using Events to Control Flow of Monitoring Data

- dCache produce large amounts of data about its operation, e.g. billing data
- Build a message stream for dCache billing and logging data based on Apache Kafka → Monitoring becomes a consumer of these messages

kafka

- **Message Producer – Broker – Consumer Model**



From dCache to Analysis

Connecting dCache with Monitoring Infrastructure

- Apache Kafka common IT tool → a lot of services can subscribe and consume Kafka messages

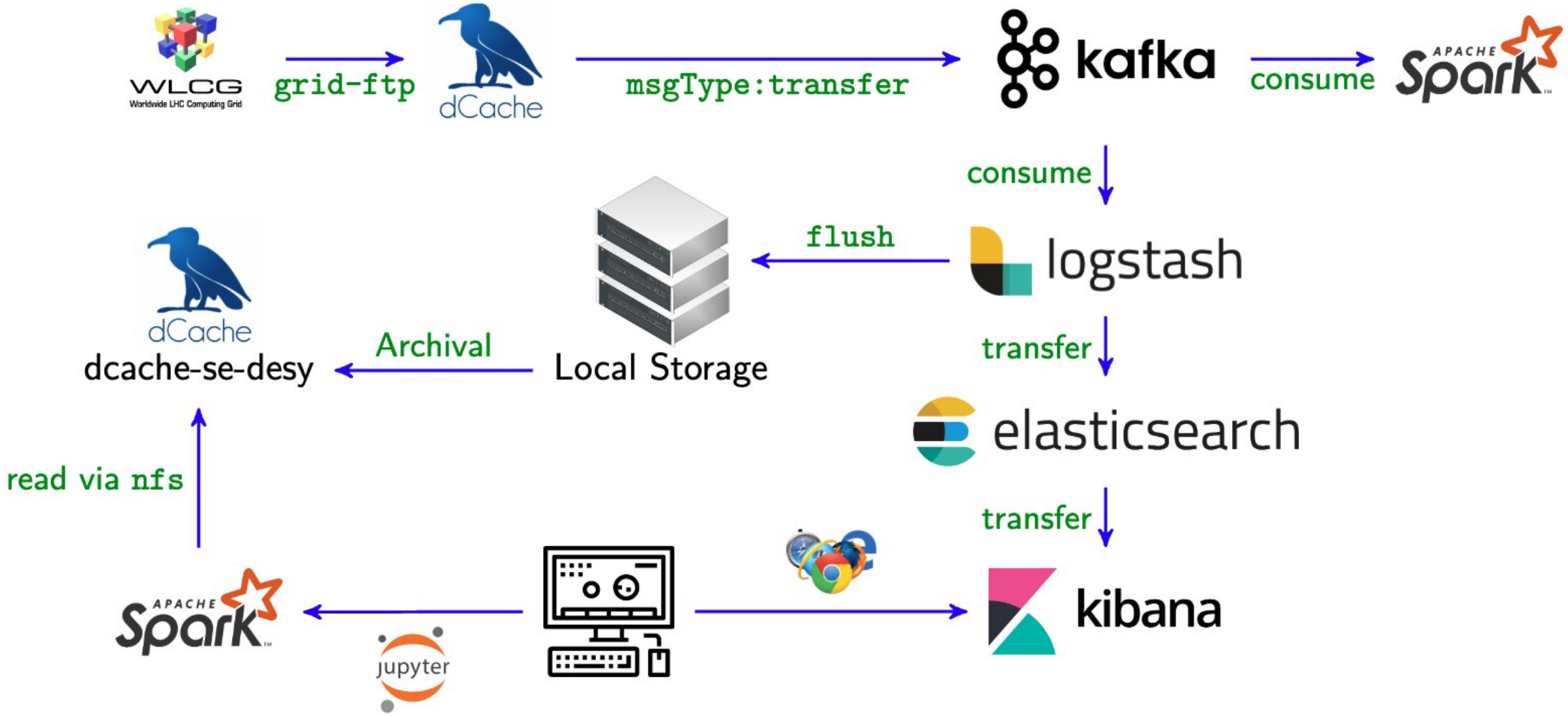
- Beats can easily push JournalD, Syslog, … into Kafka
- Logstash accepts Kafka streams → direct in ingest and visualisation with Elastic tool box (classic monitoring)

Benefit of Plugging Kafka in Between?

