High-Performance
Computing Center
Stuttgart

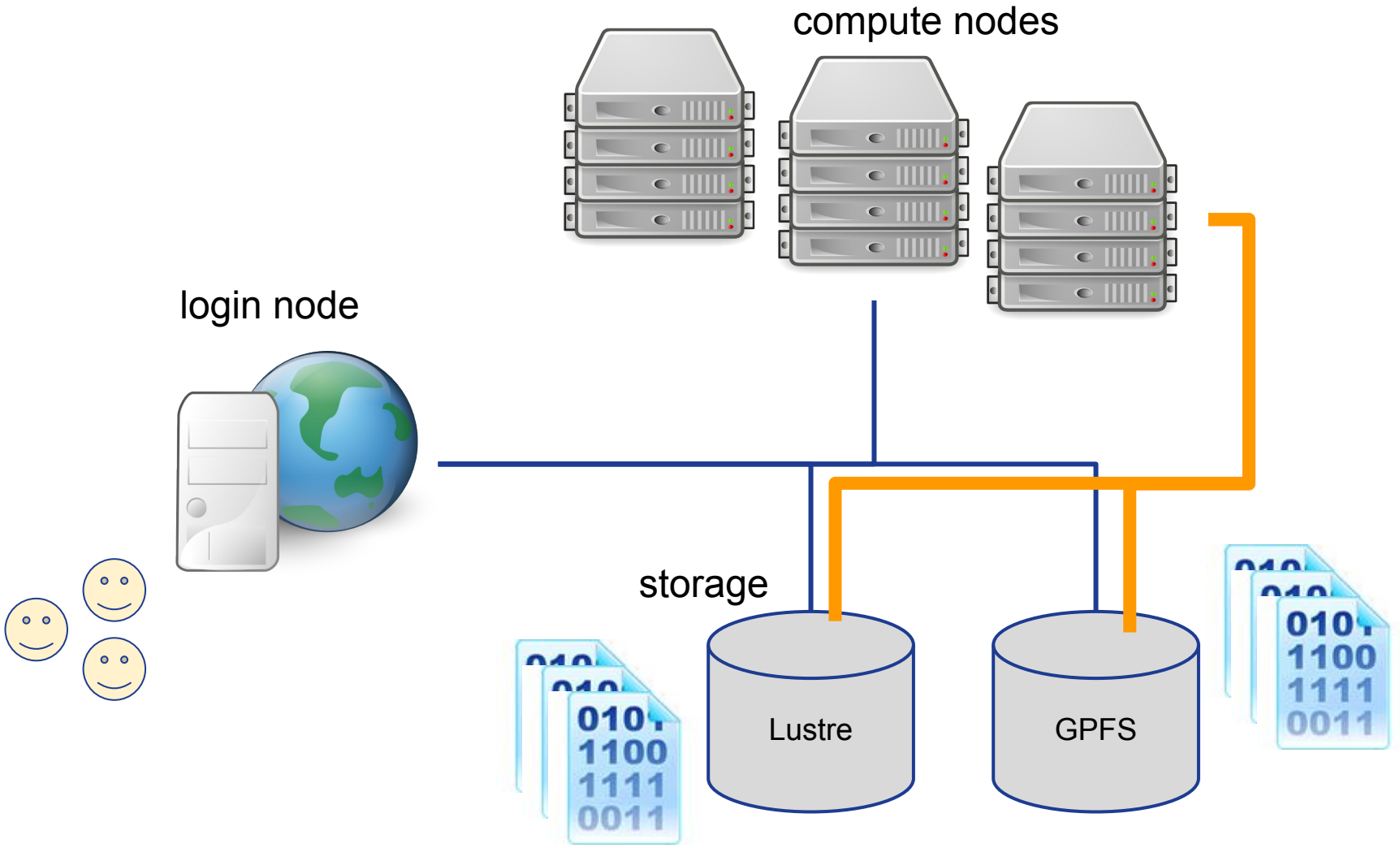
HPC-Workspace: A tool for Data Life Cycle Management

Christoph Niethammer, Holger Berger, Maria Hampel, and Martin Schroschk

Outline

- Motivation
- Background & Related work
- HPC-Workspace
  - Concept
  - Usage and commands
  - Implementation
  - Security aspects
- Best-practices
- Outlook & Conclusion

Example HPC infrastructure



A simple share is not a solution ...

