NEW SYSREPO

Michal Vaško
CESNET

24. August 2018
Brno

DESIRED PROPERTIES

- **sysrepo is ideally meant to handle all system configuration**
  - absolutely critical core software
- **MUST be**
  - stable
  - robust
  - reliable
- **SHOULD be**
  - simple
  - small
  - without unnecessary dependencies
  - fast
  - with simple API for clients
  - customizable for low memory footprint or time efficiency

REMOVED FEATURES

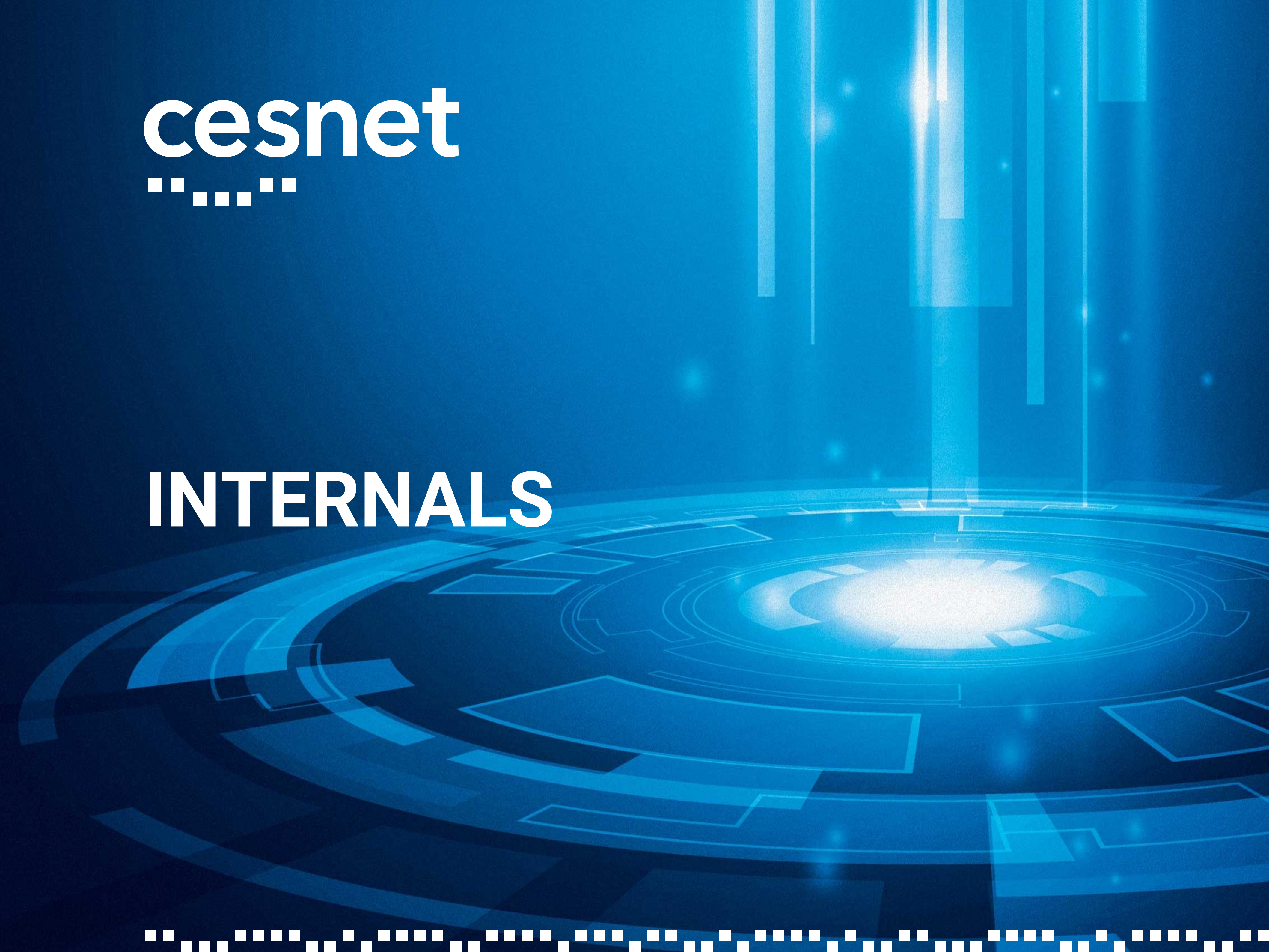
- **Some sysrepo features will be removed**
  - gain of having the feature does not justify the added complexity
- **No sysrepod daemon**
  - only client library
- **No dependencies**
  - most likely none of the current ones will be required
- **No schema dependency tracking**
  - only data dependencies (still non-trivial – leafref, instance-identifier, must, when)
- **No NACM**
  - only current file access control (is it sufficient?)
- **No SR_EV_VERIFY?**
  - keep current functionality (but rename enum values)
  - SR_EV_DONE (one callback call for each subscriber)
- **No session-exclusive changes**
  - candidate **datastores** for that

