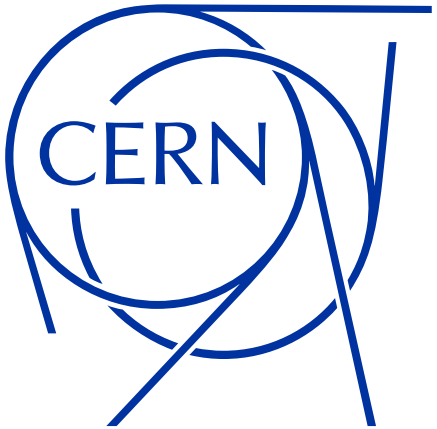


FTS4 - Optimising Performance & Efficiency

Louis RÉGNIER

September 15, 2026

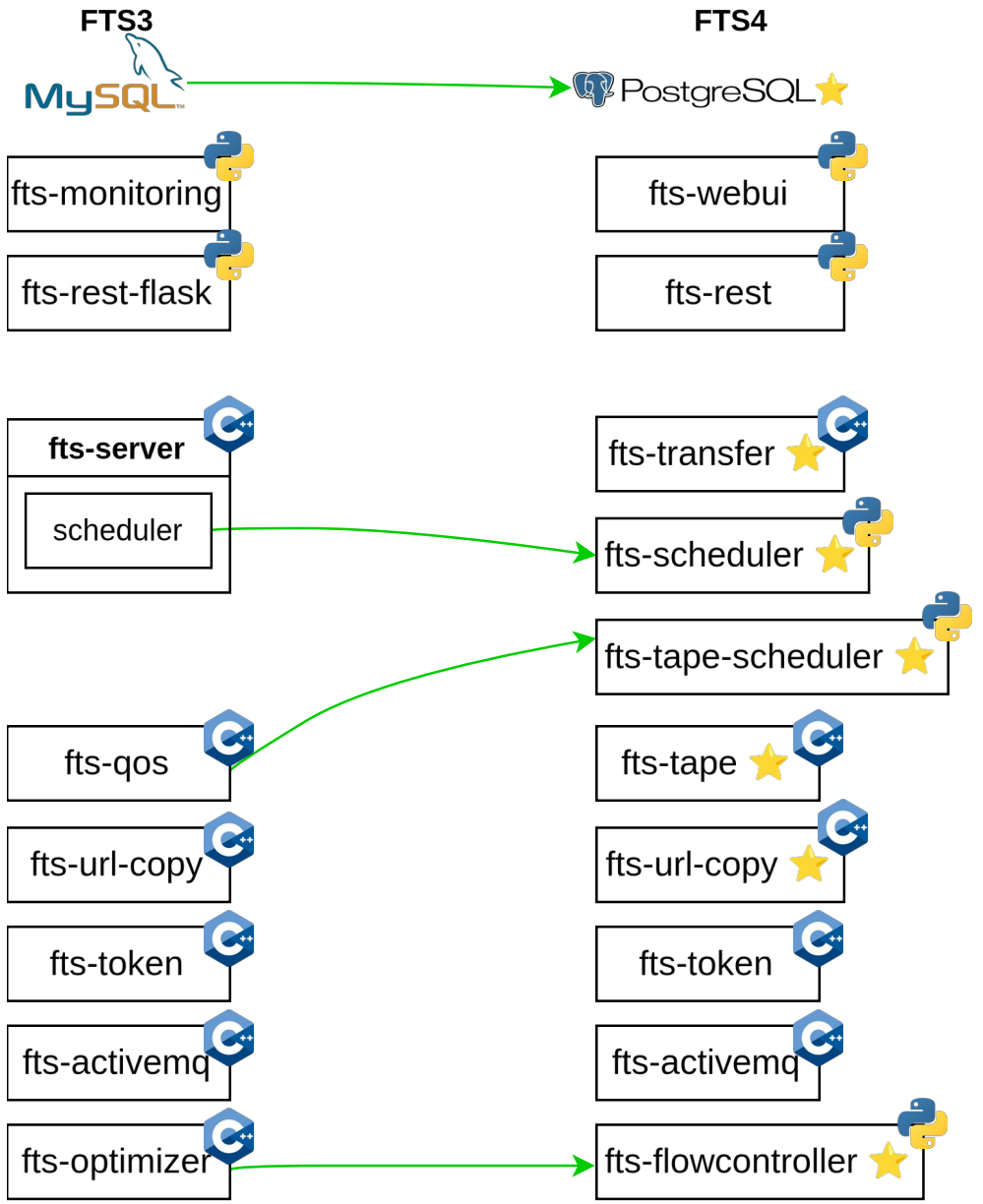
Summary:

- FTS4 architecture
- FTS4 deployment at CERN
- FTS4 transfer pipeline overview
- Optimising FTS to handle 10.000 concurrent transfers per machine

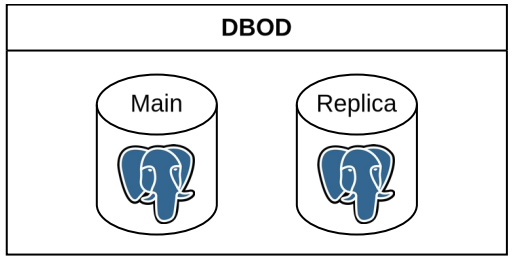
FTS4 – architecture

Main changes from FTS3:

- Moving from MySQL to PostgreSQL
- New schedulers in Python
- New fts-tape daemon replacing fts-qos
- New fts-flowcontroller replacing fts-optimizer



FTS4 – CERN deployment



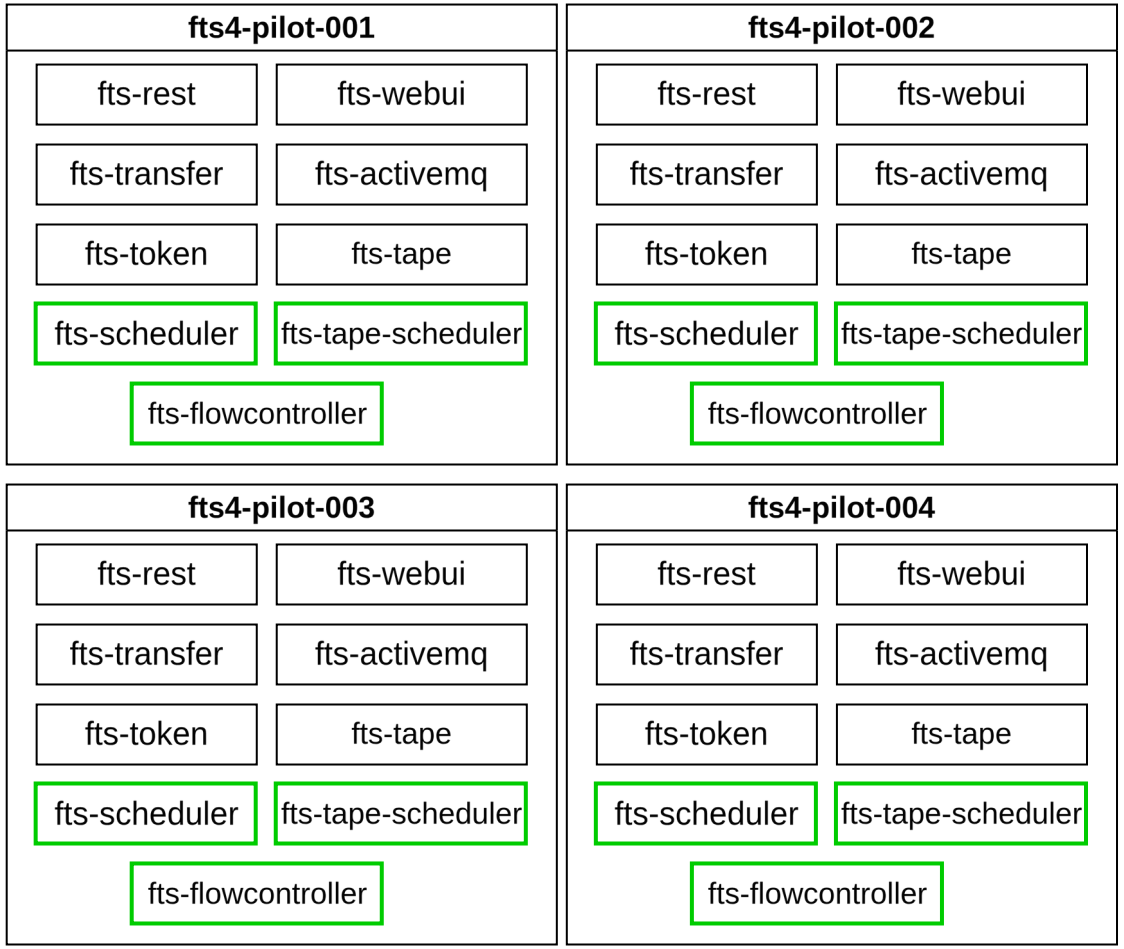
Current deployment of FTS4:

- Moving to physical hardware instead of VMs
- 4 physical machines per cluster
- 1 database and replica per cluster

Pros:

→ Less active sessions on the DB

→ Needs extra care to handle high frequency



One set active at a time per cluster (Highly Available)

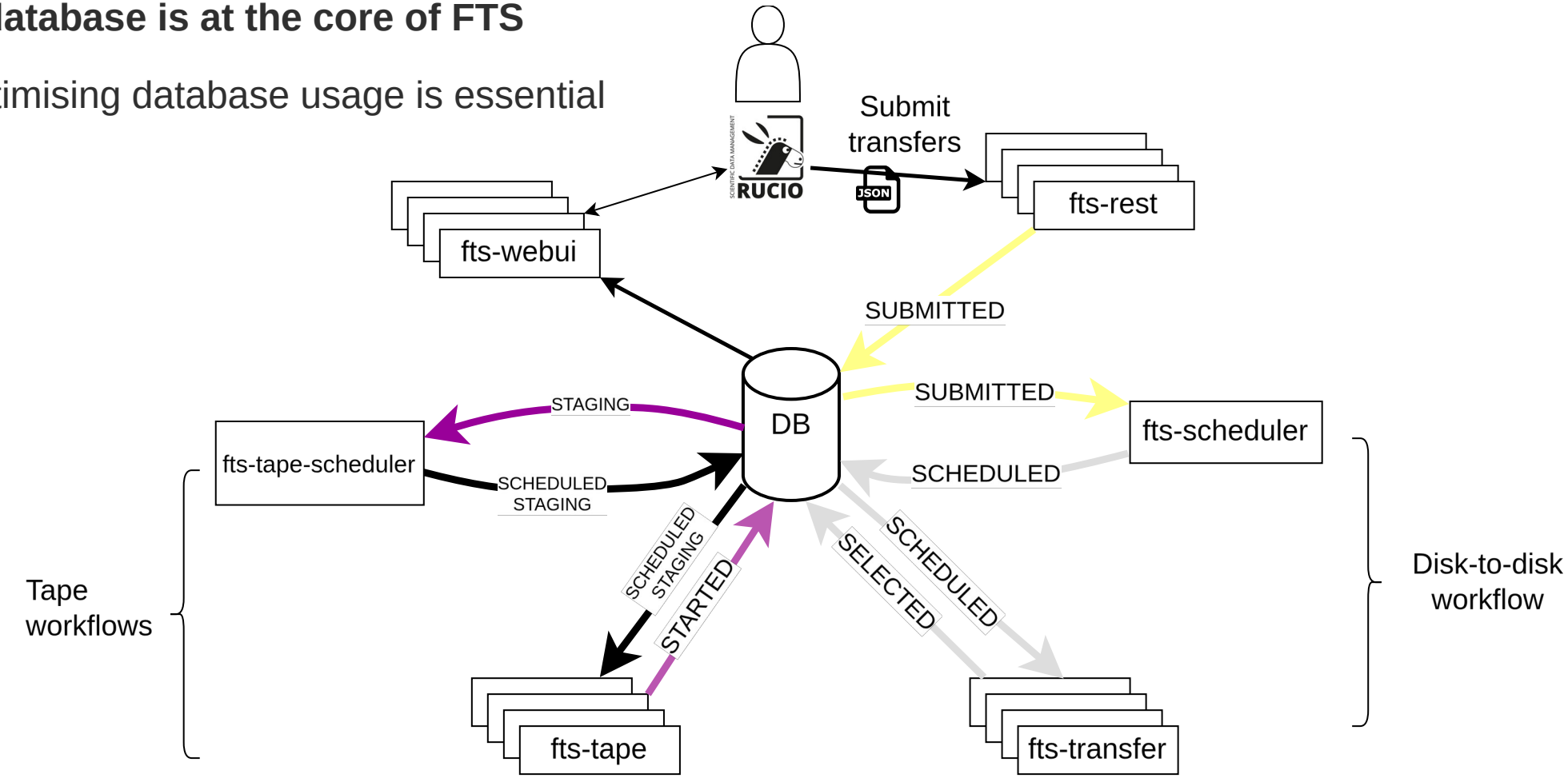
Objectives:

→ Handle 40.000 transfers concurrently

FTS4 – Transfer pipeline

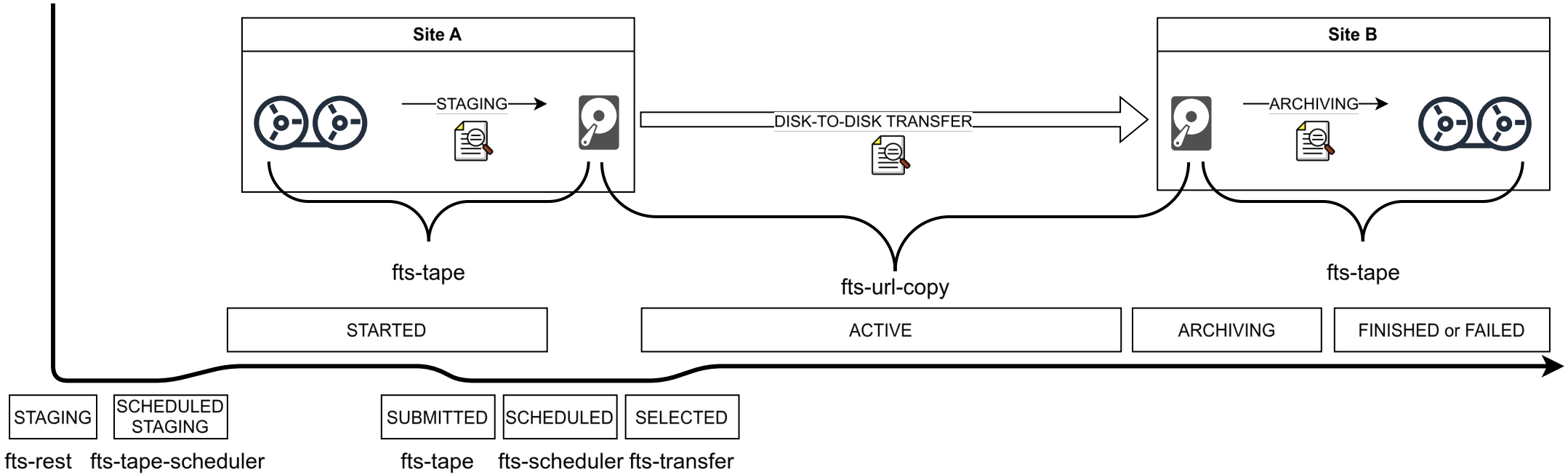
The database is at the core of FTS

→ Optimising database usage is essential

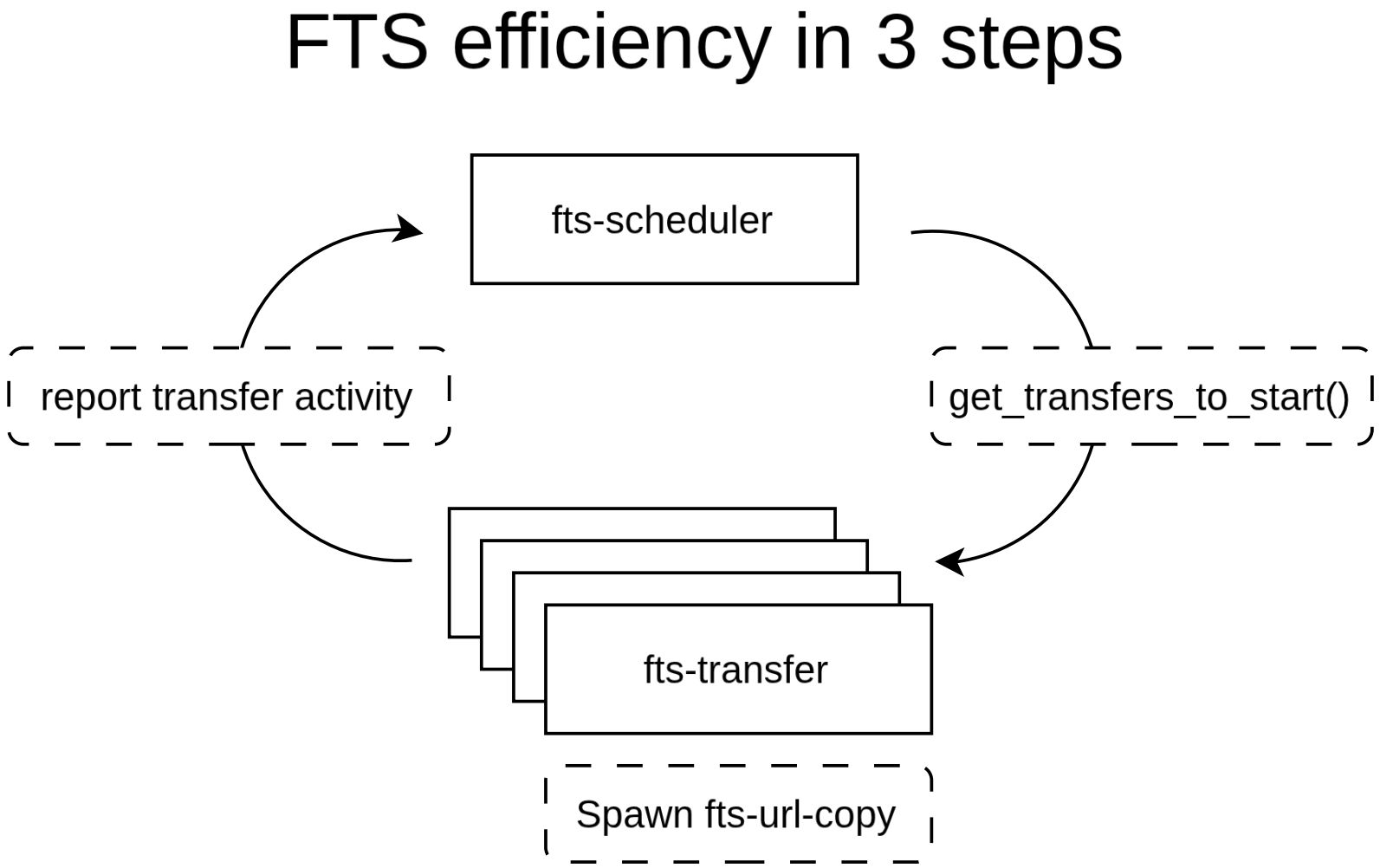


FTS4 – Complex transfer

Example:
$ fts-rest-transfer-submit --bringonline-timeout 86400 --archive-timeout 86400 https://A/xyz.root https://B/xyz.root



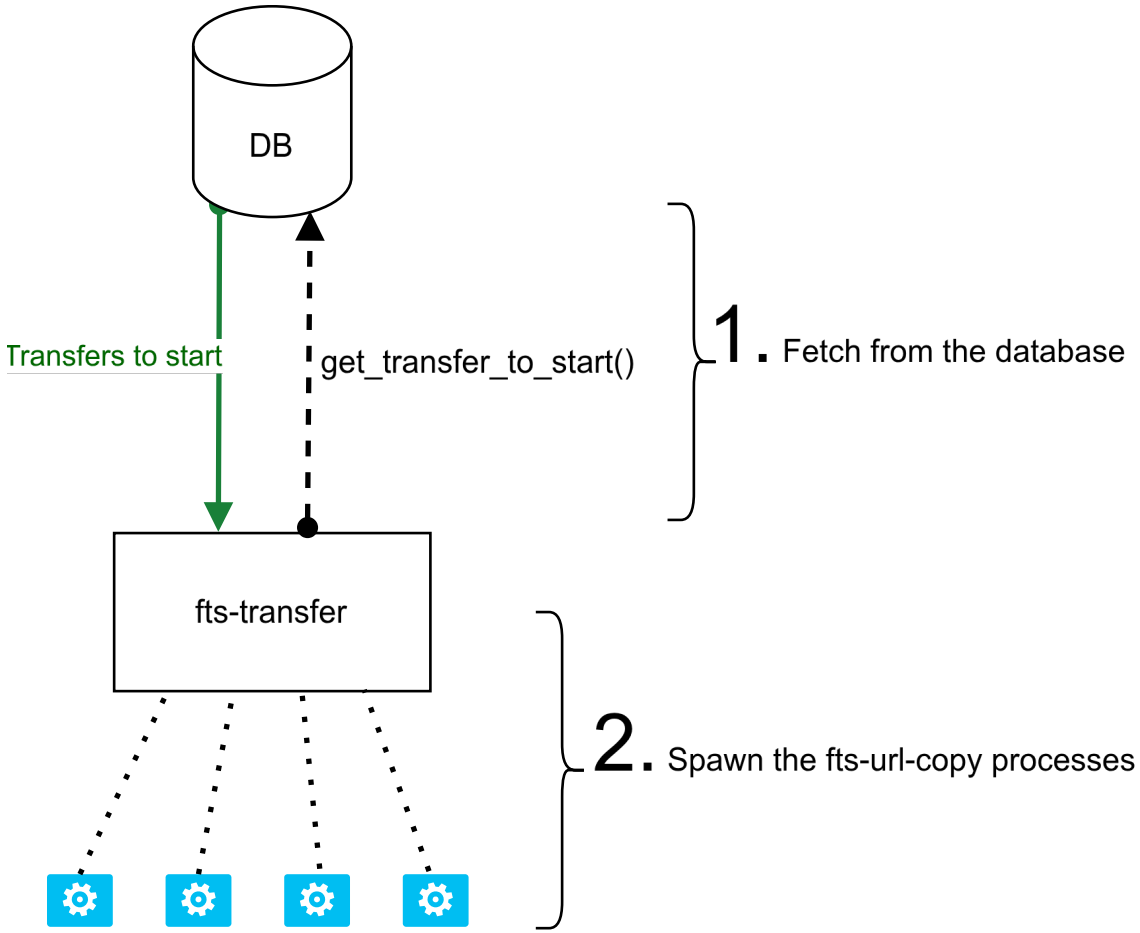
FTS4 – Enabling 10.000 concurrent transfers



FTS4 – Transfer start efficiency

First blocker: Starting the transfer is too slow

1. Getting 10.000 transfers to start
   → Can take up to 100 seconds to fetch from DB
2. Spawning 10.000 fts-url-copy processes
   → Can take up to 40 seconds (250Hz)