- data life cycle policy enforcement
- transparency of policies
- portability
- security
- efficient resource usage
- administrative effort

Goals

- **manage access** to the fast HPC (scratch) file system for users
- **enforce data life cycle management**
- **prevent full file systems resulting in performance degradation**
- allow load-balancing between various file systems (or within file systems)
- all in a portable way for various HPC platforms and file systems
- minimal maintenance effort

Related work

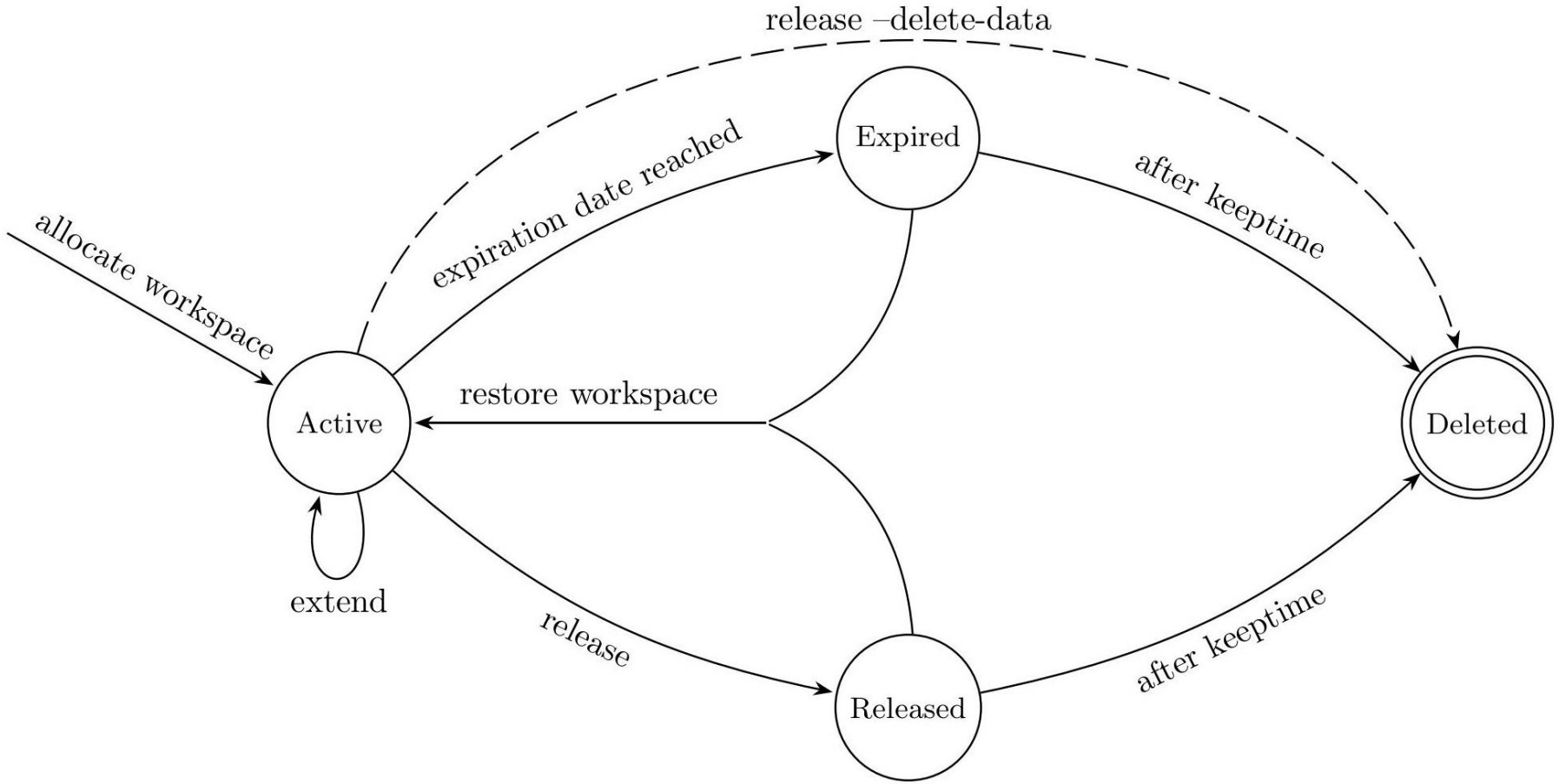
- **High Watermark Deletion**
  - Delete files based on size and age when certain filling of the file systems is reached
  - Deletion process not always transparent to users
- **Hierarchical Storage Management**
  - uses storage systems with different characteristics and automatically migrates data based on access patterns
  - Lacks data life cycle management, i.e., data expiration policy
- **Quota System**
  - uses per user / group limits to prevent exhaustive storage usage
  - lacks life cycle management