- Have operations triggered on Kafka messages → Listen passively rather than search actively
- Often have well known signatures in billing/logging → introduce an automatic response (alarm or fix)
- Ubiquity of Kafka gives rise to easy-to-use libraries in C++, Java or Python
- Have a number of lightweight python based consumers in operation listening for specific patterns
- Complex data analysis frameworks can consume and analyse Kafka events at large

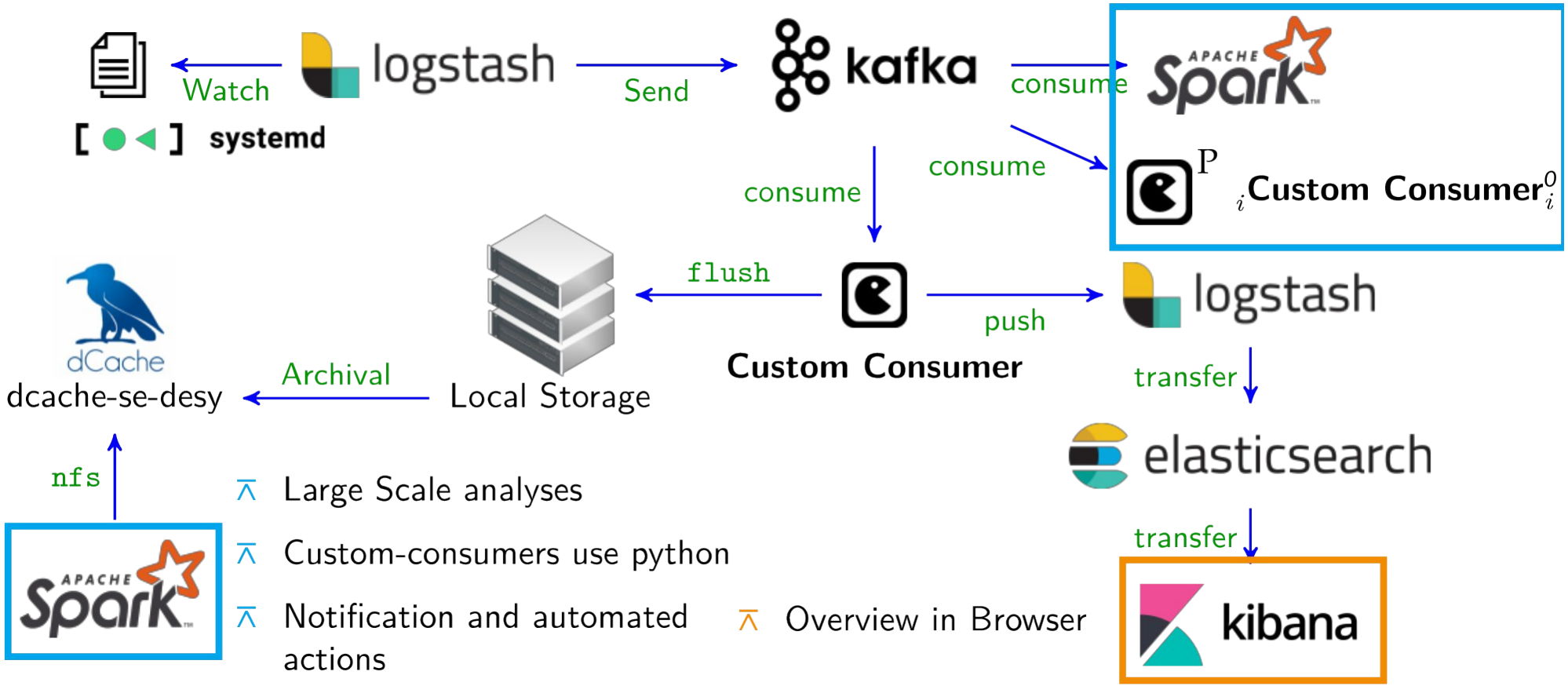
Full Kafka Billing Stream Workflow

Online and Offline Analysis Tools



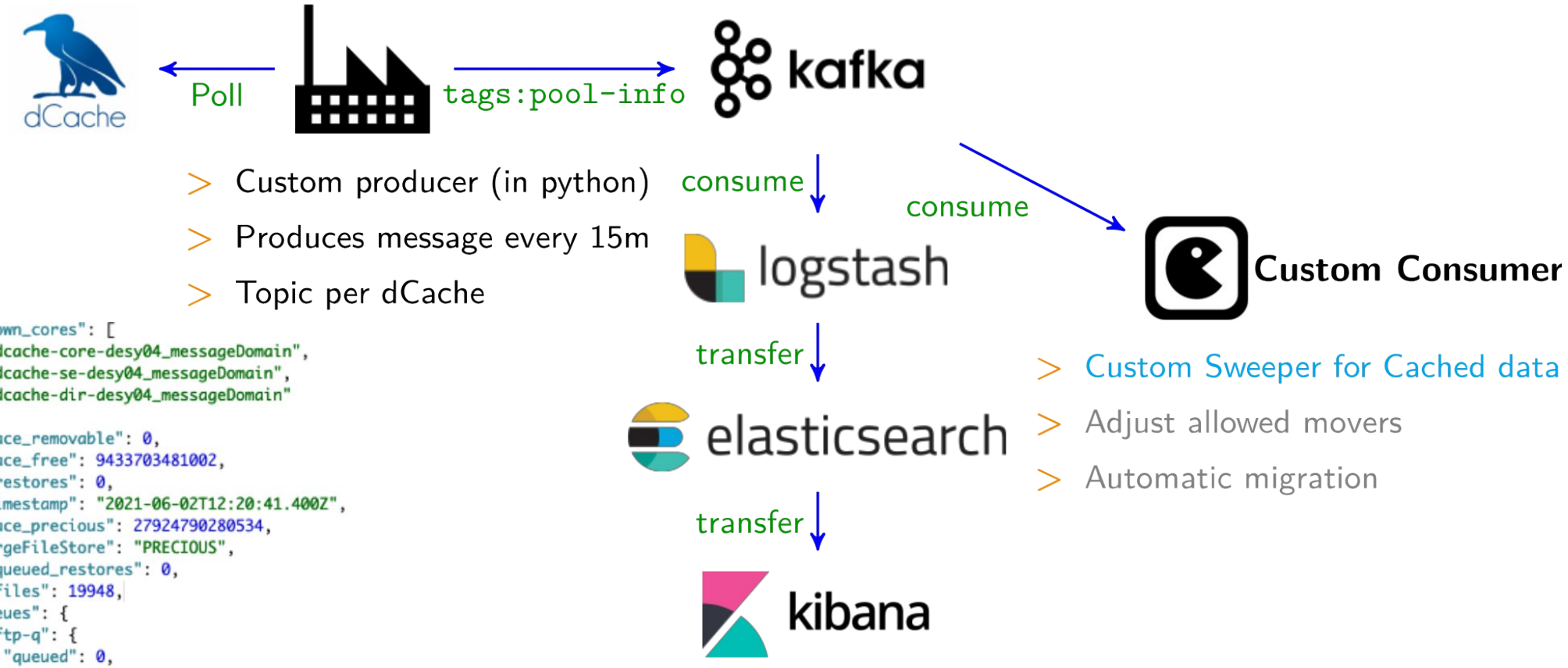
Full dCache Monitoring Workflow

Example for Loggings Stream



Example of Custom Data Stream

Collect Data for Each Pool Every 15min




```
"known_cores": [
  "dcache-core-desy04_messageDomain",
  "dcache-se-desy04_messageDomain",
  "dcache-dir-desy04_messageDomain"
],
"space_removable": 0,
"space_free": 9433703481002,
"n_restores": 0,
"@timestamp": "2021-06-02T12:20:41.400Z",
"space_precious": 27924790280534,
"LargeFileStore": "PRECIOUS",
"n_queued_restores": 0,
"n_files": 19948,
"queues": {
  "ftp-q": {
    "queued": 0,
    "active": 0
  },
```


- Pool plot shown earlier based on this stream
- Also archived on dCache → allows forensics on which pool belonged where in the past

Custom Kafka Consumers

An Example for dCache Kafka Data

- Apache Kafka Stream-API allows an easy development e.g. in Python
- Kafka JSON structures makes handling with data structures convenient and easy
- Python makes also further actions easy to develop down to employing ML/AI methods

