Transfer2go R&D

V. Kuznetsov, Cornell Univ., R. Shah GSoC'17 student, D. Bonacorsi Univ. of Bologna

November 2017, CMS C&O week

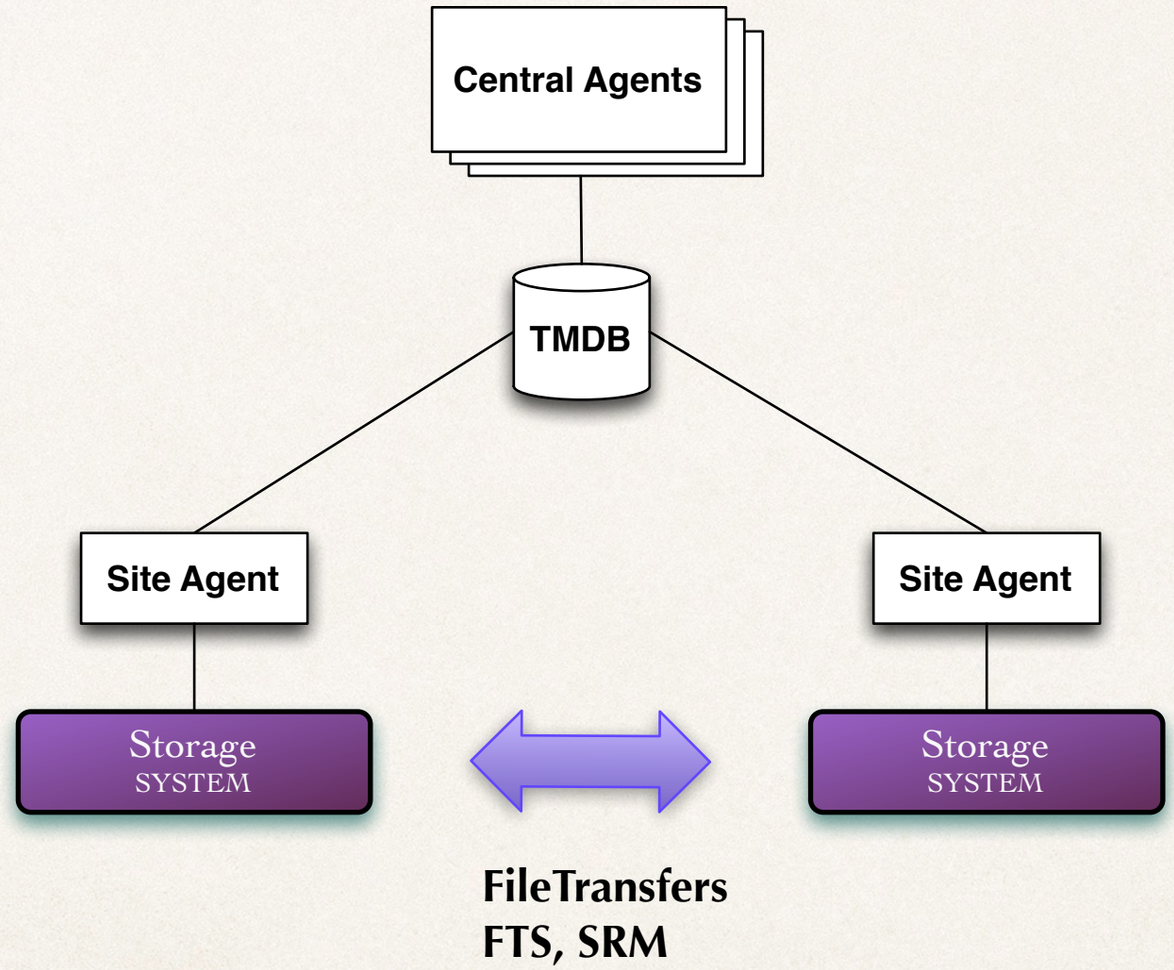
Transfer2go project

- **transfer2go** project is an R&D for future data replication and bookkeeping system with aim to
  - study de-centralize approach (remove ORACLE back-end)
  - support transfer of user-based data
  - support multiple OSes: Linux (amd64, power8, arm64) , Mac, Windows
  - study data streaming, e.g. individual events (part of go-hep rootio)
  - push/pull model with asynchronous agents and de-centralized bookkeeping system
  - intelligent routing based on ML predictions of historical data transfers
  - implement ideas in modern concurrent Go-programing language
- ideas can be used/tested towards understanding our needs for next generation of data location/replication and bookkeeping system

Why Go?

