ARC with caching

NeIC NT1 Manager
Mattias Wadenstein
<maswan@ndgf.org>

2019-10-11
ARC workshop, Amsterdam

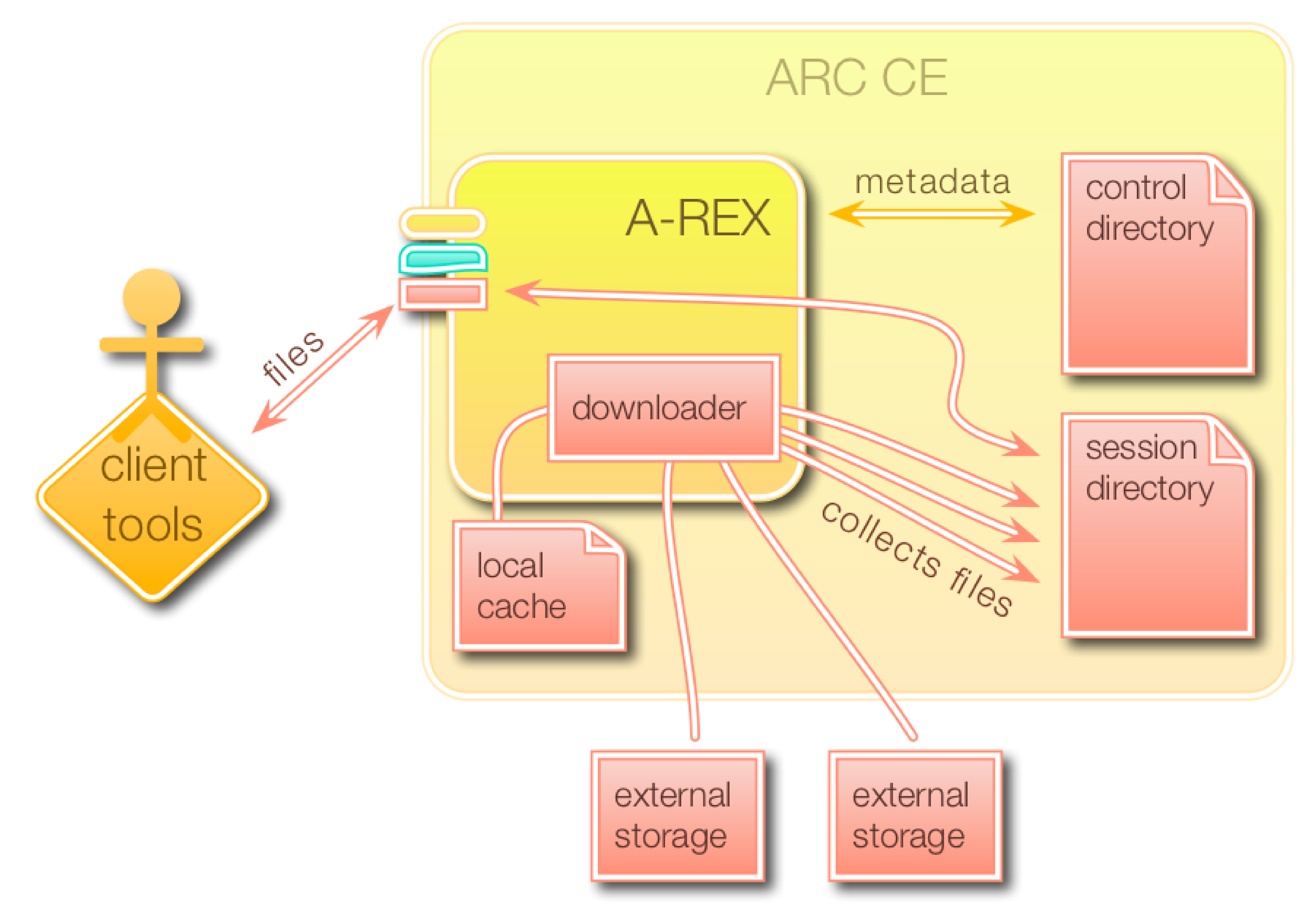
NORDIC E-INFRASTRUCTURE COLLABORATION

Overview

- What
- Why
- Design
- In practice

What

- ARC can do data staging
  - Prepares all input files needed by the job before submission to batch system
  - Saves all requested outputs to remote storage afterwards
  - Cache for reuse of input files between jobs



What

- ARC in data caching mode
  - Each job description has a list of input and output files (rucio://...)
  - The CE stages all these files to local cache and links them in the session directory
  - The job is submitted to batch system and runs on local files only
  - Afterwards the listed output files are uploaded to SEs
- Caches are normal shared filesystems
  - NFS, CephFS, GPFS, Lustre, etc

