HITACHI
Inspire the Next

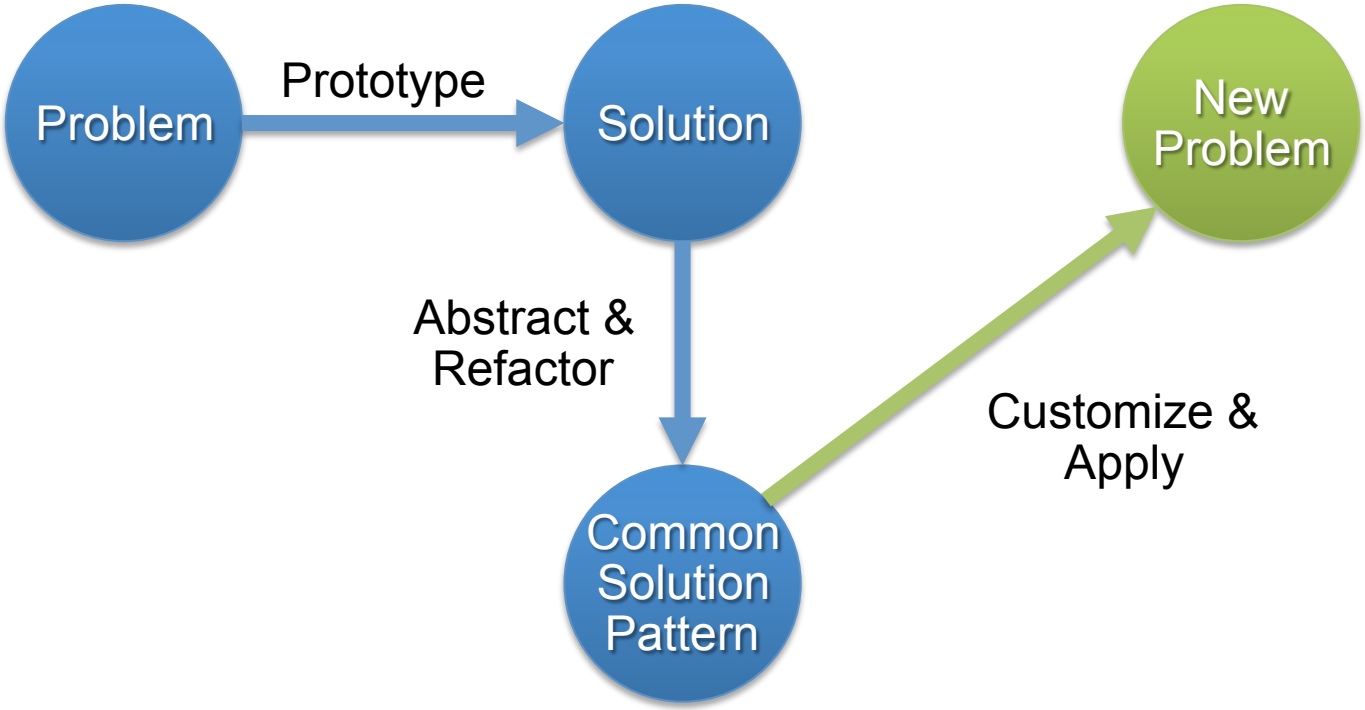
SUBFLOW Enhancements

Hiroyasu Nishiyama

Background

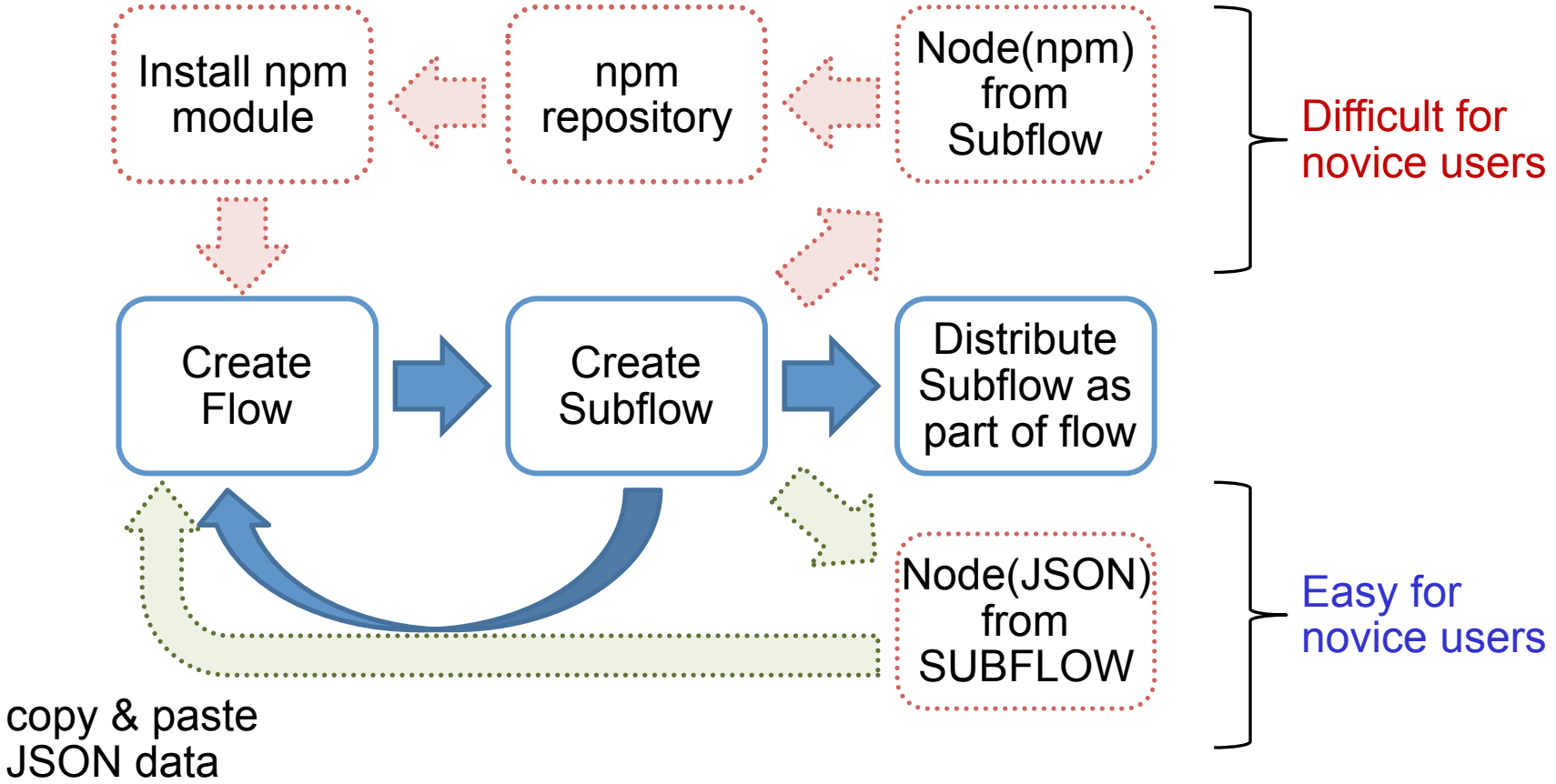
HITACHI
Inspire the Next

- Node-RED is a highly effective tool for rapid creation of new solutions.
- On the other hand, we would like to create basis for **sharing common solution patterns (or templates)** useful for creating new custom solutions by novice IT users.



Exporting Node in JSON from SUBFLOW

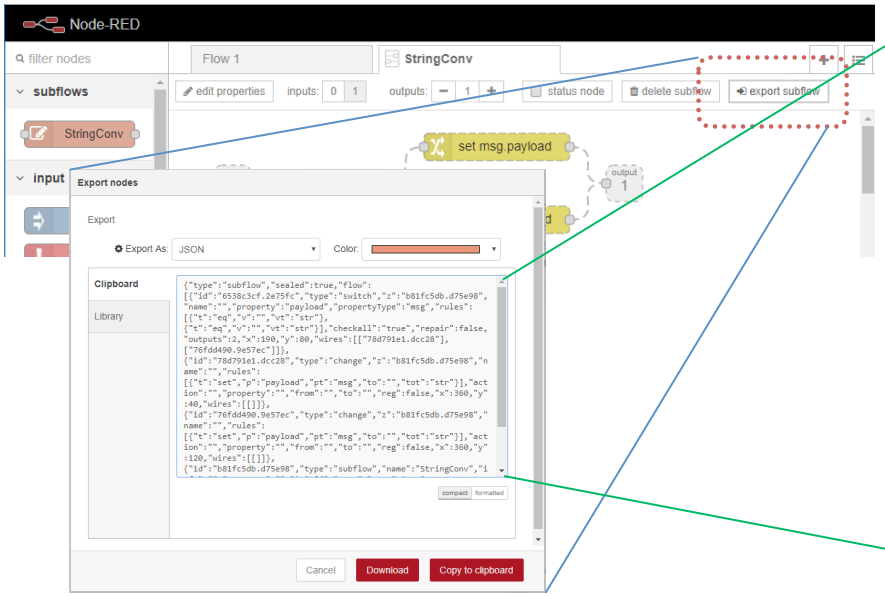
- Add new feature to export SUBFLOW as a node in JSON format.
- Node can be shared using JSON (text) format in addition to npm.