FTS4 – Investigating slow database queries

Different ways to identify and analyse slow queries

- Looking at the FTS webui

→ Wondering why the transfer is not “ACTIVE” yet

- Client logging

→ Simple, and shows quickly the scalability in production

- PostgreSQL specifics

→ pg_stat_statement

→ EXPLAIN ANALYZE (probably the most important one)

→ pgBadger

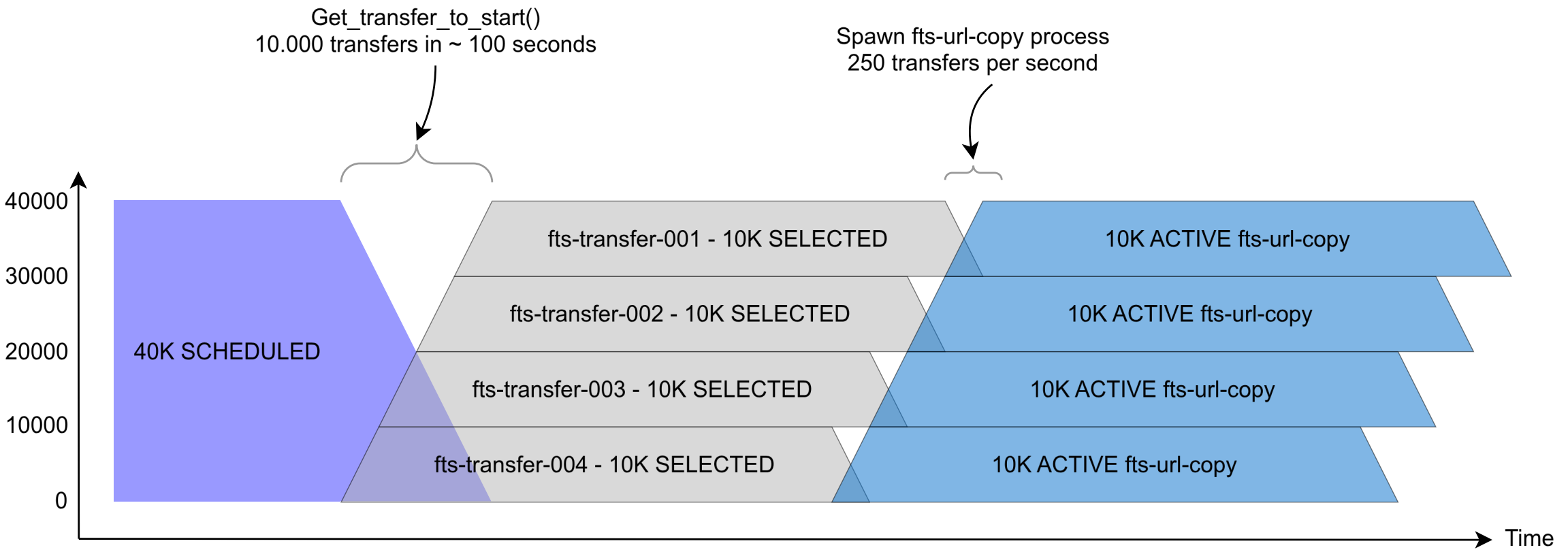
→ pg_activity

FTS4 – get_transfer_to_start()

Query fetching all the informations required to manage a disk-to-disk transfer

- Source, destination
- Credentials to use: oauth2, x509
- User defined constraints: timeout, checksum, overwrite, retry, clean-up…
- Transfer metadata, Job metadata, Archive metadata
- Third Party Copy mode
- ...

FTS4 – get_transfer_to_start()

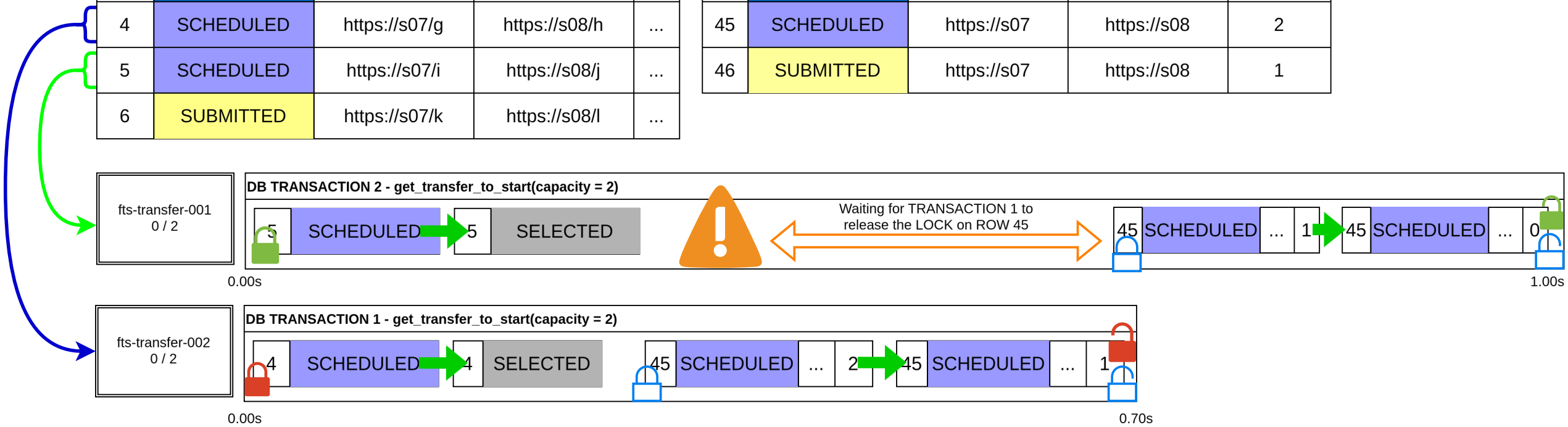


FTS4 – get_transfer_to_start()

The problem:

Transfers (t_file table) - simpflied view of the database				
ID	STATE	SOURCE	DESTINATION	...
1	FINISHED	https://s01/a	https://s02/b	...
2	FAILED	https://s03/c	https://s04/d	...
3	ACTIVE	https://s05/e	https://s06/f	...
4	SCHEDULED	https://s07/g	https://s08/h	...
5	SCHEDULED	https://s07/i	https://s08/j	...
6	SUBMITTED	https://s07/k	https://s08/l	...

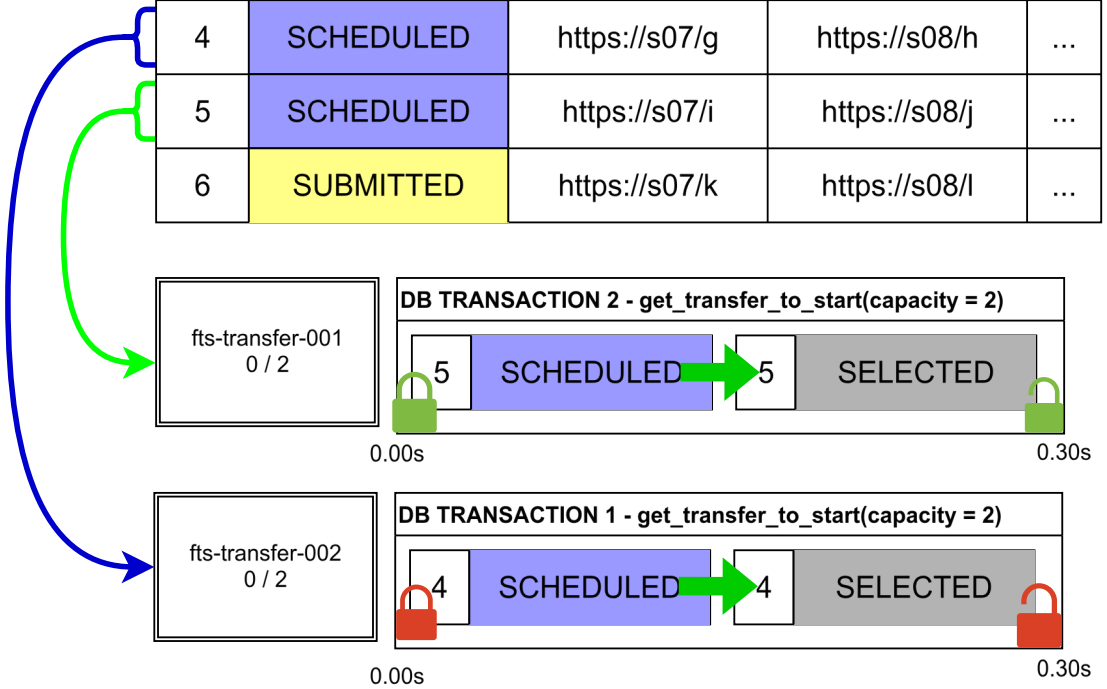
Statistics (t_queue table) - simpflied view of the database				
ID	STATE	SOURCE	DESTINATION	COUNT
42	FINISHED	https://s01	https://s02	1
43	FAILED	https://s03	https://s04	1
44	ACTIVE	https://s05	https://s06	1
45	SCHEDULED	https://s07	https://s08	2
46	SUBMITTED	https://s07	https://s08	1