Why

- Overall efficiency
  - Data access is on low-latency local filesystems
  - Download before submission to batch system → better CPU efficiency
- Non-local storage
  - Like NDGF with distributed storage
  - Or a “compute only” site
- Limited external connectivity
  - Like HPC sites where external connectivity might be blocked or only available through a slow NAT
- No need for grid-aware computational software

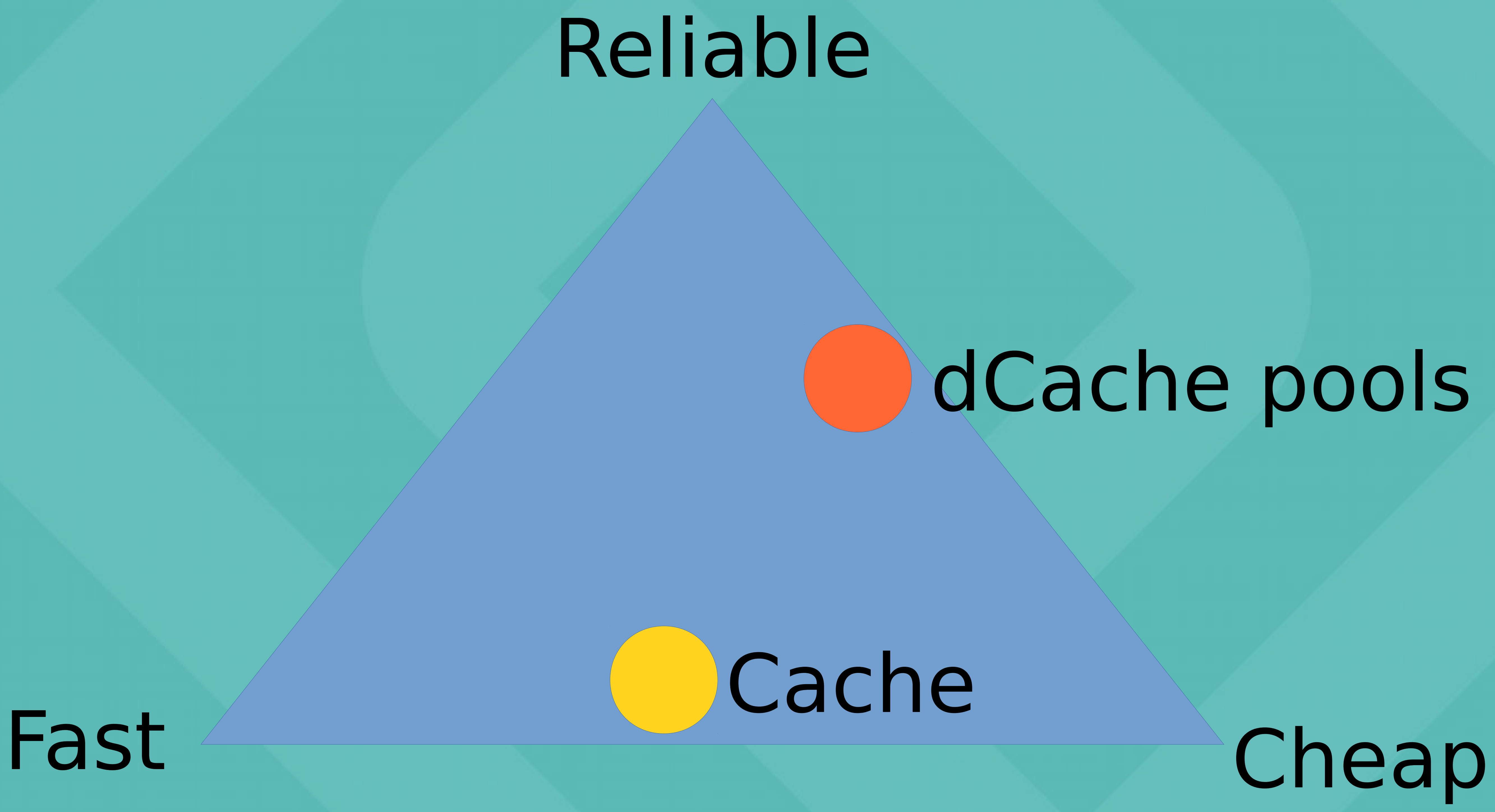
Design

- Caches and session directories are placed on shared filesystems between CE and WNs
- DataDeliveryService nodes transfer data in and out from the session directory and caches
  - Can be one or several, depending on the data rates you want to support
  - One common deployment is to have 5-15 NFS servers all running a DDS for the local filesystem
- Caches are automatically cleaned LRU

