HITACHI
Inspire the Next

Experience in SPDK Contribution – SPDK iSCSI Resource Management and JSON Configuration File

03/23/2018

Shuhei Matsumoto

Scale-out Products Development Department,
IT Platform Products Management Division

Hitachi, Ltd.

HITACHI
Inspire the Next

Contents

HITACHI
Inspire the Next

1. iSCSI Resource Management

1-1 Improvement of iSCSI Resource Management – Motivation and Goal

HITACHI
Inspire the Next

Motivation

- iSCSI is still very important protocol for our business and product.
- SPDK iSCSI had a few issues:
  - Compliance of implementation to iSCSI specification was not so clear.
  - When using a new iSCSI target, all information must have been specified at its creation and could not be changed without removing it.
  - iSCSI is one of the oldest libraries in SPDK. Design and implementation of other libraries were more sophisticated. iSCSI had much room to improve.

Goal

- Check SPDK iSCSI implementation by referring iSCSI specification and figure out how SPDK iSCSI should be.
- Support dynamic reconfiguration of iSCSI resource.
- Refactor implementation of SPDK iSCSI.

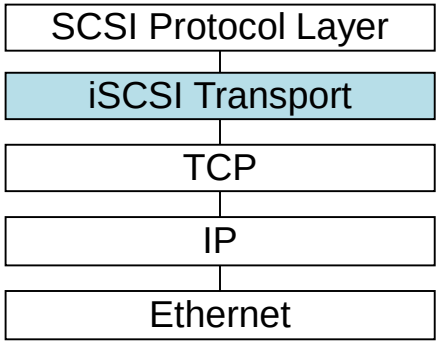
HITACHI
Inspire the Next

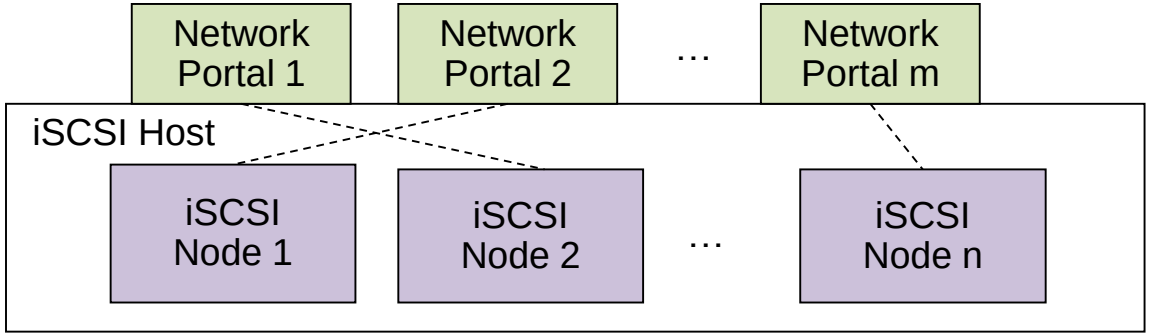
iSCSI Protocol

- iSCSI is a SCSI transport protocol that operates between the TCP/IP and the SCSI protocol layer.

iSCSI Node and Network Portal

- Each host have one or more iSCSI nodes, which is initiator or target.
- Each iSCSI node has one or more network portals for connectivity.
- Accessibility control of an iSCSI node to network portals is implemented.