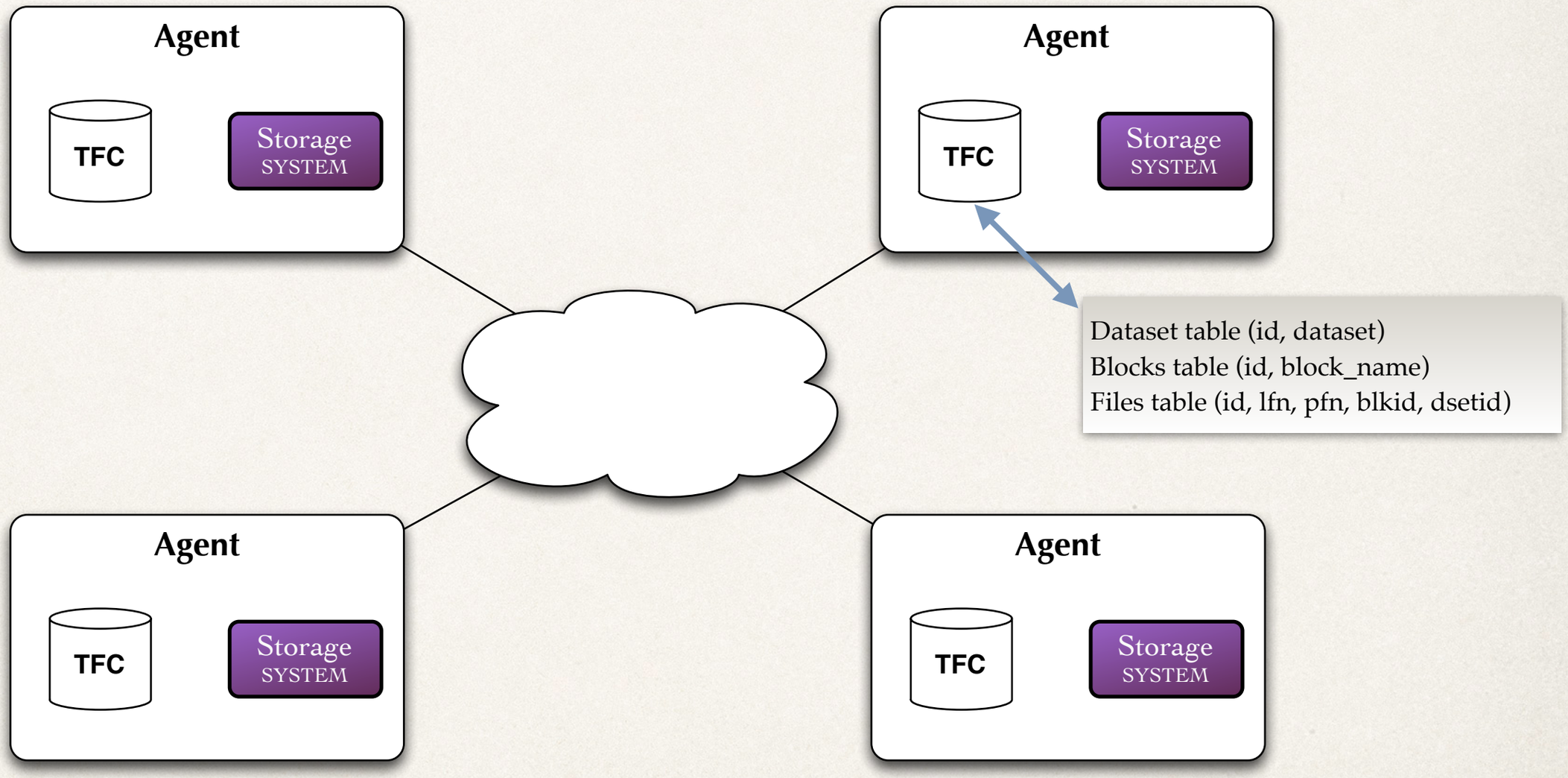
- Builtin concurrency into core of the language
  - ready for multi-core world
- Ready for web development
  - web components (engine, templates, security, etc.) are part of Standard Library, i.e. no 3rd party software is required to build services
- Simplicity, Speed and Portability
  - writing code as simple as Python (GC included), run it as fast as C
  - static executable allows you to forget about deployment headache