NEW SYSREPO DESIGN

- **All IPC using shared memory**
  - same synchronization (mutex, conditional variables or semaphore)
  - data serialization problems
- **One context with all installed models**
  - data tree and notification files separate for models
- **Installing new model**
  - as currently – permissions, owner
  - optional replay support

NEW SYSREPO DESIGN

- **NMDA support**
  - startup, running, candidate, intended, operational datastores
  - no need for enabled/disabled modules mechanism
  - conventional datastores with all the data
  - operational datastore with only subscribed to/provided data by clients
- **Data subscriptions**
  - these are the “enabled” data appearing also in operational datastore
  - optionally, define a separate callback for getting these specific configuration data to appear in operational datastore
  - callback for state data appearing in operational datastore



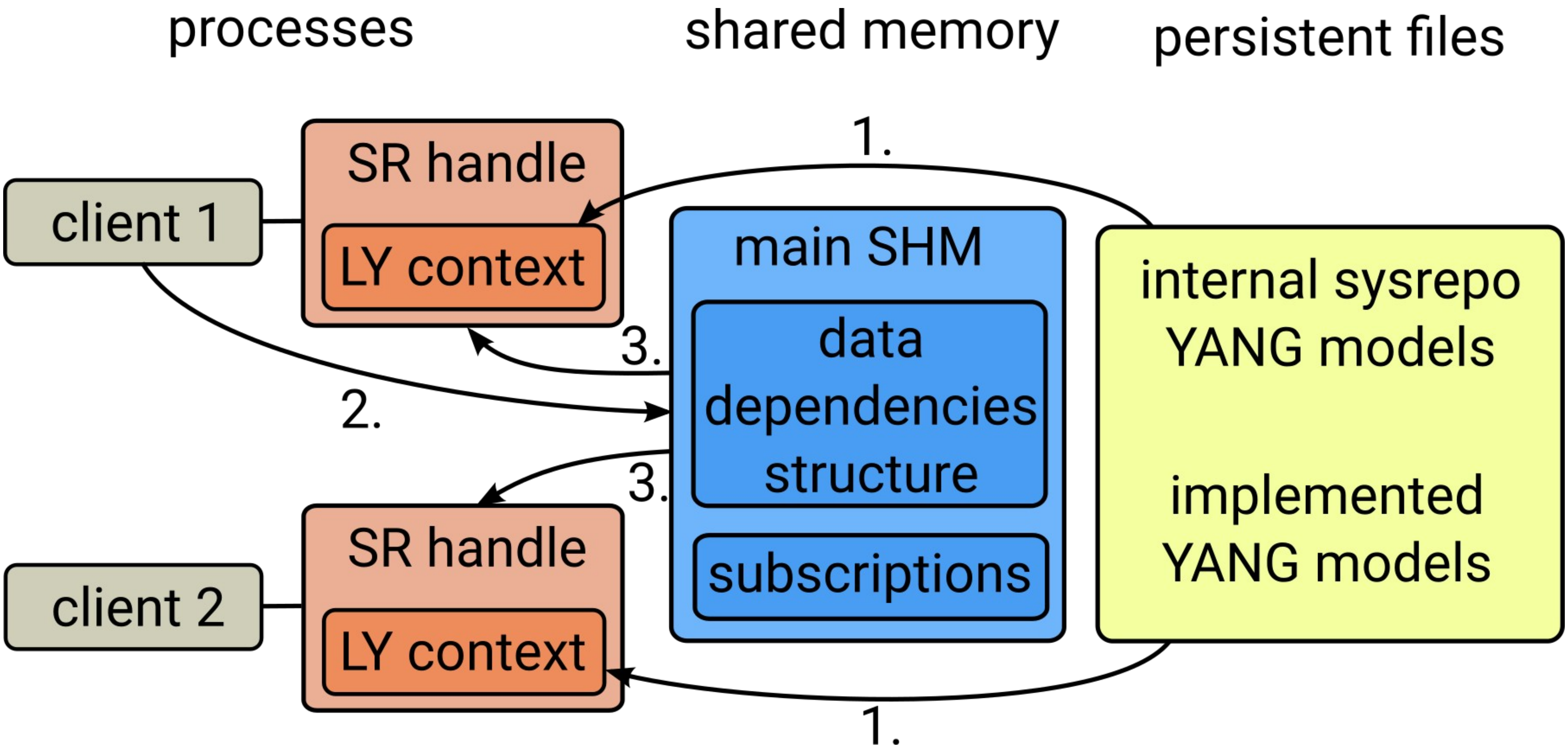
cesnet

INTERNALS

SR_CONNECT

- **sr_connect()**
  - every client starts with this call to get a global handle
  - handle is thread-safe (uses locks internally)
  - creates local LY context *(1.)*
    - in future, this could be a “compiled” LY context
    - smaller context with only information required for data handling (could not be printed)
    - ideally, also shared between all clients so only 1 instance is needed
  - connects to the **main SHM** *(3.)*
    - creates if does not exist *(2.)*
    - creates data dependencies structure from stored data
    - list of implemented models with data dependencies between them
  - this internal global handle is required for all other API calls

SR_CONNECT


processes
shared memory
persistent files
client 1
SR handle
LY context
client 2
SR handle
LY context
main SHM
data dependencies structure
subscriptions
internal sysrepo YANG models
implemented YANG models
1.
2.
3.
3.
1.

MAIN SHM

- **Created from persistent data files**
  - sysrepo-modules data tree
  - sysrepo-notification-store data
  - additional runtime data
    - current subscriptions
- **Data serialized into main SHM**
  - including strings
  - into data directly readable by all the clients
    - no parsing required
  - should be possible to effectively read each data structure separately
  - implemented as several separate SHMs so they can be dynamically resized without affecting other data structures

MAIN SHM

sysrepo-modules

- currently sysrepo-module-dependencies.yang and sysrepo-persistent-data.yang, merged into one model
- list of implemented modules with filepaths, enabled features
  - imports are loaded automatically
- foreign leafref, instance-identifier, must, when node XPaths
  - when a new model is installed, dependencies and all the existing contexts are updated

MAIN SHM

- **sysrepo-notification-store**
  - currently sysrepo-notification-store.yang
  - *different content* from the existing model
  - model-specific information
    - model, revision, replay support
  - model-specific notification file in a fixed (relative) path
  - notification file format
    - similar to what is defined in this model
    - instead of a list, just one notification after another in LYB
    - much faster notification storing