  → Can be redistributed as part of a flow(eliminates unknown nodes)
  → npm repository and explicit node installation is not needed.



Exporting SUBFLOW

HITACHI
Inspire the Next

- Add 「export subflow」 button to SUBFLOW template
- Use new FLOW format for distributing SUBFLOW:
  - type = "subflow"
  - sealed = true: hide details of imported SUBFLOW (do not allow to access SUBFLOW template)
  - flow: array of nodes exported as part of SUBFLOW

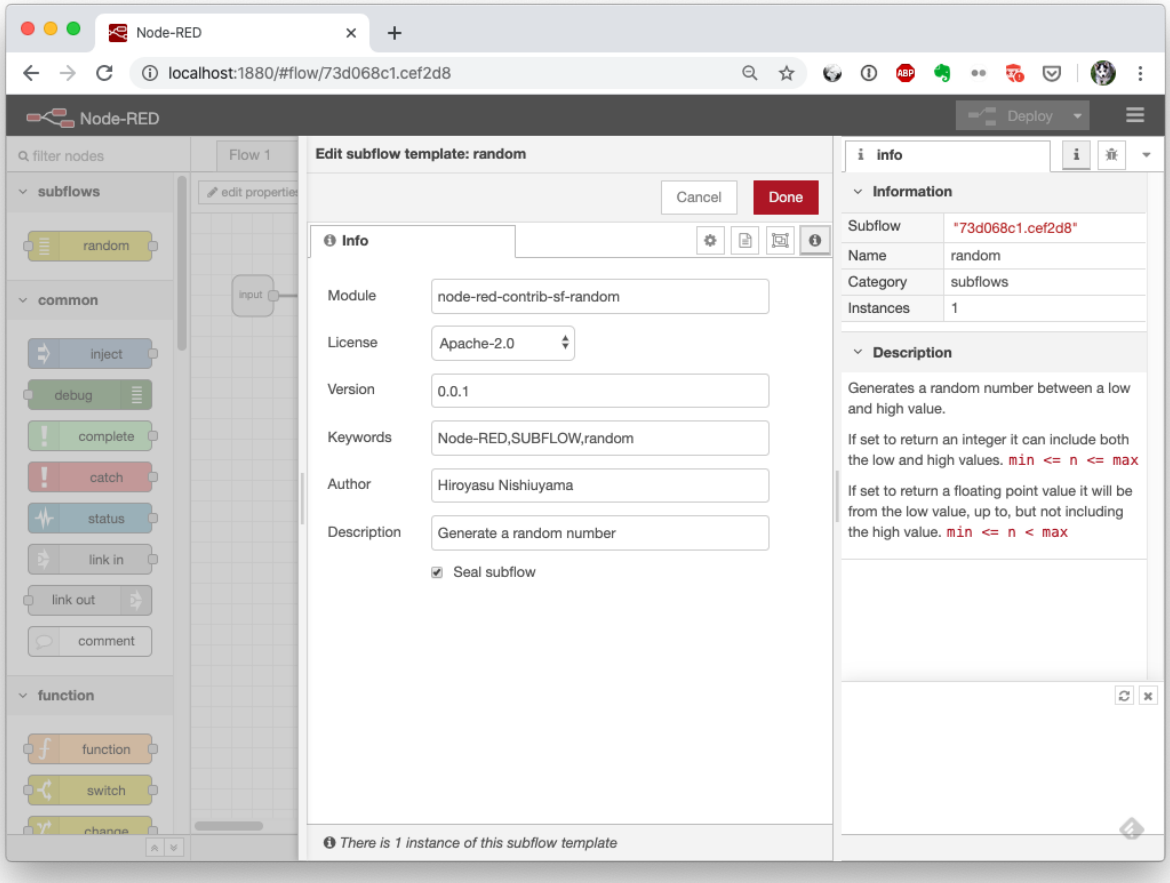



```
[{
    id: "<ID>",
    type: "subflow",
    ...
    sealed: true,
    flow: [
        { id: "...", z: "<ID>", type: "...", ... },
        ...
    ]
}]
```