Simplified PhEDEx architecture



System based on loosely coupled Site Agent daemons with central DB

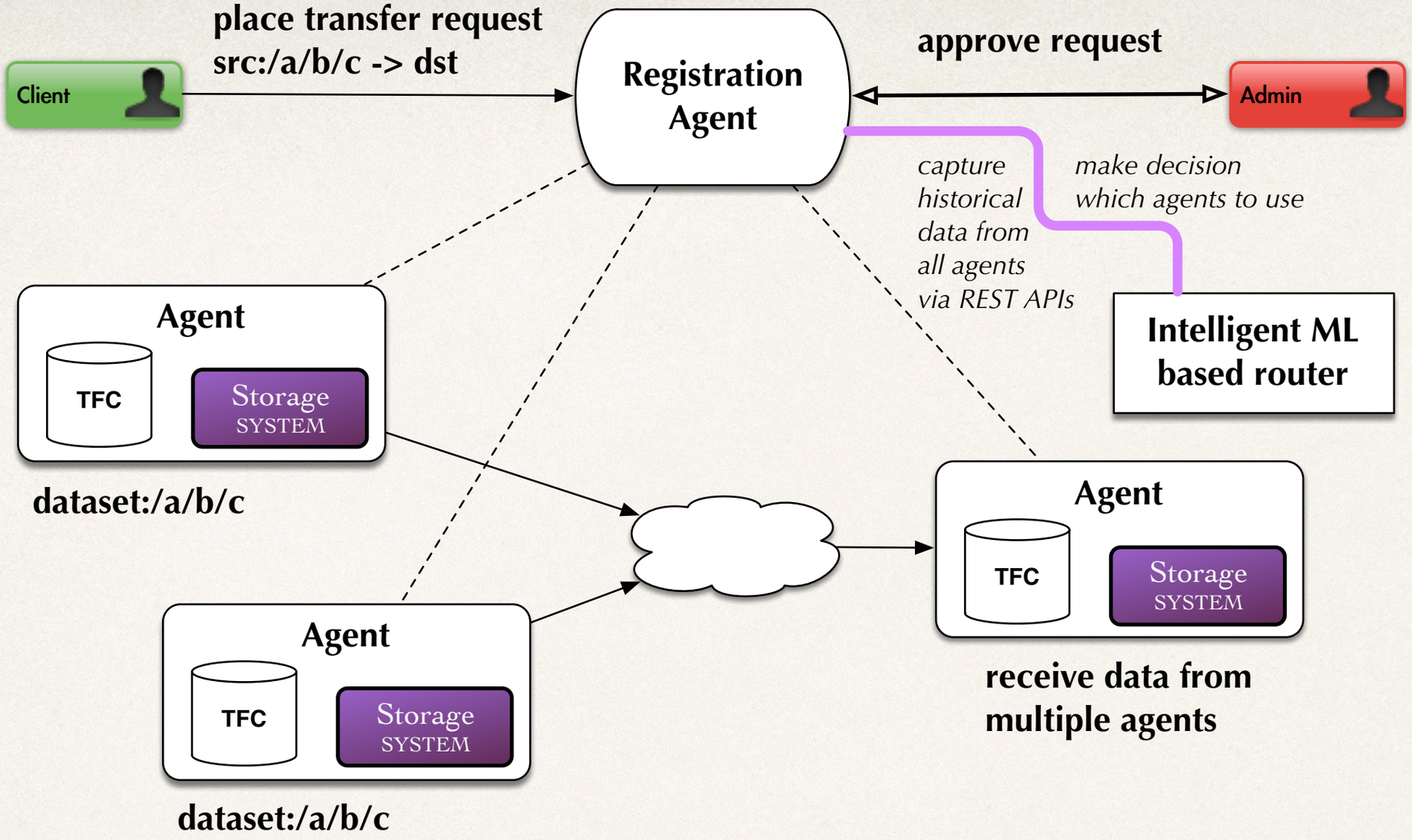
De-centralize architecture



De-centralize agents data-services with **auto-discovery**
and **intelligent routing**, can use either PUSH or PULL model for data transfers