FTS4 – get_transfer_to_start()

Transfers (t_file table) - simpflied view of the database				
ID	STATE	SOURCE	DESTINATION	...
1	FINISHED	https://s01/a	https://s02/b	...
2	FAILED	https://s03/c	https://s04/d	...
3	ACTIVE	https://s05/e	https://s06/f	...
4	SCHEDULED	https://s07/g	https://s08/h	...
5	SCHEDULED	https://s07/i	https://s08/j	...
6	SUBMITTED	https://s07/k	https://s08/l	...

Statistics (t_queue table) - simpflied view of the database				
ID	STATE	SOURCE	DESTINATION	COUNT
42	FINISHED	https://s01	https://s02	1
43	FAILED	https://s03	https://s04	1
44	ACTIVE	https://s05	https://s06	1
45	SCHEDULED	https://s07	https://s08	2
46	SUBMITTED	https://s07	https://s08	1



New implementation changes:
- We dropped the statistics for most of the transfer states
- Maximise the parallelism between multiple fts-transfer workers
- Moved the pl/pgSQL function from a FOR loop implementation to single batch UPDATE
- Reduces the amount of index look-up, dead tuples, WAL...

FTS4 – Spawn efficiency improvements

Dispatching 10.000 transfers is slow (~250Hz per machine)

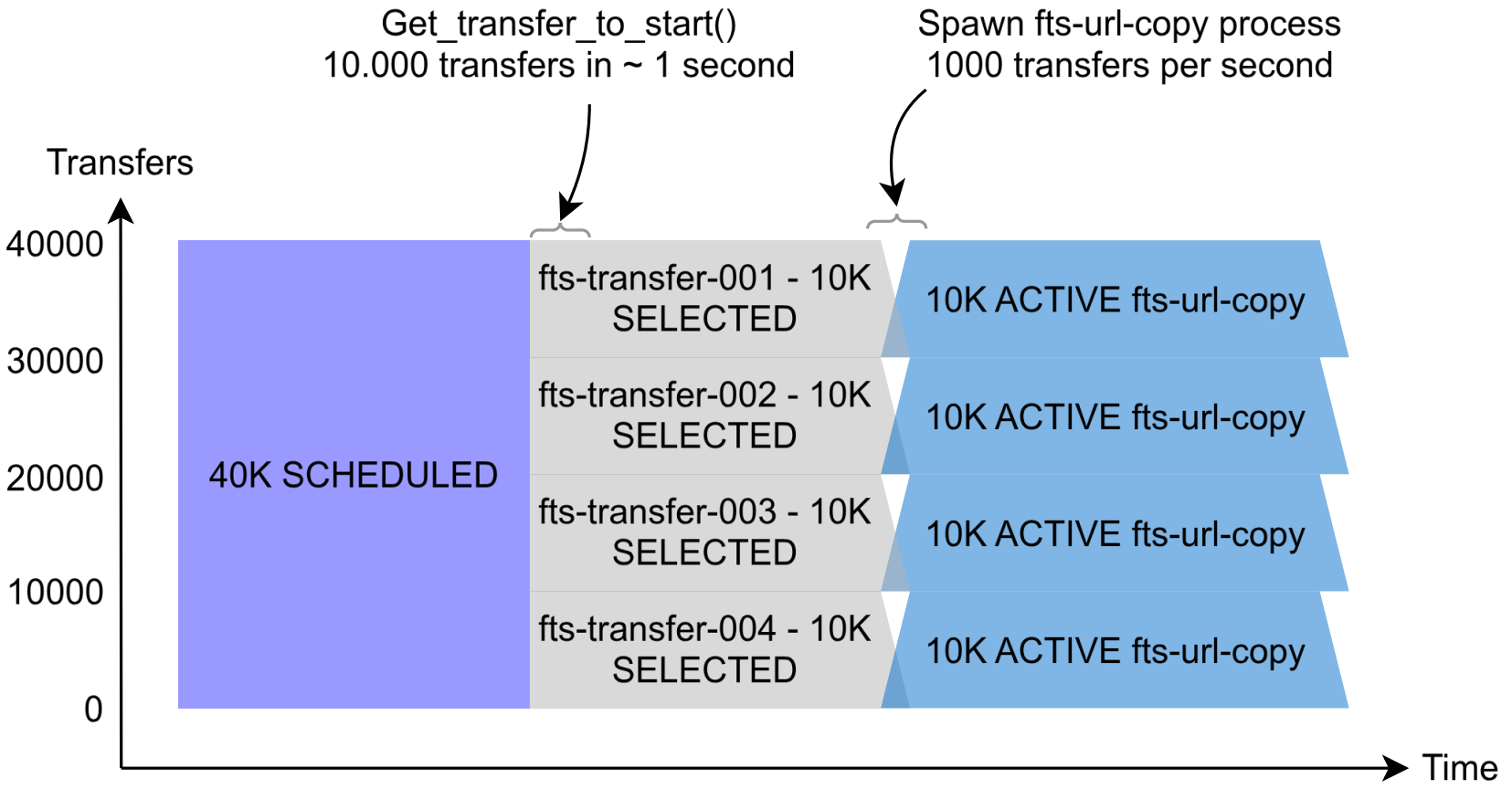
- Moved from fork/exec to posix_spawnp
- Reduced the amount of sockets involved for the errors handling