Current Status of Exportable SUBFLOW

HITACHI
Inspire the Next

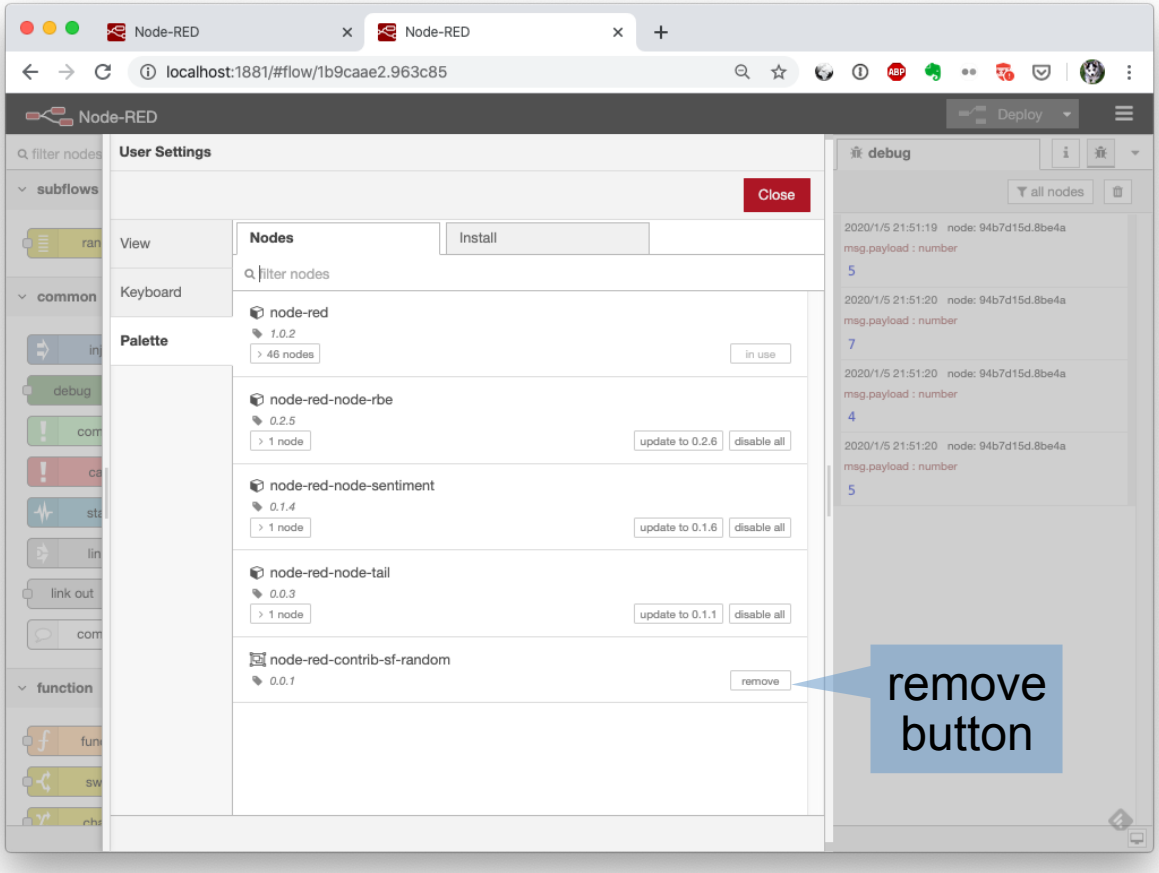
- Completed implementation of phase 1 to 3. Testing in progress.
- Algorithms, node description, settings UI, and meta-data definitions can be described in Node-RED editor and exported as JSON format.