Design

- Lots of protocol support
  - HTTPS, GridFTP, SRM, S3, rucio, XrootD, ACIX, etc
- DataDeliveryService processes only do simple transfers
  - Scheduling logic etc in the central A-REX component
- Remote access to cache contents possible
  - Sharing cache between close CEs instead of download from far SE
  - Two different methods of publishing contents
  - Can also be used for cache-aware scheduling
- Dynamic on demand downloads
  - Possible through “candypond” API usage from running job

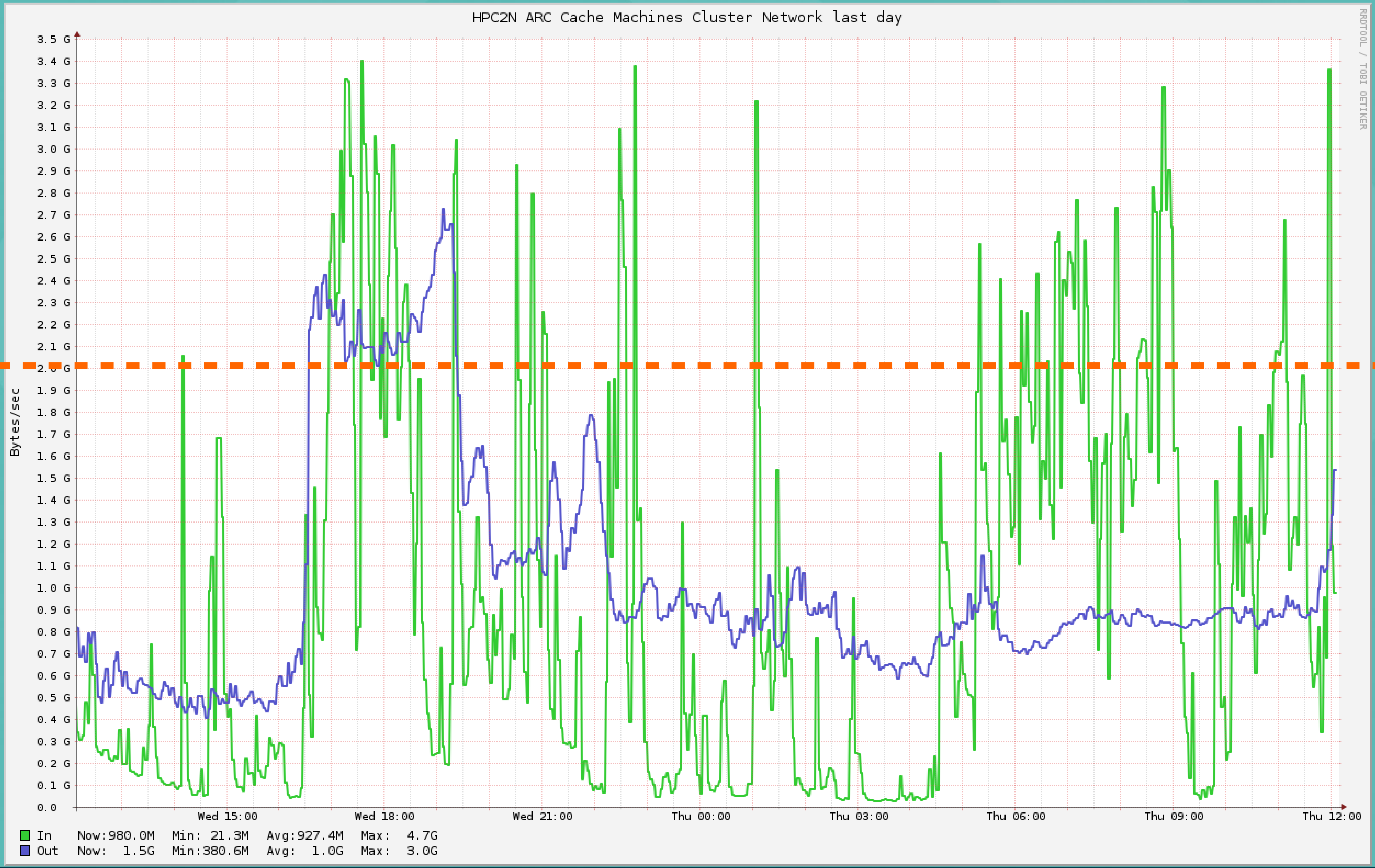
Design



In practice

- This is experience from NDGF ATLAS usage
  - Other communities might have different IO patterns
- About 100TiB is sufficient cache space to support a few thousand cores of ATLAS compute
  - Bigger will have better cache reuse
  - Sample point, a 204TiB cache for ~4k ATLAS cores:
    50% of files accessed within 24h
    90% of files accessed within 48h