MAIN SHM

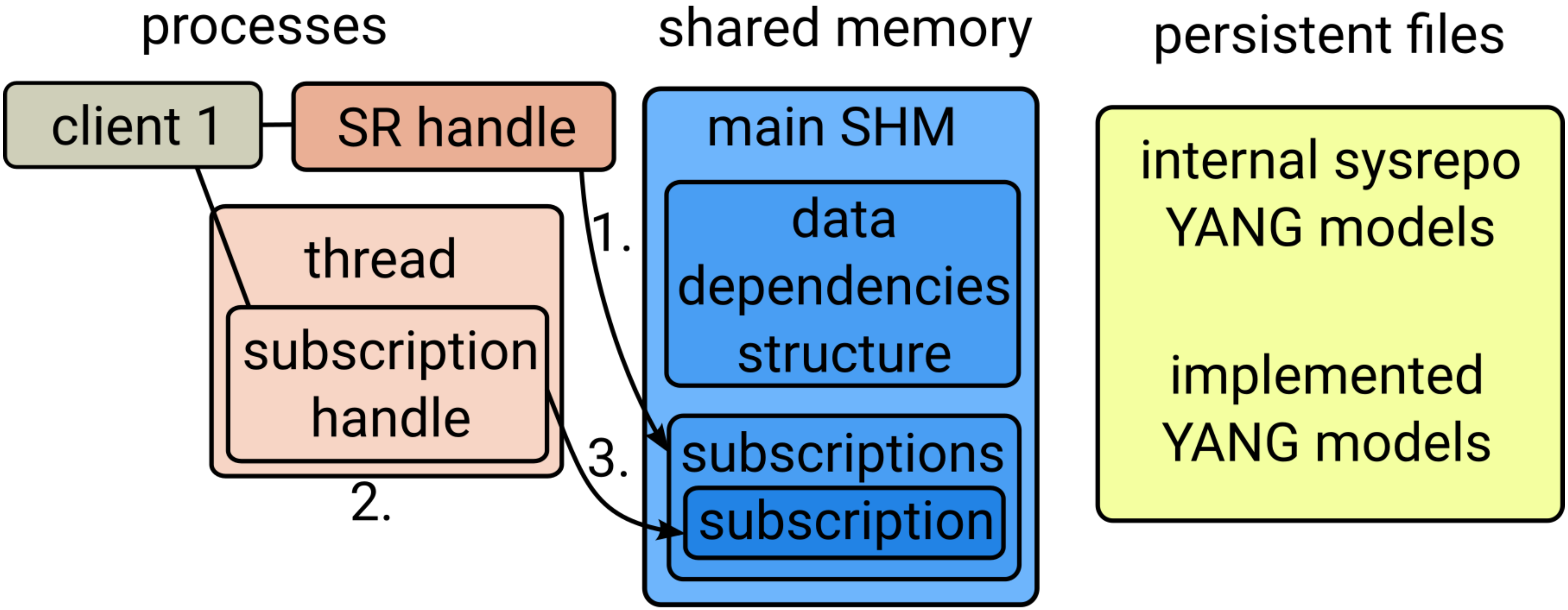
Current subscriptions

- serialized information about subscriptions
- implemented as many model-specific SHMs
- subscriptions are grouped based on their path
  - one subscription for 1 – n subscribers for a specific path
- each subscription
  - path
  - signaling synchronization item for originator/subscribers
  - space for **secondary SHM** segment identification
  - the originators will communicate data to the subscriber using this separate SHM segment

SR_*_SUBSCRIBE

- ## sr_*_subscribe()
  - checks that the subscription path is valid
  - adds this subscription to **main SHM** *(1.)*
  - returns a subscription handle *(2.)*
  - reuse of subscription handles (as it is currently)
- ## Threading model
  - as it is now – one subscription handle for 1 - n subscriptions
  - but each subscription handle is tended to by one thread
    - means it is waiting for signal in its **main SHM** subscription *(3.)*
  - should achieve easy and flexible threading model customization
  - keeps current API

SR_*_SUBSCRIBE



SECONDARY SHM

- **Configuration change subscription**
  - covers module-change, subtree-change
  - diff
    - changes
    - instead of pointers, the affected nodes themselves in LYB
    - LYB top-level subtrees with an attribute on the node-pointed-to in diff
  - synchronized counter of subscriptions to process the diff changes
  - synchronized flag whether the commit should continue (or failed)
  - space for ID of SHM segment with error information if flag is set
- **Operational data subscription**
  - covers dp-get-items (both state and configuration data)
  - XPath of the requested subtree
  - space for ID of SHM with returned LYB operational data subtree
  - new API could request all the data in one (or a few) call
    - internally working like that (library waits until the whole subtree is returned)

SECONDARY SHM

Schema change subscription

- covers module-install, feature-enable
- name + revision of the module (+ feature name)

Operation subscription

- covers rpc, action, event-notification
- LYB tree of the operation
- space for ID of another SHM with a reply

SR_SET_ITEM

Candidate datastores

- in API only one candidate
- internally there will be one (other than the API-one) candidate datastore tied to every other writable datastore

sr_set_item(_str)()