Deleting Sealed SUBFLOW

HITACHI
Inspire the Next

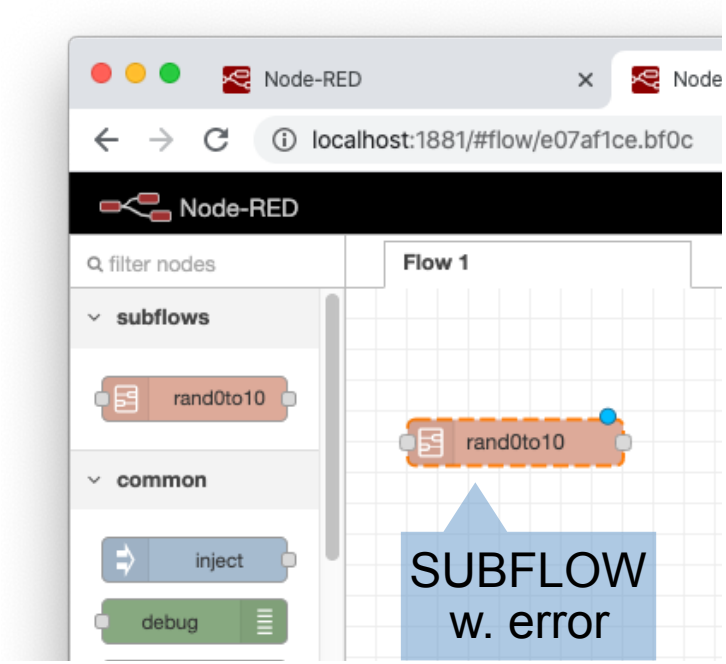
- When sealed SUBFLOW node is imported, SUBFLOW Template delete button can not be used.
- In order to allow deletion of sealed SUBFLOW, we show SUBFLOW node list on User Settings/Palette tab



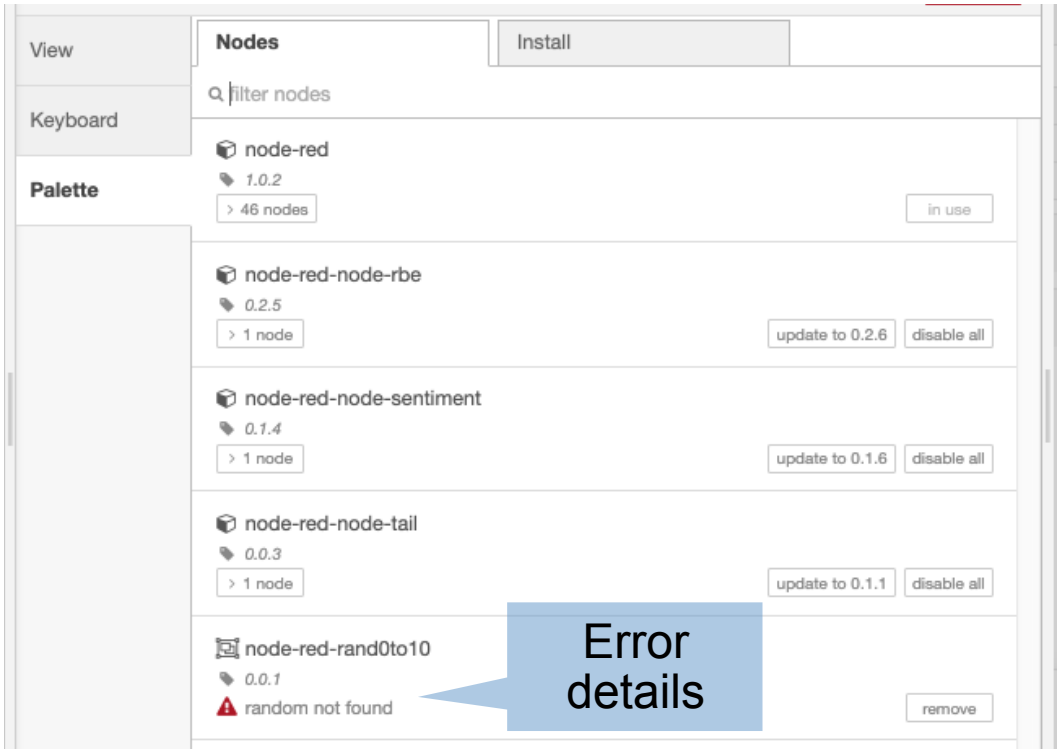
Handling Errors of Imported SUBFLOW node

HITACHI
Inspire the Next

- If there exist errors such as uninstalled node in imported SUBFLOW node, instance of the SUBFLOW is represented by dotted line
- Error details are displayed on node list of User Settings/Palette tab.



make new draft PR
on current implementation



HITACHI
Inspire the Next

DEMO

Possible Extension of SUBFLOW Features

HITACHI
Inspire the Next

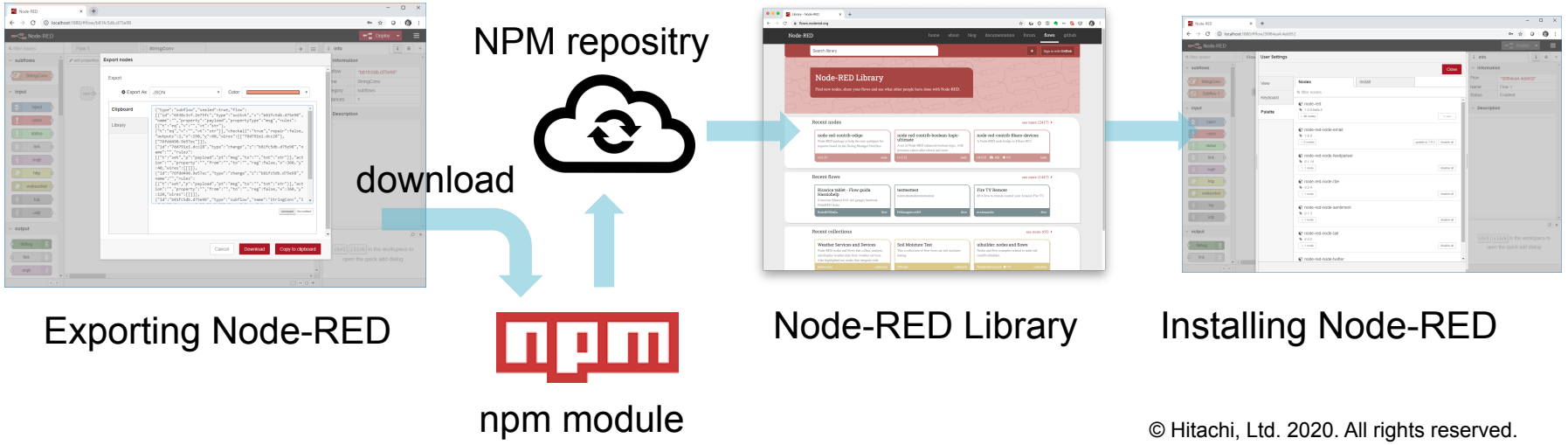
- In order to make exportable SUBFLOW feature more useful, we would like to propose following features:

1. **Exporting SUBFLOW node as NPM module,**
2. **Enhancement Function node,**
3. **Encryption of flow file,**
4. Addition of user-defined UI-type of SUBFLOW (discussed in Dashboard session)

[1] Exporting SUBFLOW as NPM Module

HITACHI
Inspire the Next

- Current node distribution uses NPM module as its format
- If we allow JSON based node (SUBFLOW) representation, redistributing SUBFLOW as NPM module may be useful:
  - automatic detection of node update,
  - embedding example flows,
  - listing in flow library by crawling npm repository,
  - ...
- Allow exporting SUBFLOW as NPM module, or command to create NPM module from SUBFLOW JSON data



[1] Node API for Installing SUBFLOW Node

HITACHI
Inspire the Next

- Current Node-RED API for installing nodes only accepts JavaScript/ HTML description of nodes.
- Two way to make SUBFLOW node as NPM module:
  1. **convert SUBFLOW to JavaScript/HTML code,**
  2. **make JSON flow definition of SUBFLOW installable from NPM module**
- Since method (1) needs complex flow conversion, we would like to propose method (2) with new API and node file format.

[JavsScript file]

```
module.exports = function (RED) {
    RED.nodes.registerSubflow([
        // SUBFLOW definition
    ]);
}
```

add metada to package.json that distinguish module type (JS or SF) and eliminate HTML for SF

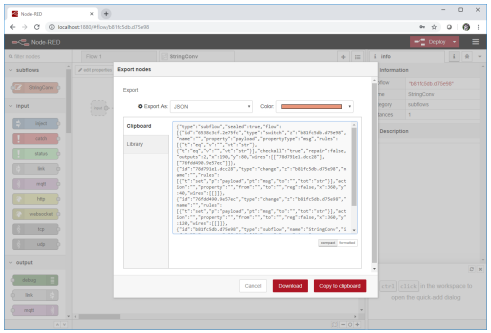
[HTML file(placeholder for marking package contents as SUBFLOW)]

```
<script type="text/x-red-subflow">
    // contents are ignored
</script>
```

[2] Converting SUBFLOW to NPM module

HITACHI
Inspire the Next

- Store SUBFLOW definition in a file with fixed name (e.g. subflow.js) which is load loaded from pre-defined JavaScript module, and generate meta-data, then create NPM module



Exporting Node-RED

SUBFLOW NPM Module

HITACHI
Inspire the Next

SUBFLOW NPM Module contains following contents

- <SUBFLOW>.json
  Exported SUBFLOW JSON definition
- <SUBFLOW>.js
  template code that loads SUBFLOW.json and call registerSubflow API
- README.md
  generated from SUBFLOW matadata (description property)
- package.json
  generated from SUBFLOW metadata

```
{
    "name": <exportName>,
    "version": <version>,
    "description": <description>,
    "dependencies": [],
    "keywrods": <keywords>,
    "node-red": {
        "subflows": {
          <name>: <SUBFLOW>.js
        }
    }
}
```

Empty/Create from nodes and require list of function nodes within subflow?

Q: should be consisted with function node enhancements?
-> same as NPM module

[2] Extension of Function node

HITACHI
Inspire the Next

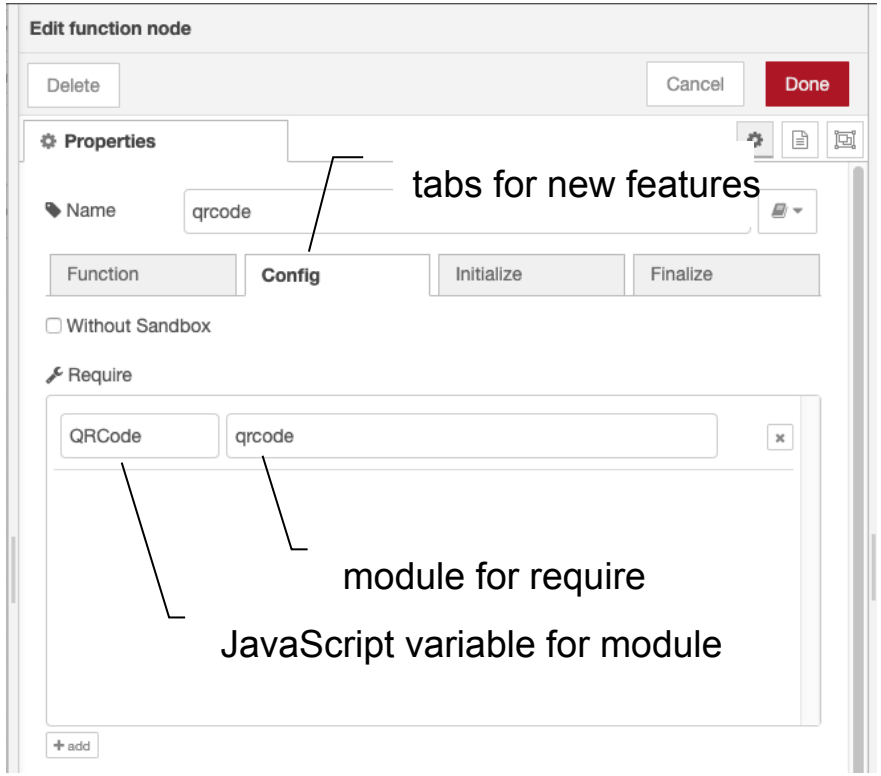
- When describing logic in SUBFLOW, function node plays a central role for expressing complex algorithms
- It has following problems:
  a. Can't use external libraries without modifying settings.js,
  b. Execution of function body is performed in VM environment (incur overhead),
  c. Function body is executed each time message is received.
     So, describing common initialization or shutdown code is difficult.