Data transfer models

- In de-centralized system agents may have different roles:
  - registration agent responsible for handling user requests
    - find out where data resides; obtain status of data agents; provide board for request approval; initiate data transfers; train and use intelligent routing
  - source and destination agents handle the transfer requests either via PUSH or PULL model
- PUSH model
  - source agent initiates the transfer to destination one (HTTP POST request to destination agent)
- PULL model
  - destination agent initiates the transfer upon request from registration agent (HTTP GET request to source agent)

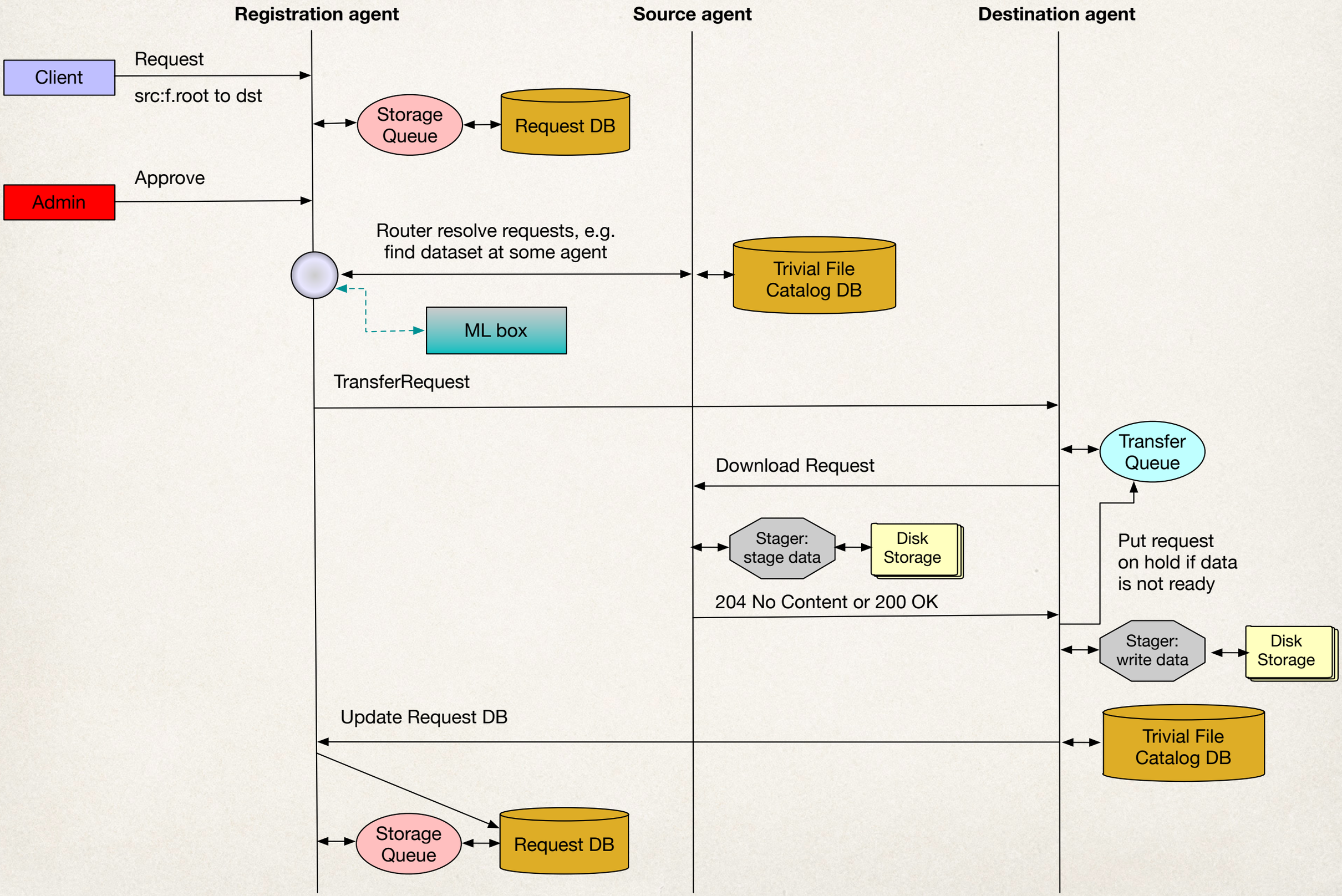


There are 3 types of agents: **registration agent** register client's request and delegate work to other agents; **source agent** holds the data to be transferred; **destination agent** fetch the data from the source one. These roles can be interchanged among de-centralized system (anyone can become in one or few roles).

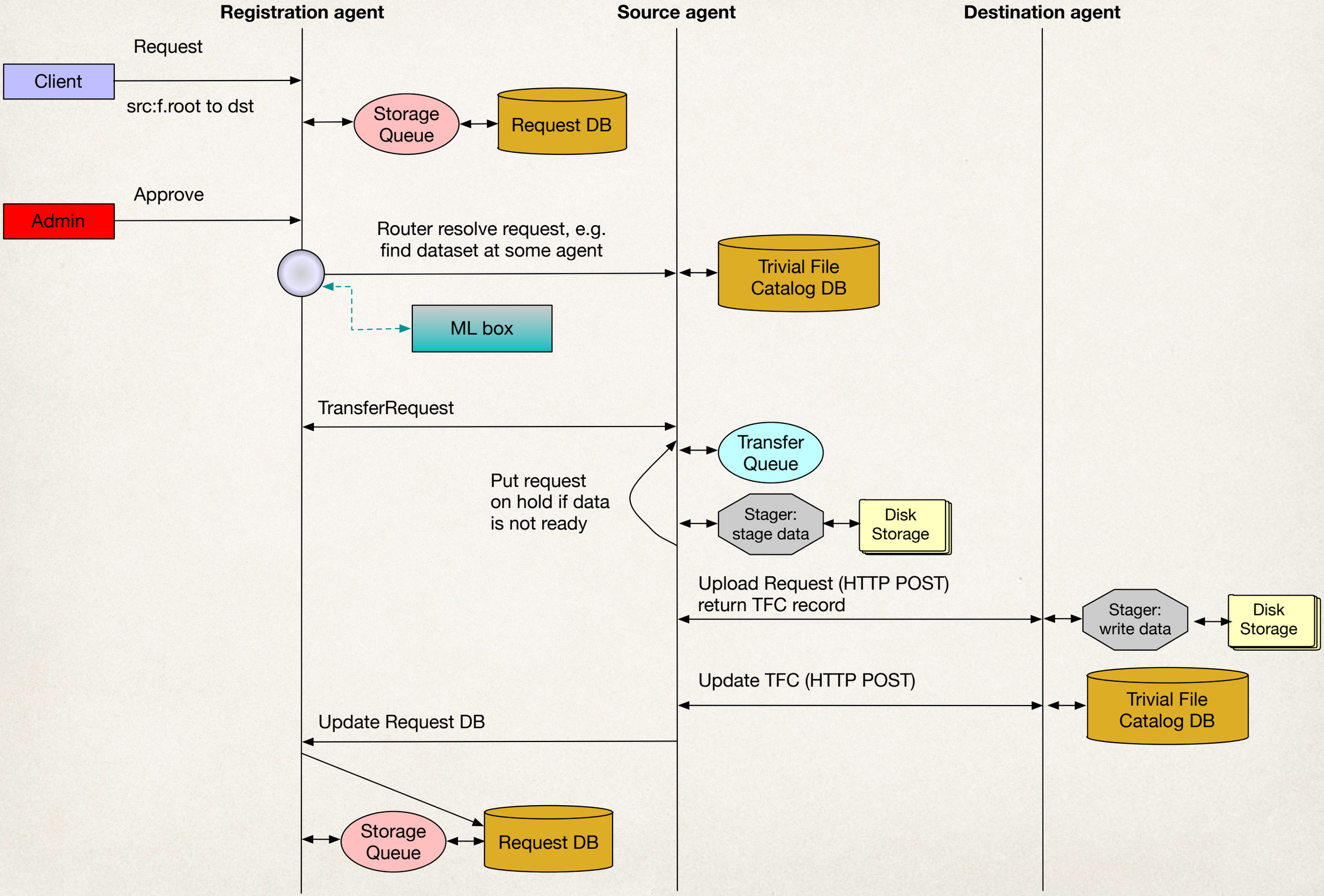
A client sends data transfer request to registration agent