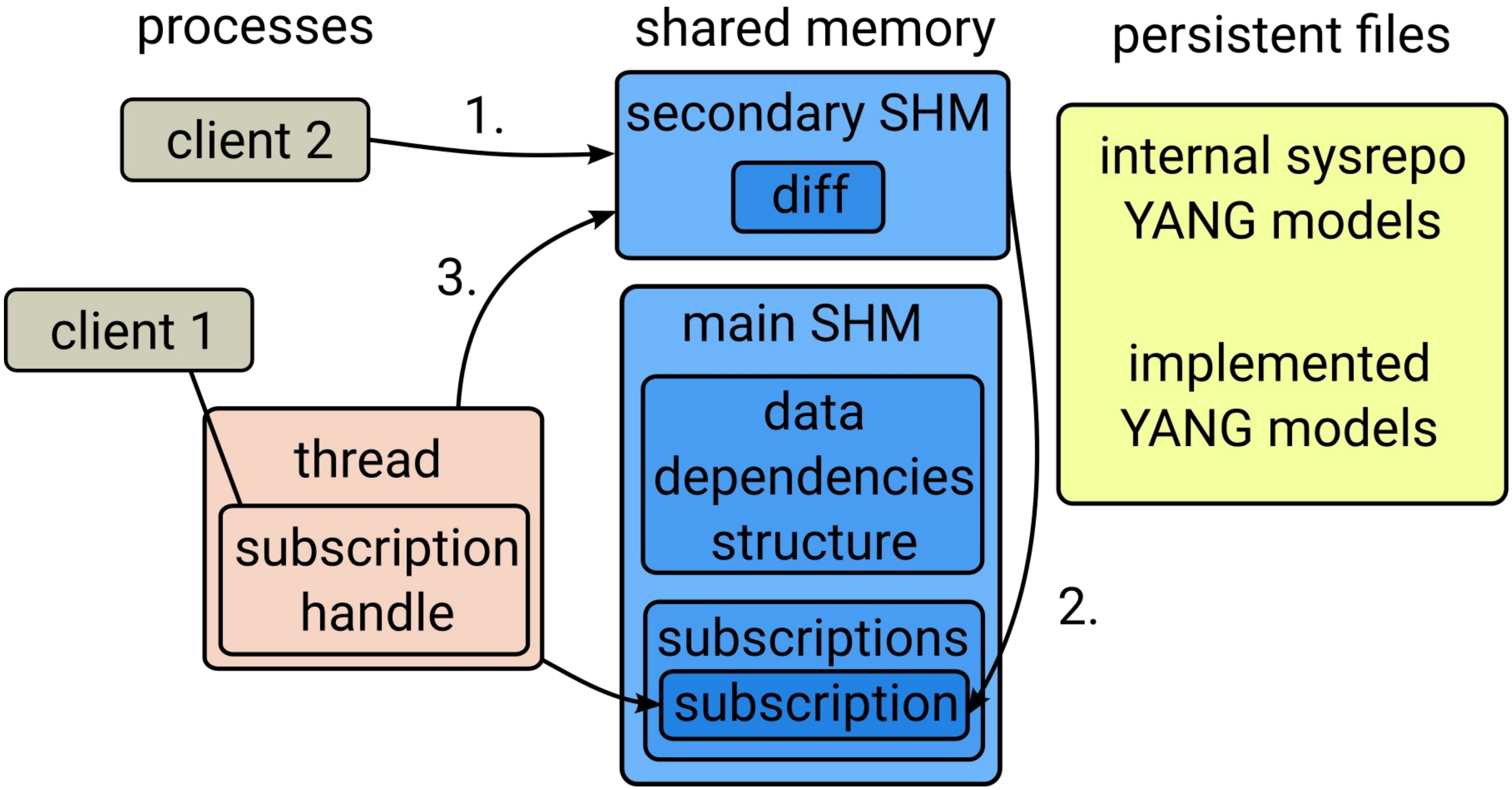
- if no candidate datastore tied with the target datastore, create it
  - make a copy of the target DS (only the affected model data trees)
- perform the small change in the candidate DS (if possible)
- change is client-exclusive (non-visible for other clients)

SR_COMMIT

sr_commit()

- rename to sr_apply_changes()?
  - confusing collision of terms with NETCONF commit
- create diff comparing the current DS content with this candidate DS
- transform, provide it in the **secondary SHM** *(1.)*, and signal subscriptions having filled **secondary SHM** ID *(2.)*
  - first process subscription that is changing values, then all others
- subscribers can now access **secondary SHM** using the ID and process changes *(3.)*
- waits for subscriber counter to be 0 or fail flag set
- returns success or error
  - in case of an error, it does not wait for subscriptions abort

SR_COMMIT


processes
shared memory
persistent files
client 2
1.
secondary SHM
diff
3.
client 1
main SHM
thread
subscription handle
data dependencies structure
internal sysrepo YANG models
implemented YANG models
subscriptions
subscription
2.