[2] New Settings Panel of Function Node

HITACHI
Inspire the Next

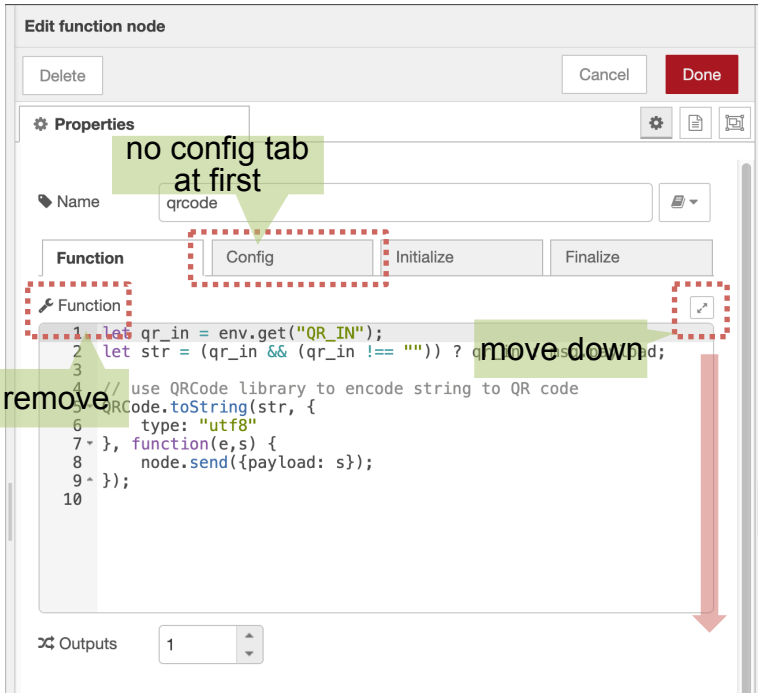
- Add tabs (Function/Config/Initialize/Finalize) to new features in Function node settings panel
  - **Function**: JavaScript code for function body
  - **Config**: Function node configuration (use of sandbox, module import, ...)
  - **Initialize/Finalize**: initialization and finalization code

tackle 3 items separately

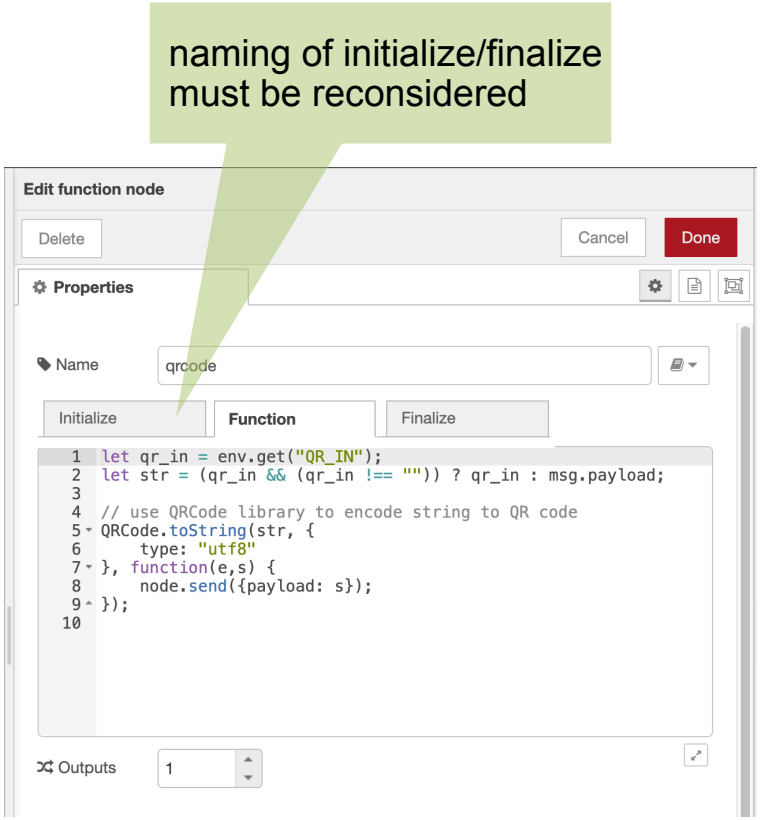


[N]Initialization/Finalization

HITACHI
Inspire the Next



Current Implementation



New Proposal

[N] Library Format

HITACHI
Inspire the Next

- Export format of function node to library currently uses comment to encode properties.
- This must be extended to be able to include init./fin. code and require list.