After request approval the registration agent will fetch the catalog entry of the request from registered agents along with their status. This can be used by intelligent router to find optimal route.

Request data flow diagram in pull model



Request data flow diagram in push model



Central catalog

- In de-centralize system there is no central catalog (PhEDEx ORACLE DB)
- Each agent keeps local transfer catalog which contains information about this specific agent
- Agent's catalogs can be streamed to central location on periodic basis, e.g. to HDFS, which can be treated as a central catalog and backup system
  - any snapshot can be taken based on timestamp which will represent "central" catalog
  - if specific agent loose its catalog it can be restored from central system
  - queries can be done against "central" catalog (via Spark job) or send to individual agents (à la DNS look-up)
  - this design choice allows to perform analytics studies of catalog(s) without affecting system behavior, e.g. Hadoop+Spark+MONIT+Next-generation of BigData tools

Transfer2go current status

- Go-implementation (native concurrency support): github.com/vkuznet/transfer2go
- Agents are data-services based on secure server with SiteDB authentication (no need for cmsweb)
- PUSH and PULL model implementations are in place
- Intelligent router trained as linear regression model using agent's metrics (CPU, RAM, throughput)
- Auto-discovery among agents, agents can play any role in a system (registration, source or destination agent's role)
- Agent keeps track of transfers and metrics in local DB
- Default transfer protocol is HTTP or transfer can be delegated to FTS
- Single static executable for client and agents
- Zero deployment overhead, i.e. easy to deploy at any node regardless of OS or software stack

Tests (push & pull models)

- We performed the following integration tests between CERN (VMs) and UI-enabled resources at Bologna Tier-3 colocated with CNAF (tests used SSH tunnels via lxplus to avoid port openings)
  - **Transfer Monitoring test:** check file arrival, monitor node usage
  - **Dataset/block test:** transfer files from a given dataset/block
  - **Incomplete dataset test:** if one|few files missing on a site transfer the rest
  - **Multi-source test:** dataset is spread across sites, transfer will initiate transfer of all files from different sites
  - **Site failure test:** dataset is spread across sites, imitate failure of one site, transfer should compete remaining files from another site
  - **Measure transfer throughput** on a single node: start as many concurrent transfer as possible until site will be fully saturated

Test results

- Pull model http://bit.ly/2zBcM2e and Push model http://bit.ly/2lXprGQ
- System is capable of finding data across de-centralized distributed agents
- Successfully transfer individual files, blocks or full datasets
- Transfer completion in case of missing data and/or failed sites
- Low memory impact at source and destination nodes
- Low-to-medium impact on CPU utilization at source nodes
- Medium-to-high impact on CPU utilization at destination nodes
- Transfer time scales linear with file size (1GB in 1sec, 15GB in 14sec)
- Parallel file transfer on a single node is natively supported

Summary

- The currently developed prototype matches the specification and tests show it works as expected
  - unfinished parts: auth policies, priorities, usage of external middleware (e.g. FTS)
- Go-language proves to be an excellent choice for developing web-based de-centralized asynchronous concurrent system due to its native support of those components in core language
  - built-in concurrency, web components (engine and templates) and usage of multi-cores is part of the language and its standard library (no 3rd party tools are required)
  - zero deployment overhead, both agents and clients can be easily upgrade, drop a static executable in place and you're done (no tedious upgrade cycles)
  - static executable is available for modern architectures: Linux (amd64, power8, arm64), OSX, Windows
- De-centralized architecture works and does not require ORACLE DB
  - local DB can be stored on HDFS and we may use Spark to perform analytics studies over TFC
- Intelligent routing, transfer of user data is part of the system design