SR_COMMIT() SUBSCRIBER

- **sr_commit() subscriber**
  - gets notified
    - based on its priority, maybe not all subscribers at once
  - attaches to **secondary SHM** with diff and synchronization
  - processes diff into desired format
  - in loop until no changes left
    - checks fail flag (breaks if true)
    - applies another change (calls callback)
    - stores rollback information
  - on success decreases subscription counter
    - but must wait until it reaches zero or fail flag is set
    - then clears rollback data and finishes
  - on error reads the error information and provides it to client
    - must be clear that some other subscription generated the error
  - on failure rollbacks all applied changes disregarding whether it succeeds or not

SR_GET_ITEM(S)()

- **sr_get_item(s)()**
  - finds all configuration data subscriptions without separate operational data callback
    - gets these nodes from current datastore and creates a tree
  - finds all relevant operational data subscriptions
    - requests these data one subtree after another for each subscription
    - merges them into the resulting tree
  - returns the final tree
    - converted to whatever format was requested
  - reading the data from datastore and requesting operational data for each subscription could occur in parallel
    - but probably wait with it and consider it an optimization

OPERATIONAL DATA SUBSCRIBER

- **sr_dp_get_items_subscribe() caller**
  - gets notified
  - attaches to **secondary SHM** with XPath and space for new SHM ID
  - gets the data from the client using callback
    - following sysrepo get rules (data are requested individually for all depths and parents)
  - merges them into one top-level subtree
  - stores in LYB in a new SHM segment and sets back this SHM ID

OPERATION SUBSCRIPTIONS

sr_(rpc, action, event_notif)_subscribe() caller

- gets notified
- attaches to **secondary SHM** with operation (input) and space for new SHM ID (only for RPC and action)
- calls the callback with input
- gets output from client
- copies into separate SHM in LYB and provides the ID