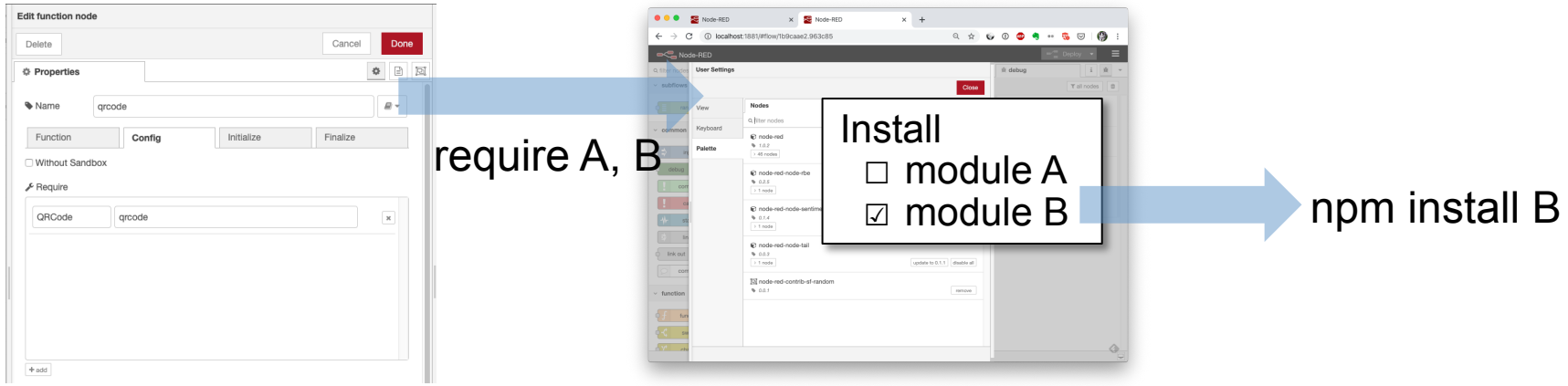
[2] Importing External NPM Module

HITACHI
Inspire the Next

- Current implementation expects required NPM modules are pre-installed
  → need a means to install NPM modules from Node-RED editor
- Solutions:
  1. runtime/Function node automatically install modules,
  2. add NPM install interface to Function node,
  3. add NPM install interface to editor settings
     a. runtime recognizes Function node configuration, or
     b. add API to register required modules from Function node

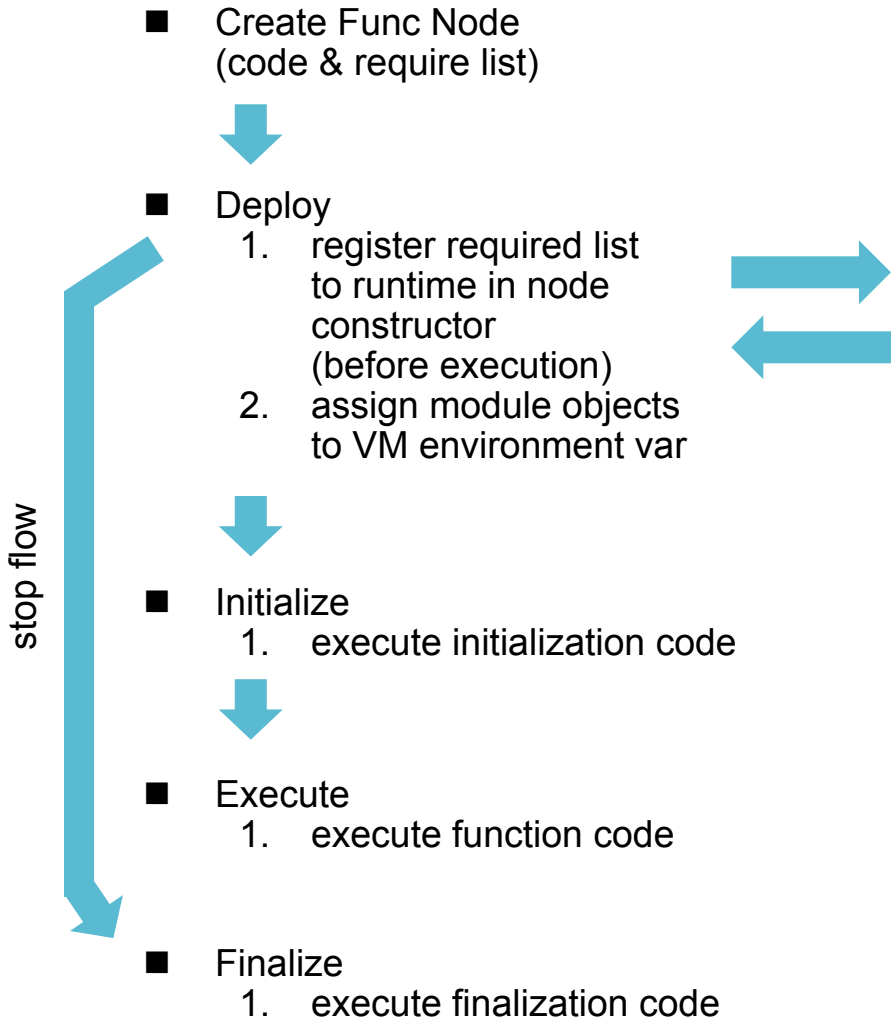
consider flow lifecycle
(e.g. RPI, container, etc.)
- security consideration
- some environment requir
installation of NPM module
- startup scenario

- We would like to suggest 3-b because the new API will be useful for other cases and it can manage which modules to be installed manually.



[N] Lifecycle Model of Function Node

HITACHI
Inspire the Next



- Create Func Node
  (code & require list)
- Deploy
  1. register required list to runtime in node constructor (before execution)
  2. assign module objects to VM environment var
- Initialize
  1. execute initialization code
- Execute
  1. execute function code
- Finalize
  1. execute finalization code

- New API (registerRequiredModules)
  1. get required list
  2. filter required list using black/white list to ensure only allowed modules are installed
  3. if auto-install-mode is on then automatically install required module
  4. return list of module objects. undefined if not loaded.