  → Improved the spawning speed from 250Hz to ~1000Hz

FTS4 – DNS errors

Second blocker: the local DNS cache gets saturated under high load

→ Discovered by ATLAS during joint testing sessions

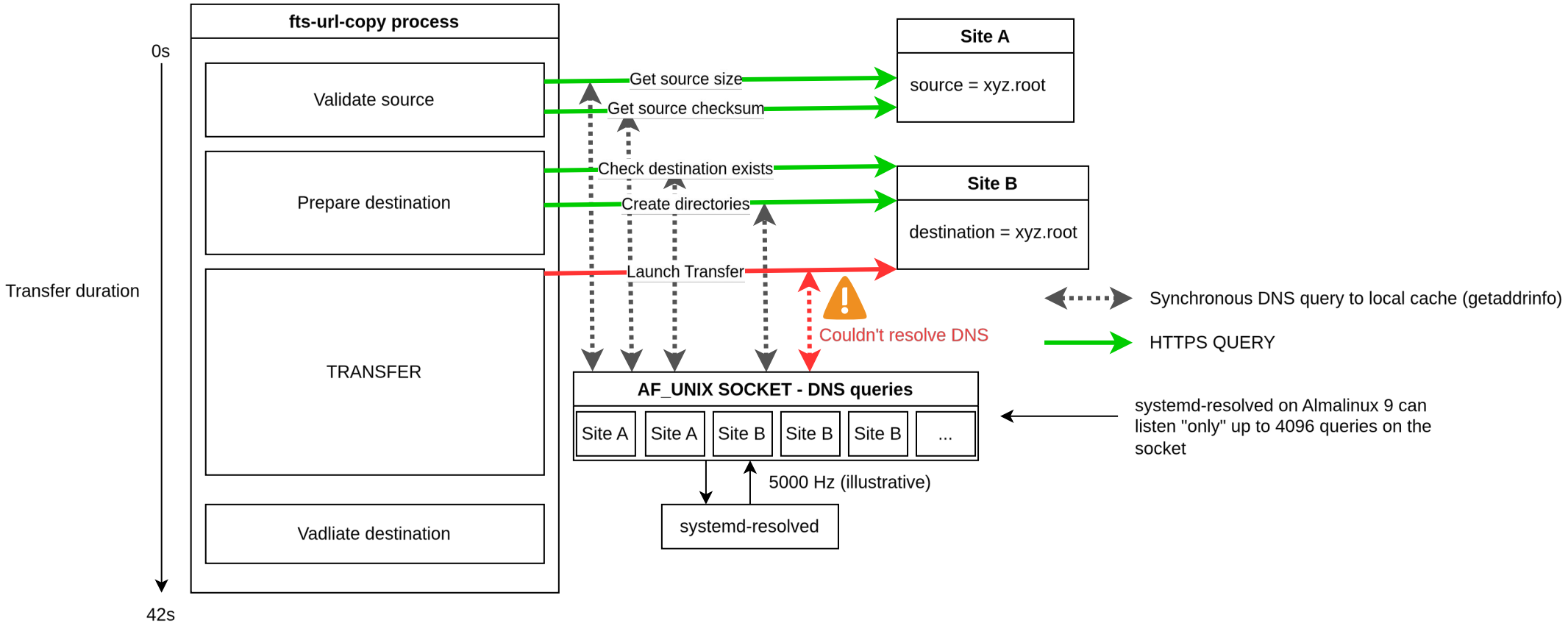


FTS4 – DNS errors

Starting 10.000 transfers quickly saturates the local DNS

→ A single FTS4 cluster can achieve up to 15.000/s DNS resolution queries

→ DNS failures makes **HTTPS query fails**

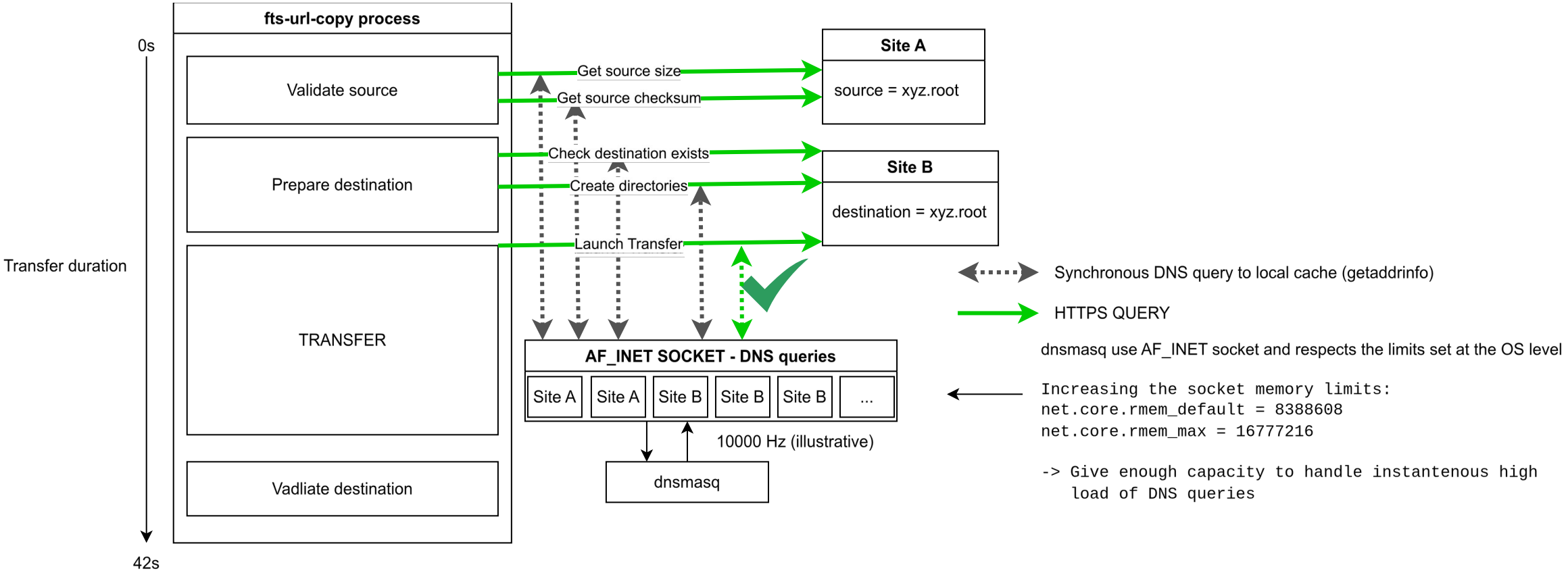


FTS4 – DNS errors

Solution:

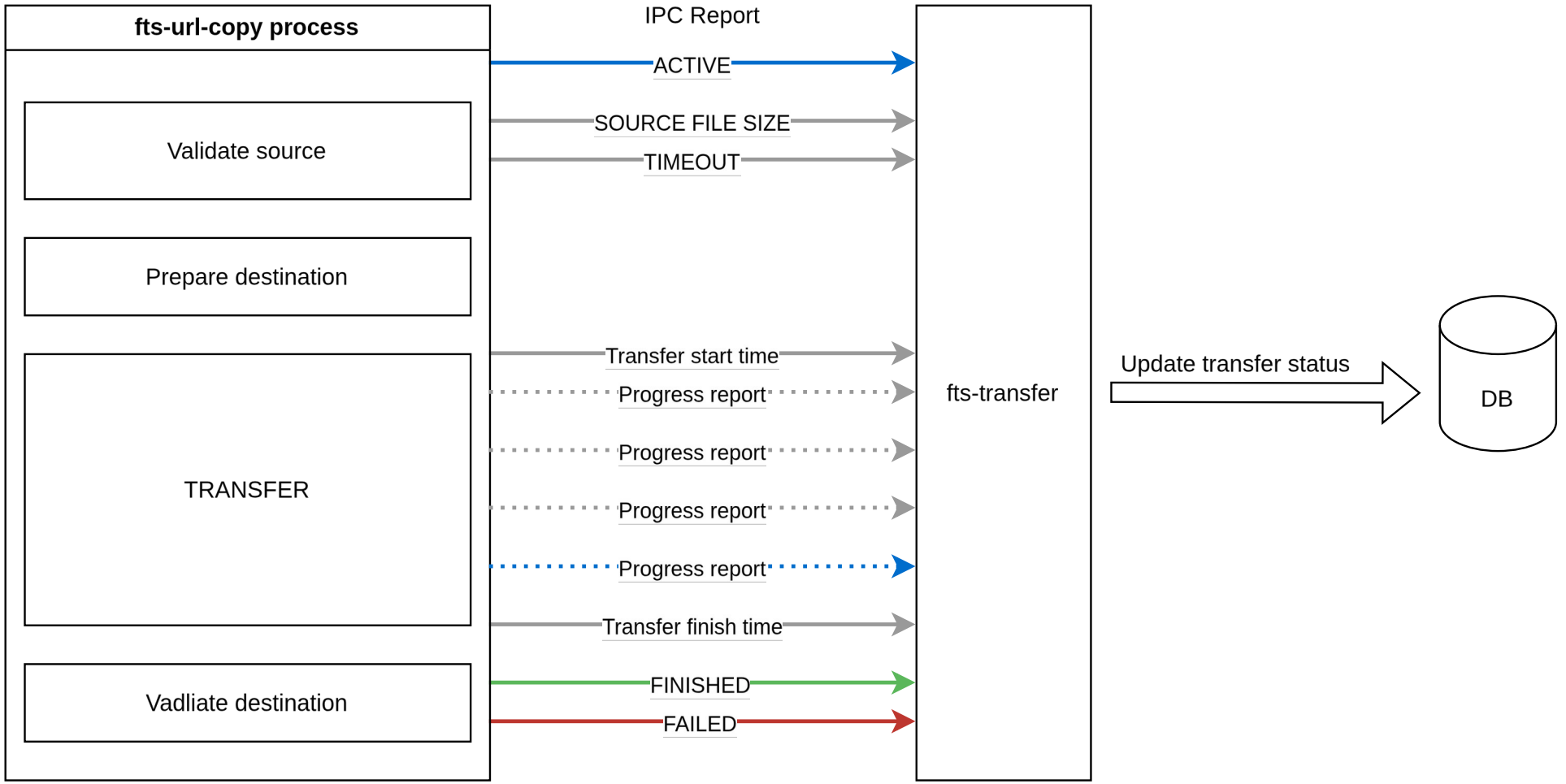
→ Moved from systemd-resolved to dnsmasq