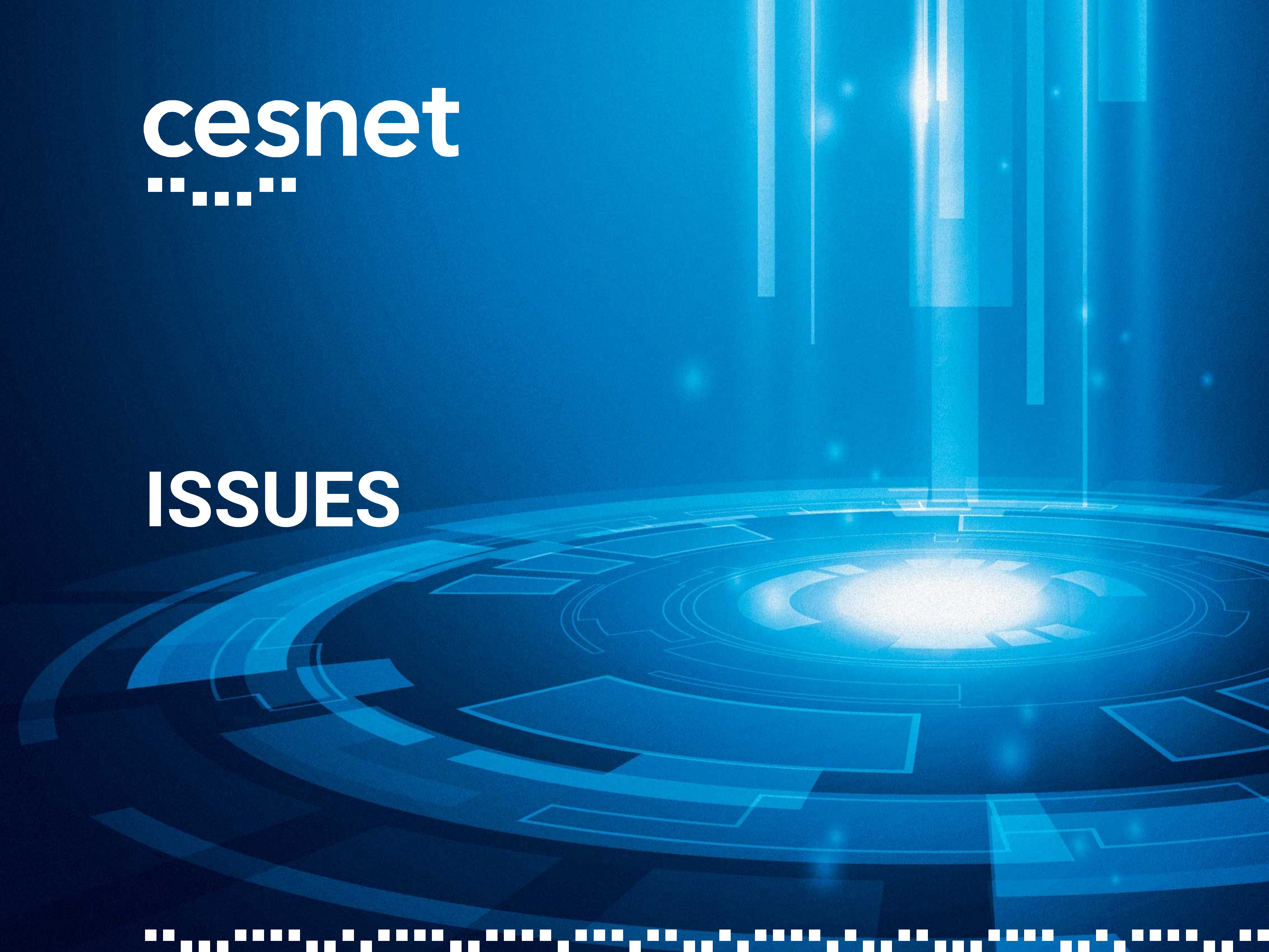
Operation sender

- provides input, gets output if any
- analogous to other scenarios

VALIDATION

Validation

- validation of one model data tree
- start with this data tree
- learn about any possible foreign dependencies (leafrefs, instids, must, when nodes) from **main SHM**
  - in the form of XPaths whose instances are looked for in the data tree
- if some found (are instantiated), merge also dependency data trees recursively (otherwise you do not need it)
- validate



ISSUES

DATA TREE LOCKING

Consistency

- many options depending on the behavior we require
- example commit situation
  - commit 1 starts and is calling callbacks
  - commit 2 starts, calls one callback, finishes and replaces the data tree
  - commit 1 finishes and replaces the data tree again
  - commit 2 changes are lost
- are we okay with this happening if no explicit locks are used?
  - the locks will be available and this situation avoidable if required
- should commit 2 be allowed to start while commit 1 is in progress?
- should commit 1 detect commit 2 changes and keep them?
  - **preferred solution**, keep all changes, when both commits modify same subtree, the second commit will fail
  - option to require explicit locks for commits on a model during its installation?

DATA TREE LOCKING

Partial locks

- allow to lock particular subtrees/individual nodes
- would allow simultaneous changes of independent data
  - meaning 2 commits changing independent data nodes/subtrees
- data dependencies