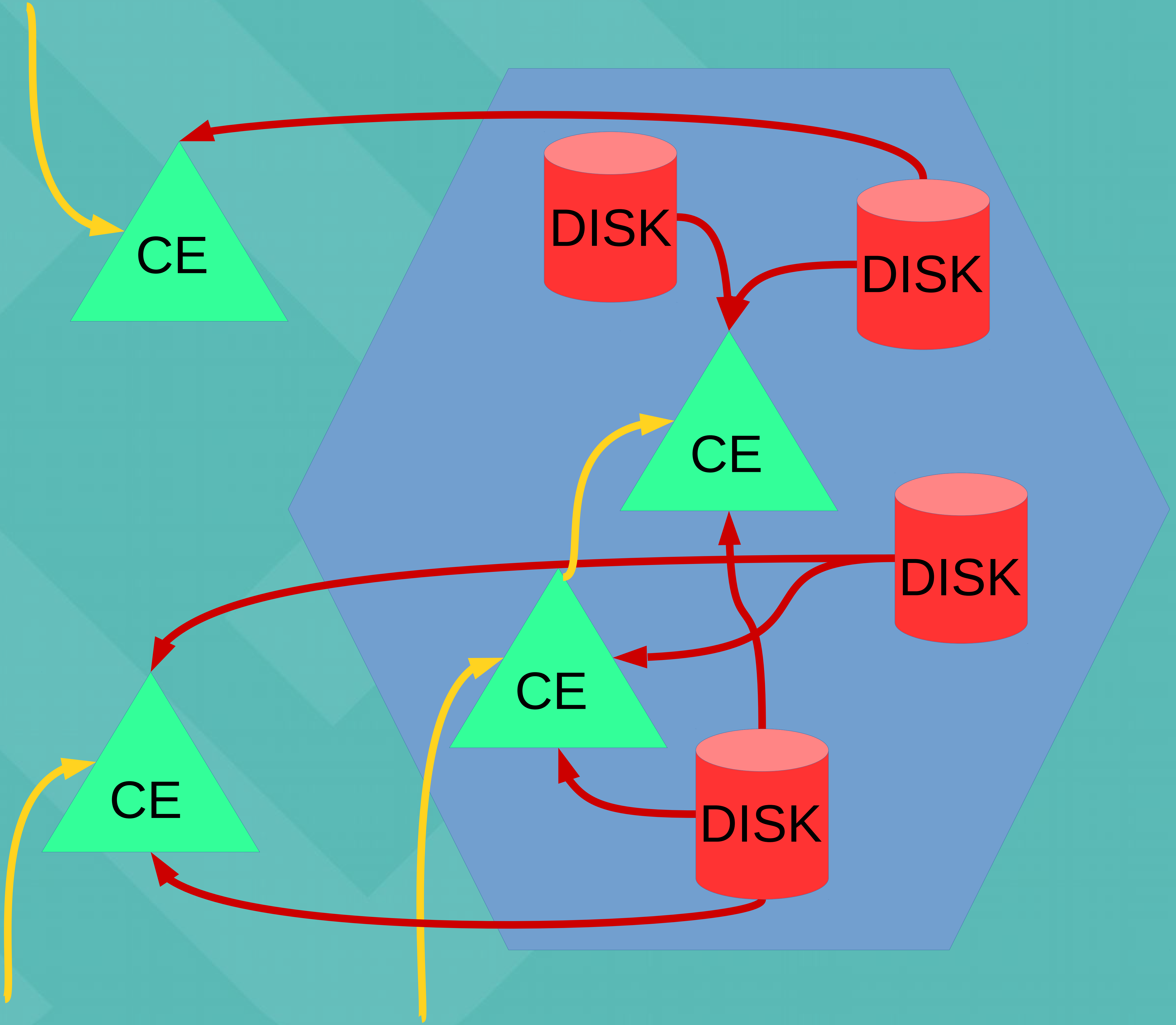
In Practice (4k cores ATLAS)



2 Gbyte/s

ARC in a distributed site context

- Staging makes ARC location agnostic
- No problem getting some data from other sites
- CE-CE transfer from one cache to another



Questions?

NORDIC E-INFRASTRUCTURE COLLABORATION