HPC-Workspace concept

Introduction of ***workspaces:***

- a workspace is a temporary place for user data
- expires after some time and will be cleaned up/deleted automatically
- has a unique workspace identifier (name) → indirection
- administrator configures on which filesystem resource the associated directory is created
- can be shared with other users if needed

Workspace lifecycle



Background - Unix file permissions

Unix permission model is based on three roles:

- **owner**: owning user, all rights - including changing permissions and group
- **group**: (single) unix group, only granted permissions
- **other**: everybody else, only granted permissions

Provides permissions for accessing directories and files: **read - write - execute**

Additional permissions to provide advanced functionality:

- **SUID**: executable runs as owning user, directory inherits owner
- **SGID**: executable runs as owning group, directory inherits group
- **sticky bit**: prevent anybody except the owner from deleting a file/directory

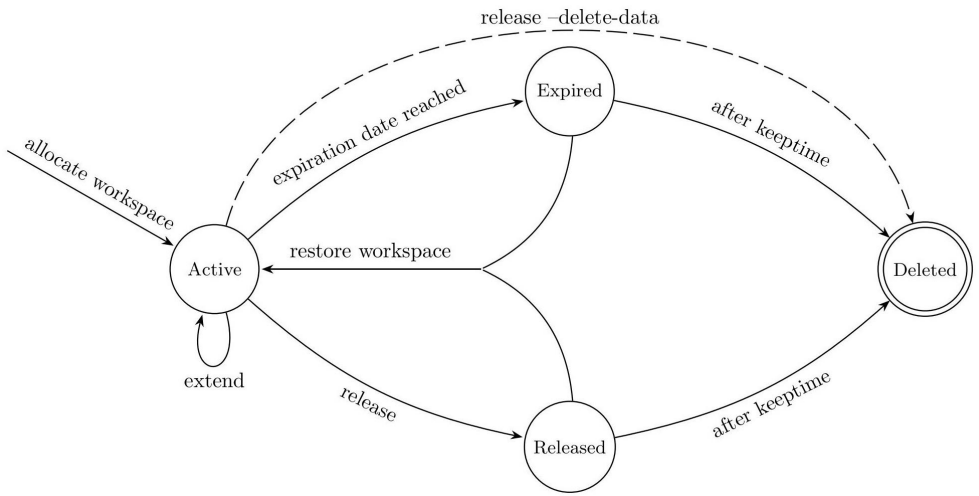
Access control to file system and workspaces

Access to file system and individual workspaces controlled via Unix file permissions:

- base directory of the file system not writable for users
- workspaces can only be created and modified with ws_* commands
- the user allocating a workspace gets the owner role for the workspace directory and access rights are restricted to the owner by default
- metadata, e.g., the expiration date for the workspace, are stored in the workspace DB

HPC workspace commands

command	description
ws_allocate	Allocate a workspace
ws_list	List all active workspaces for a user
ws_find	Find the directory path for a given workspace
ws_release	Release an active workspace
ws_extend	Change the duration of a workspace
ws_restore	List/Restore a released or expired workspace
ws_share	Share workspace resource with users via ACLs



Workspace allocation

ws_allocate [options] <workspace_name> <duration>

- Allocate a new workspace for <duration> days

Example:

Allocate a workspace named “my_workspace” that will expire in 21 days.

```
# ws_allocate my_workspace 21
```

In batchscript:

```
wd = $(ws_allocate my_workspace 21)
cd $wd
```

Listing and finding workspaces

ws_list [options] [pattern]
- List all workspaces for a user

ws_find [options] <workspace_name>
- Find the path for an existing workspace

Example in script:

```
wd = $(ws_find my_workspace)
cd $wd
```

Releasing and deleting workspaces

ws_release [options] <workspace_name>

- Release an active workspace

Example:

Release the workspace my_workspace

```
# ws_release my_workspace
```

- Workspace gets expired immediately and is marked as released
- It can be restored within a short grace period

  → data is still on the file system and counts against the user's quota!

  to delete data immediately and irrecoverably use the --delete-data option

Extending a workspace

ws_extend [options] <workspace_name> <days>

- set the expiry date to be in <days> days and consume one available extensions until no extensions are left for this workspace

Example:

Extend the workspace “my_workspace” to expire in 14 days

```
# ws_extend my_workspace 14
```

(implementation detail: ws_extend is a thin wrapper around ws_allocate -x)

“Workspace expirer” - Admin Tool

Data life cycle policies applied by “workspace expirer” tool:

- workspace state changed to “expired” if its lifetime is exceeded
- workspace directory is moved to a hidden directory on the same file system that is not accessible to users
  - “fast deletion”
  - no data is removed → data still occupies disk space
- short grace period started → workspace can be restored
- after grace period → workspace and its data are irretrievably deleted
- can be automated with a cron job

ws_expirer [options]

Restoring a workspace

ws_restore [options] <workspace_name> <target_name> | -l

- Restore a released or expired workspace to another workspace

Example:

Restore the released workspace “my_workspace” to the already existing workspace “another_workspace”

```
# ws_restore -l   # this gives the IDs of workspaces that can be restored
usera-my_workspace-1762315512
       unavailable since Wed Nov  5 05:05:12 2025
# ws_restore usera-my_workspace-1762252513 another_workspace
```

Group workspaces

Use case:

- Share workspace with colleagues, e.g., common data set directory

Condition:

- All collaborators need to be in the same group
- Has to be specified at time of workspace allocation (`ws_allocate -G <GID>`)

- SGID used for subdirectories to inherit permissions
- Group can get read-only access or read and write access

Beware: Write access for collaborators! → quota & cp & ws_release!

Background - Linux Access Control Lists (ACLs)

- ACLs enable fine-grained access control, e.g., access for individual users
- File system must be “ACL-enabled”
- Tools and commands processing files must handle ACLs – attention with cp!!
- Users can list ACLs with the getfacl command:

```
# standard unix permissions are are a subset of ACLs:
[christoph] # ls -l /tmp/hello
-rw-r--r-- 1 christoph christoph 0
[christoph] # getfacl /tmp/hello
# file: hello
# owner: christoph
# group: christoph
user::rw-
group::r--
other::r--
```

Background - Linux Access Control Lists (ACLs)

- ACLs can be modified with the setfacl command:

- ACL types:
  - users
  - groups
  - mask
  - default

```
# Add read permissions to the http user:
[christoph] # setfacl -m user:http:r /tmp/hello
[christoph] # ls -l /tmp/hello
-rw-r--r--+ 1 christoph christoph 0
[christoph] # getfacl /tmp/hello
# file: hello
# owner: christoph
# group: christoph
user::rw-
user:http:r--
group::r--
other::r--
```

Share existing workspaces

ws_share <share|unshare> [options] <workspace_name> <user>

- give read access to an existing/active workspace via ACLs
  - e.g., for support staff to work on bugs or share software installations with others

Example:

Share the workspace another_workspace with userb

```
# ws_share share another_workspace userb
```

Beware: ACLs might not grant access if data are copied into a workspace after using ws_share; path to workspace has to be exchanged by users manually

History of HPC-Workspace

H L R S

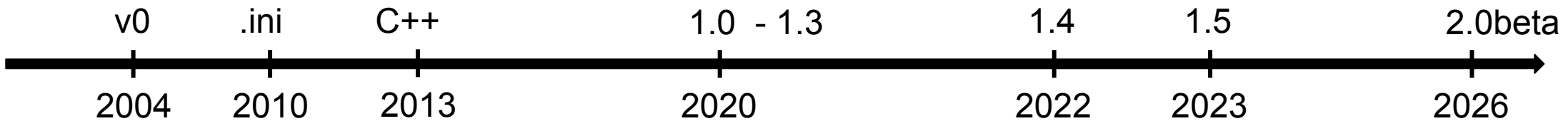
version 0
- some python tools
- DB format simple list
- unstructured simple config file
- later .ini file

version 1
- C++ for SUID programs
- new YAML based configuration

version 1.3
- Group workspaces (1.3.5)

version 1.4
- ws_share
- user default settings in ~/.ws_user.conf

version 2
- complete rewrite in C++17
- switch to “modern cmake”
- modularisation
- extensive test setup
- lots of code hardening …
- improved logging