  - leafrefs, instance-identifiers, must, when, multiple cases, uniqueness (leaf-lists, lists, list unique), others?
  - all these dependencies, if not tracked, could cause unexpectedly invalid datastore or invalid operations (e.g. creating existing leaf-list)
  - tracking of these dependencies non-trivial (both implementation effort and efficiency)
- not worth it in my opinion
- current solution of locking models separately and not the whole datastore is enough

RUNNING DATA RETRIEVAL

- **Config data with operational data callback set**
  - sr_get_items() is being processed and these data are requested
- **operational data callback fails**
  - ignore and treat as no data present? (current sysrepo)
  - use the data from running DS?
  - return no data but inform about error?
    - **preferred solution**, we cannot send the error using NETCONF but can at least display it

VALUES VS SUBTREES

Currently, many API functions use 2 formats

- either subtrees (sr_node_t) or values (sr_val_t)
- in new sysrepo libyang data tree structures will internally be used everywhere
- proposed **API change** of all functions working with subtrees to use struct lyd_node * instead of sr_node_t
- structures include the same attributes
  - some are just not accessible directly as an attribute
  - e.g. to get module name, you must look into **node->schema->module->name**
- would become the most preferred (and most efficient) API variant
- transforming libyang nodes into current sysrepo subtree nodes would be non-trivial and inefficient

cesnet

THE END