- Editor
  1. show list of npm modules in settings panel.
  2. npm modules can be installed/removed/updated from this panel

[N] Debug From Function Node

HITACHI
Inspire the Next

- Debug output is sometimes needed when creating function node code.
- console.log is not useful on our environment because we do not allow console access.
- node.warn or node.error can be used to output data to debug side-bar but its name seems to be not appropriate

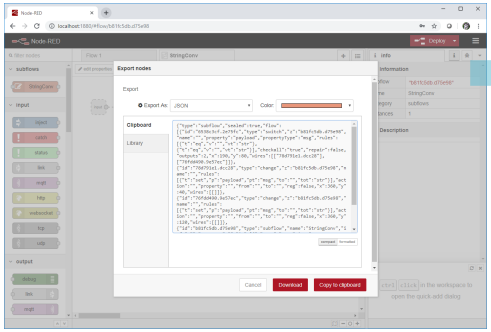
- Ideas
  - make node.debug or console.log output to debug sidebar.
    concern: current users may be affected by this change of behaviors
    -> Nick-san ask to users on this change

[3] Encrypting SUBFLOW

HITACHI
Inspire the Next

- In some cases, we want to hide details of SUBFLOWs because it may contain intellectual property

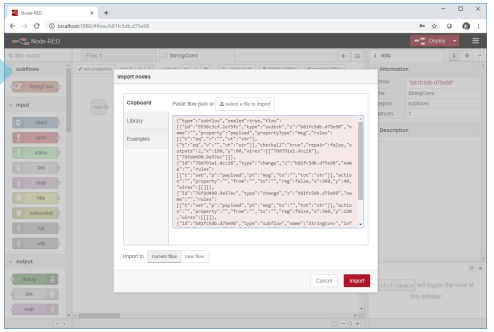
- By using new flow format, we can encrypt SUBFLOWs for distribution and decrypt it on installation.



Exporting Node-RED

```
{
    id: "<ID>",
    type: "subflow",
    sealed: true,
    flow: "W3siaWQiOiI2NTM..."
}
```

Encrypted FLOW format



Importing Node-RED

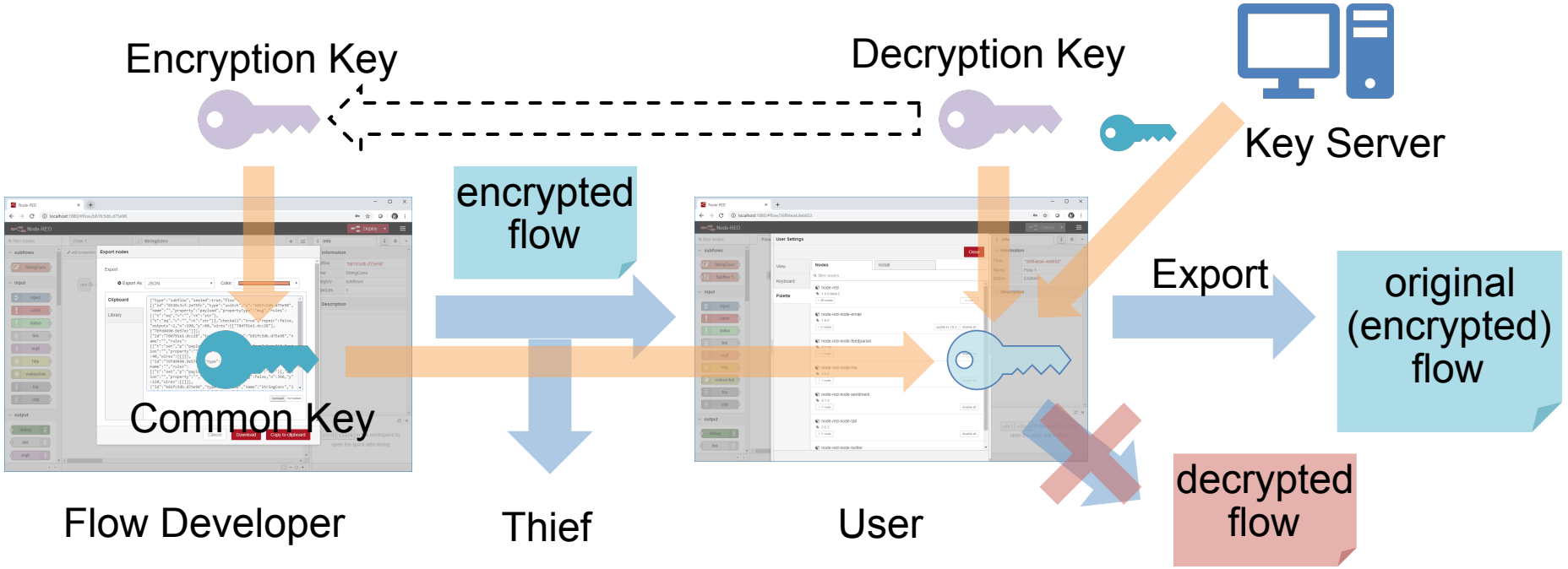
[3] Encryption Scenarios

HITACHI
Inspire the Next

- Expects good-willed user (users with authorization do not decrypt flow illegally).
- Prohibit decryption of a flow by users without authorization.
- Make encryption method selectable:

  a) Common Key, b) Key Server, c) Public-Key, ...

→ add hooks for encoding/decoding a flow to/from external format



[3] Settings for SUBFLOW Encryption

HITACHI
Inspire the Next

- Add a section for specifying hooks for encrypting SUBFLOW settings in settings.js

```
encryptSubflow: {
    encode: function (flow) {
        // code for encoding flow
    },
    decode: function (flow) {
        // code for decoding flow
    }
}
```

Create more detailed design and discuss with security professional

add default behavior
make multiple encoding types selectable

[N] Encryption Lifecycle

HITACHI
Inspire the Next

HITACHI
Inspire the Next

DEMO

Summary

HITACHI
Inspire the Next

- Proposed following features for improving usability of exportable SUBFLOW:

1. Exporting SUBFLOW node as NPM module,
2. Advanced mode of Function node,
3. Encryption of flow file,
4. Addition of user-defined UI-type of SUBFLOW
   (discussed in Dashboard session)

HITACHI

Inspire the Next