→ **Extended the socket capacity used by the DNS stack to handle the spike of DNS queries of FTS4**



FTS4 – Transfer status reporting

Third blocker: Reporting the transfer activity is slow

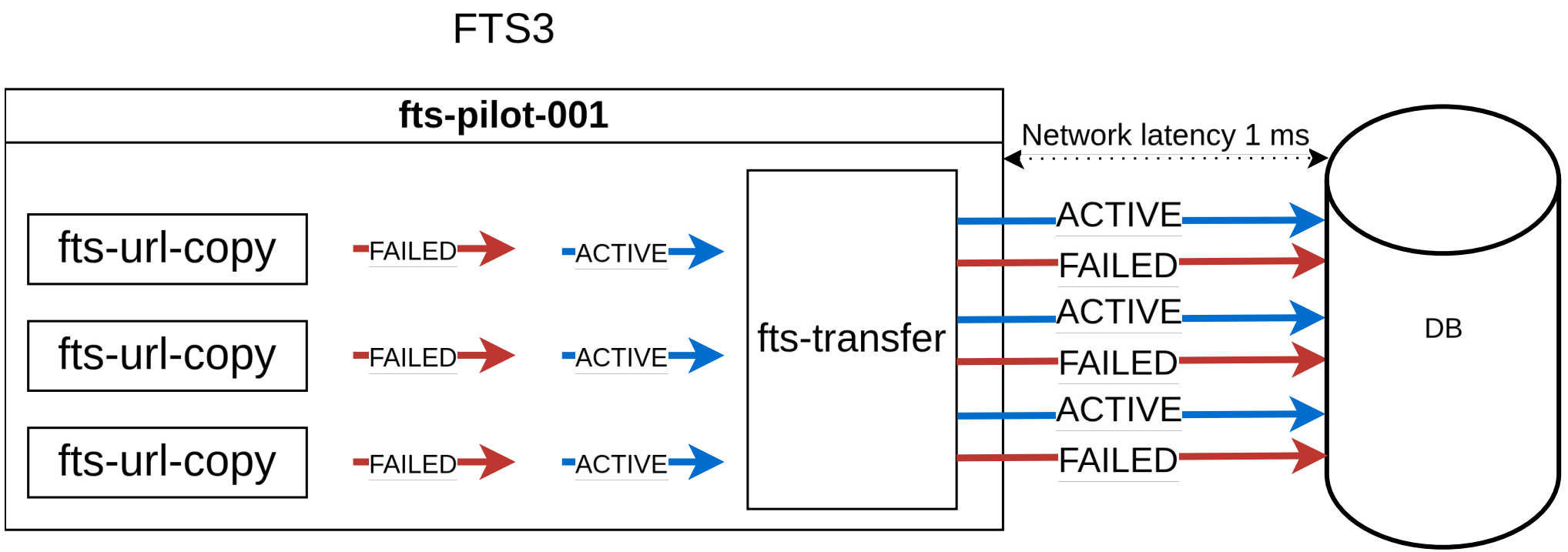


FTS4 – Transfer status reporting

Third blocker: Reporting the transfer activity is slow

→ Reporting 40.000 FAILED transfers can take up to 40s

→ One UPDATE per transfer state transition

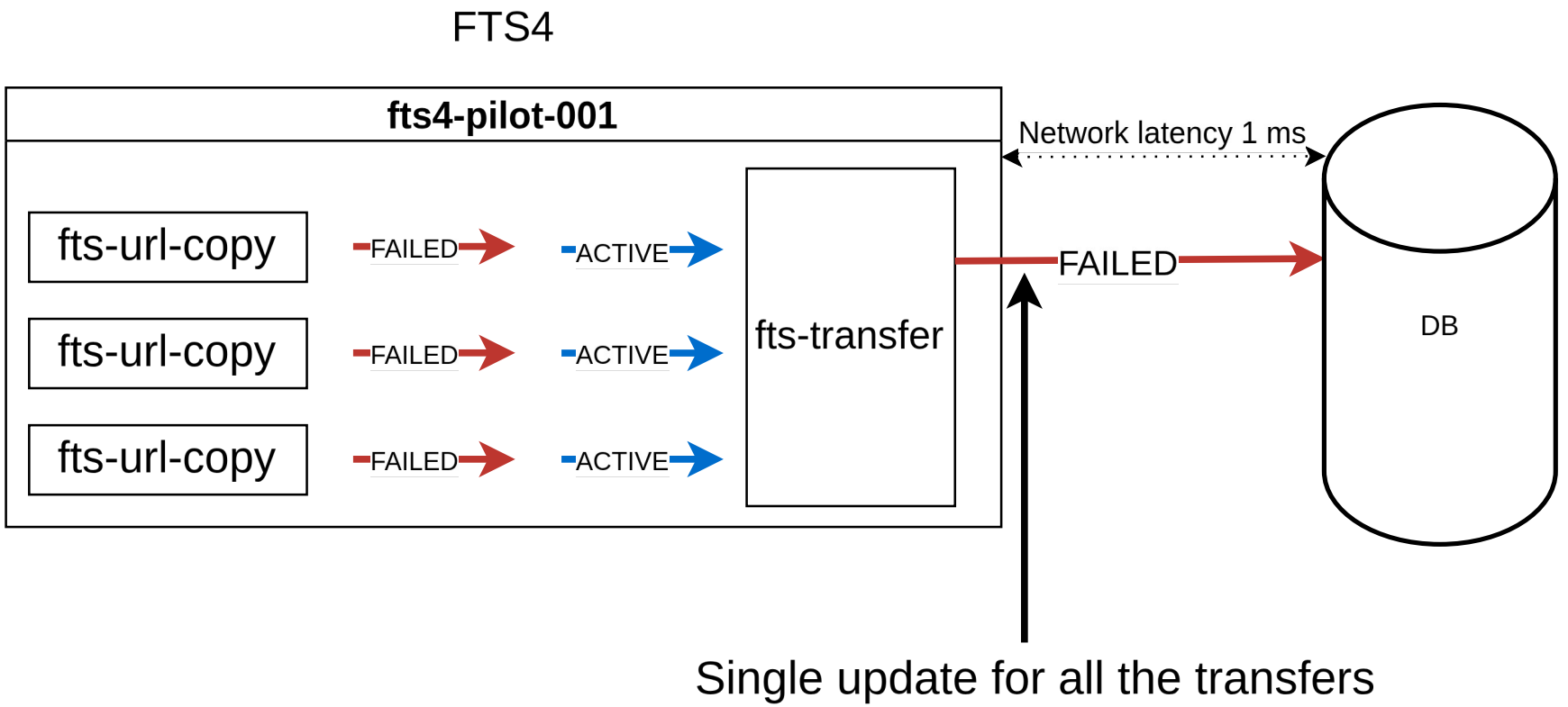


FTS4 – Transfer status reporting

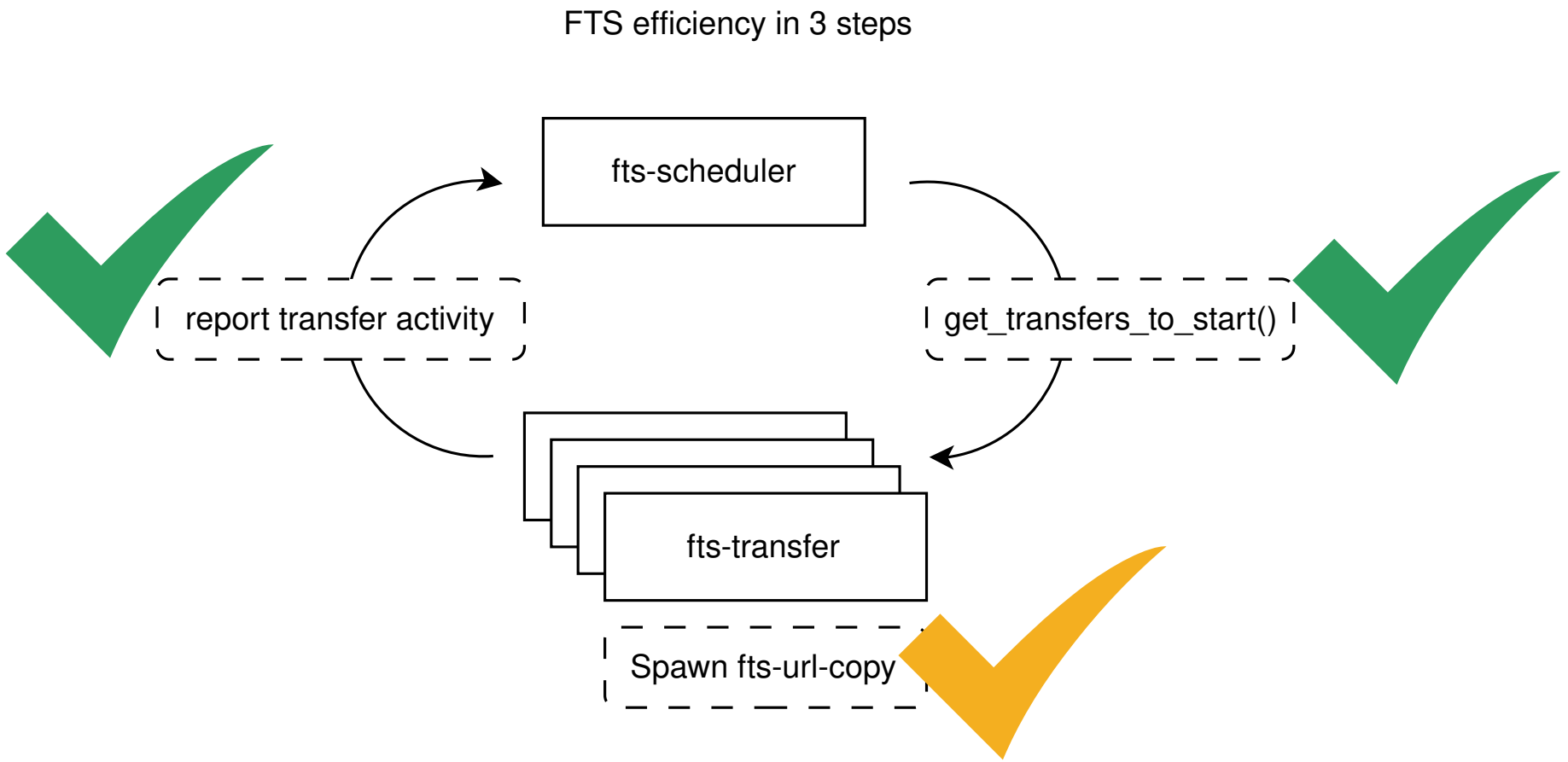
Solution:

→ **Aggregate events and optimise out updates which are not needed**

→ **Fixed size batch UPDATE**