Sync&Share File Registration

Execute for every event

Filter selection

Trigger the scan

```
class ScanOnArrival(DOTKafkaMessageAnalyser):
    def __init__(self, instance = '', listenPath = '/', verbose = False) :
        self.instance = instance
        self.listenPath = listenPath
        self.verbose = verbose

    def execute_shell(self, command=''):
        systemcall = subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True)
        log, logerr = systemcall.communicate()
        return log

    def execute(self, message):
        message_data = message.value

        if message_data.get('msgType', None) == 'request' and \
           message_data.get('client',None) != '127.0.0.1' and \
           message_data.get('moverInfo',{}).get('isP2p', None) == False and \
           message_data.get('moverInfo', {}).get('isWrite', None) == 'write' and \
           message_data.get('moverInfo', {}).get('status',{}).get('code', None) == 0 and \
           self.listenPath in  message_data.get('transferPath',None):
            print("File: {} - written from {}; requires file-scan in NextCloud)".format(message_data.get('transferPath',None),
                                                                                        message_data.get('client', None)
                                                                                        ))
            file_scan_cmd = 'sudo -u apache php /var/www/nextcloud/occ files:scan --path "{}"'.format(scan_path)
            scan_log = self.execute_shell(command=file_scan_cmd)

            if self.verbose :
                print(scan_log)
```