Global configuration file

Configuration file is stored under */etc/ws.conf*

```
clustername: vulcan                         # name to identify the system
admins: [root]                              # list of admins
default: ws3                                # default workspace
dbuid: 1                                    # user id, owner of workspace top-level directories
dbgid: 1                                    # group id, owner of workspace top-level directories
duration: 60                                # max. duration in days
maxextensions: 3                            # number of extensions
[...]                                       # other options like default lifetime settings
workspaces:
  ws3:                                      # name of workspace location
    database: /lustre/ws3/var/ws/db         # DB directory
    deleted: .removed                       # subdirectory for expired workspaces
    keeptime: 5                             # time in days to keep the workspace after expiration
    [...]
    spaces: [/lustre/ws3/ws]                # list of locations for workspace directories
```

Workspace Database

The list of workspaces and associated metadata is stored in ws database:

- DBv1 implemented as directory tree
- Full **ACID** is not really needed → each DB entry is a file in the directory
- Users have read-only on their own DB entries
- POSIX file system semantics offer atomicity and uniqueness of filenames

```
# DB entry of an active workspace

$ cat /lustre/ws3/var/ws/db/user-test
workspace: /lustre/ws3/var/ws/user-test
expiration: 1762339037
extensions: 10
acctcode: forschung
reminder: 0
mailaddress: ""
comment: ""
```

```
# DB Entry of a released workspace

$ cat /lustre/ws3/var/ws/db/.removed/user-test-1762256796
workspace: /lustre/ws3/var/ws/user-test
expiration: 1762256796
extensions: 10
acctcode: forschung
reminder: 0
mailaddress: ""
released: 1762256796
comment: ""
```

HPC workspace security: Elevated permissions

Various parts of HPC-workspace need elevated or even root permissions:

- database manipulation (ws_allocate, ws_release, ws_extend, …)
- creation and ownership changes (ws_allocate, ws_release)
- expiration / deletion of workspaces (ws_expirer)

Two supported methods:

SUID:

+ works on all unices
+ works on all filesystems
- coarse grain

Linux capabilities:

+ allow restriction for specific permissions
  (e.g. CAP_CHOWN, CAP_DAC_OVERRIDE)
+ works on root_squash lustre
- require OS and file system support

HPC workspace security: Elevated permissions

- ws tools drop privileges when dealing with user provided data
- elevation only when needed, e.g. accessing filesystem

Example - moving workspace to removed area:

```
caps.drop_cap(...); // drop all not necessary permissions
...
caps.raise_cap({CAP_FOWNER, CAP_DAC_OVERRIDE}, ...);      // raise (needed) permissions
cppfs::rename(wsfilepath, removepath);                  // perform action requiring permissions
caps.lower_cap({CAP_DAC_OVERRIDE, CAP_FOWNER}, ...); // lower permissions again
...
```

More security aspects

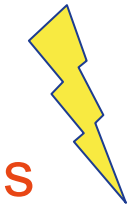
- very strict input validation
  e.g., workspace names restricted to avoid any issues with unicode complexity
- using C++ STL types wherever possible for type checking
- avoiding raw pointers to minimizes risks of memory safety issues
  if raw pointers are unavoidable use gsl::nonull<T*>

- number of setuid tools kept at a minimum
- static binaries
- sanitizer checks

Software testing

- Unit tests with Catch2
- CI/CD pipeline running static code analyzers (docker based)
- *bats* integration and command tests
- various sanitizers in test runs
- VM test setup to test suid/capability functionality (not possible in docker environment)
- test coverage checking

Best Practices - as a User



- Use ws_find <workspace-ID> instead of hard coded paths
- Use multiple workspaces, to help load balancing (important on lustre DNE)
- Use ws_register to keep track of your workspaces (manages links)
- Get calendar entries via ws_ical to keep track of expiration dates
- Set mail reminder at allocation
- “ws_quickcheck“

Best Practices - as an Administrator

- Disable expirer during maintenance periods!
- DBv1 directories are good candidates for DOM (data on metadata) directories in lustre, files are very small
- always do a ws_expirer dryrun when changing anything
- put DB in same file system as data if possible to ensure that always both are available at the same time (expirer checks since a while if DB is really a marked DB)