FTS4 – Bottleneck moved to spawning



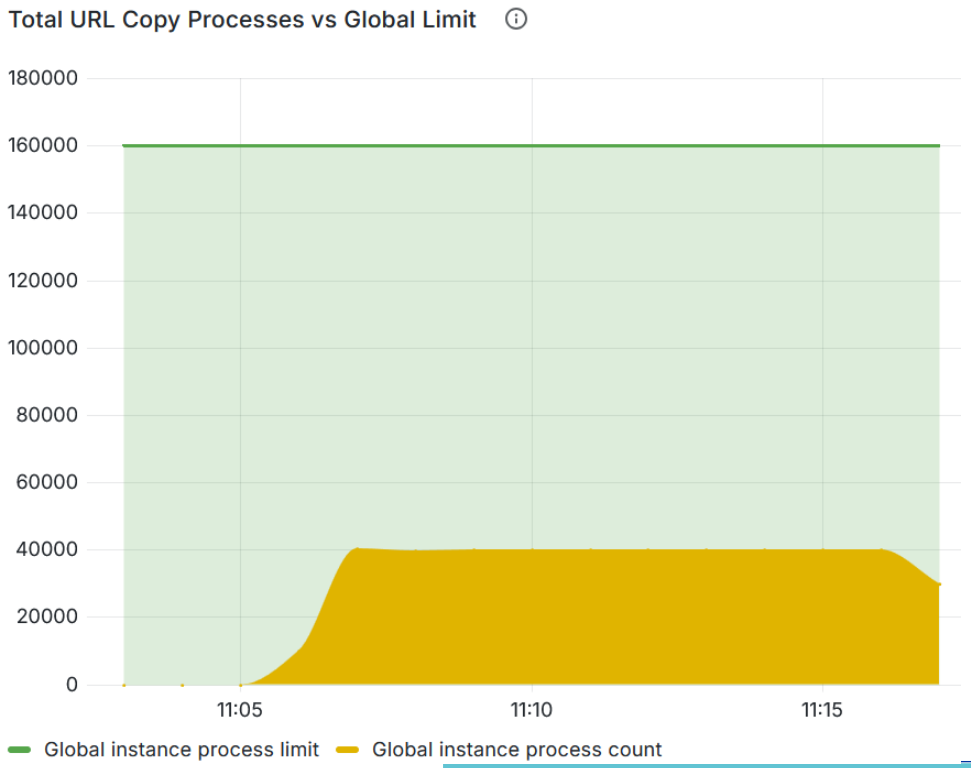
FTS4 – Outcome

Results:

→ FTS4 is now able to handle efficiently 40.000 transfers concurrently

→ Deployed for months on FTS4 clusters (ATLAS, CMS) and showed great results

→ Special Thanks to the ATLAS and CMS experiments as well as the EOS team

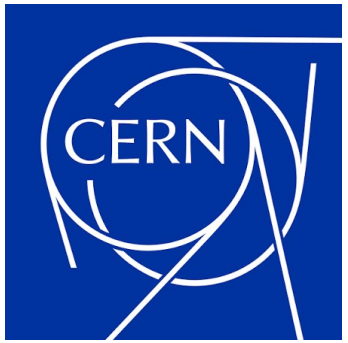


FTS4 – Remaining challenges

- Ensure that the performance is stable when the database reach 20M rows
- Investigate the impact of a single process per transfer
  - → Cost of TLS without session re-use
  - → Is the multi process architecture a blocker to go beyond 40K transfers?

Thanks you!

Questions?



home.cern