Example for Custom Consumers

Alarming if Pools Become Disabled

- Apache Kafka Logging-Stream listens to all logging streams
- Check each message if the message contains e.g. ‘pool mode changed to disabled’
- Generate an email to admins and enter entry to database
- Notify if pool recovers and remove entry in database

Time: 07:51:05 (+0000) - 2024.06.06, Stale entries: 1779, Host: christif@naf-it01.desy.de

Pool Domain	Pool Group	Error Status	Error Type	Error Message	Date of latest Occurrence
dcache-xfel497-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:49:55.404000+00:00
dcache-xfel498-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:50:06.872000+00:00
dcache-xfel500-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:50:30.024000+00:00
dcache-xfel499-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:50:18.443000+00:00

At most 300 entries are shown. Refer to the database to get all entries.

Legend:

unresolved	The issue persists
resolved	The issue has been resolved
stale	The issue has persisted for multiple days. The entry must be manually reviewed and pruned
finalized	The entry will be deleted momentarily
temporary	The issue has no associated resolved-messageand will be treated as self-resolving
oddity	The issue is treated as self-resolving and doesn't warrant an email notification on its own
permanent	The issue will not be deleted and must be removed manually.
no-delay	This issue will always produce a notification

Developed by Felix Christians

The Larger Picture – Connecting to Other dCache Consumers

Context to Other Active Consumers of dCache Storage Events