- V2 brings ws_editdb to, e.g., alter workspace expiration to compensate downtimes

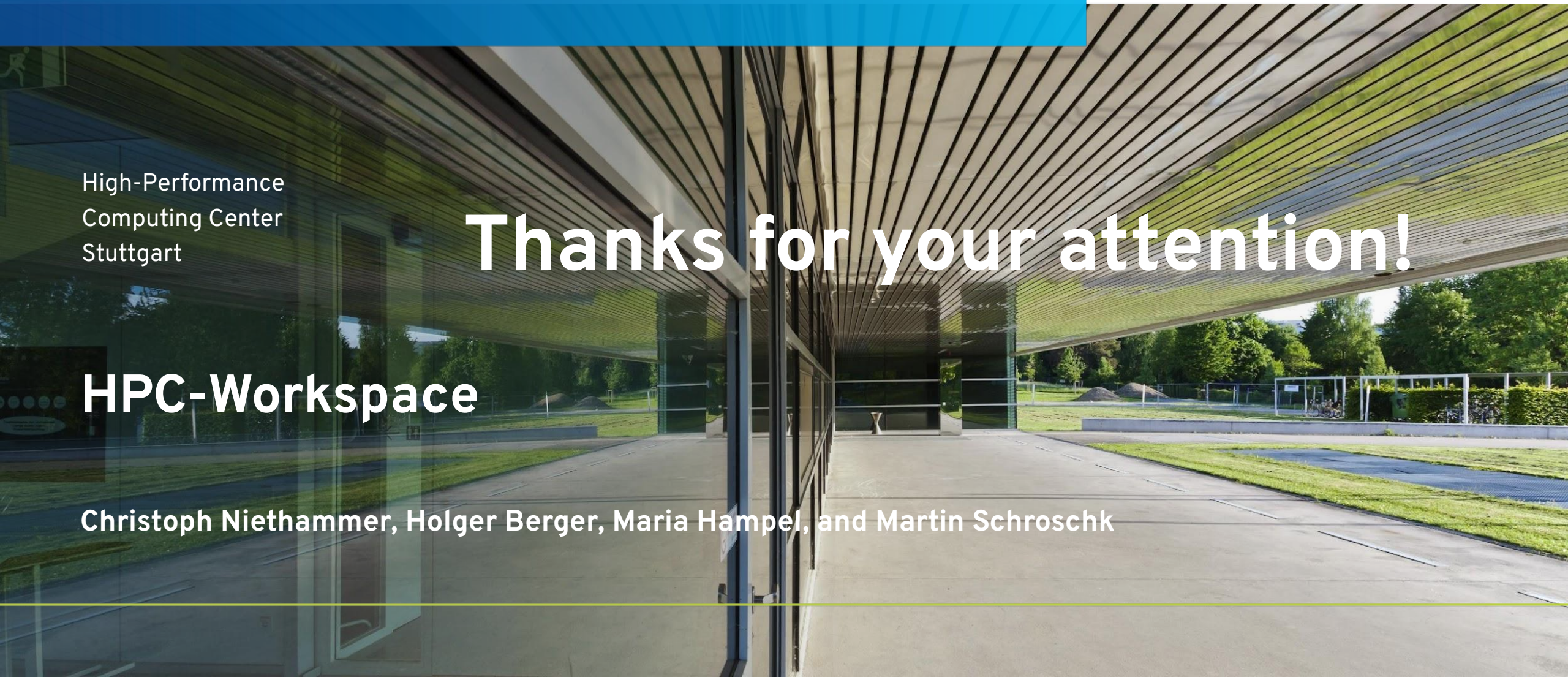
Outlook

HPC-Workspaces v2:

- Performance improvements
- improve backward compatible with version 1
- split configuration using several files
- new options for some tools (e.g. deletion of expired workspaces)
- more consistent command options (e.g. group workspaces)
- apply more security techniques, e.g. STL hardening
- even more tests

Conclusion

- HPC-workspace is a proven tool for data life cycle management in HPC
- Good user acceptance due to stable and scriptable UI
- Version 2 coming soon with many improvements ...



High-Performance
Computing Center
Stuttgart

Thanks for your attention!

HPC-Workspace

Christoph Niethammer, Holger Berger, Maria Hampel, and Martin Schroschk