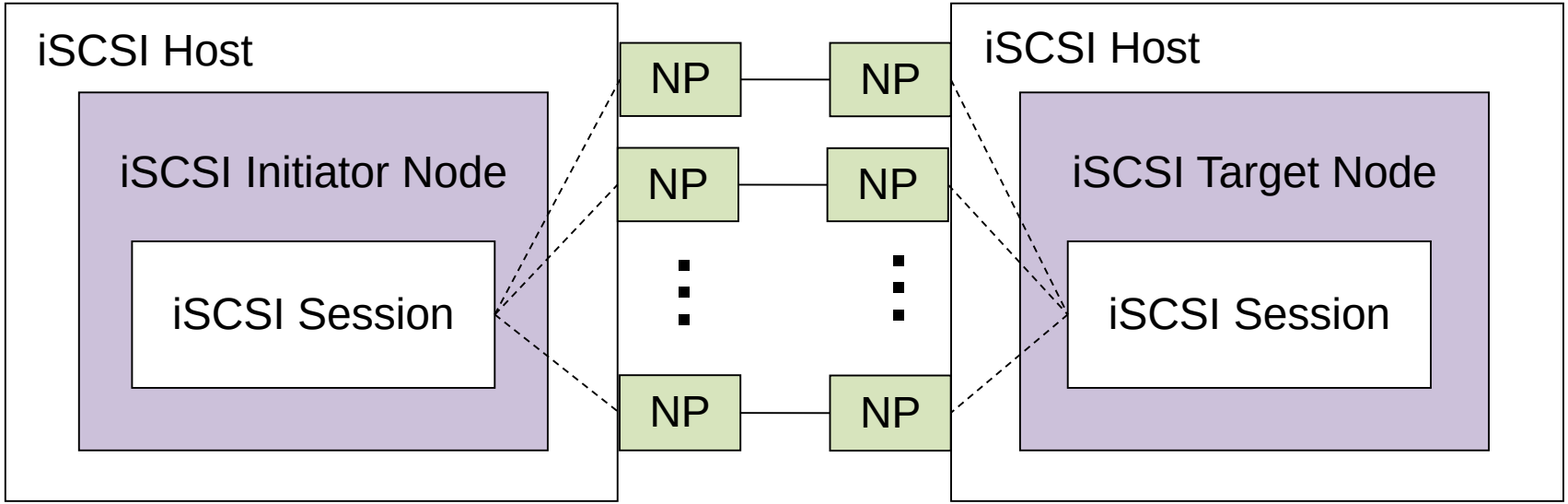




HITACHI
Inspire the Next

Session and Connectivity

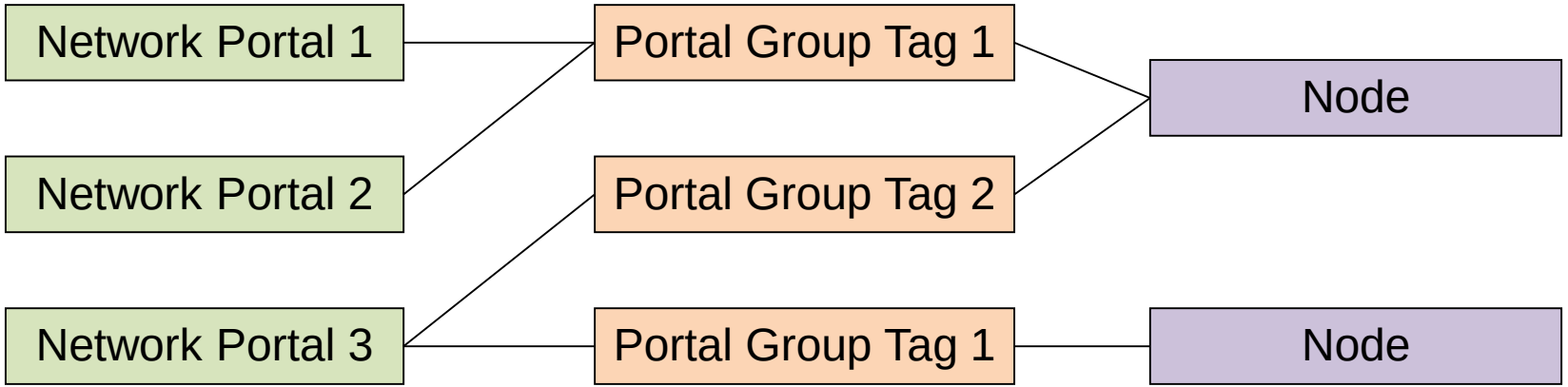
- An iSCSI session is the group of TCP connections that link an initiator node with a target node (equivalent to a SCSI I_T nexus).
- An iSCSI session may have one or more TCP connections (MC/S).



HITACHI
Inspire the Next

Portal Groups (PG)

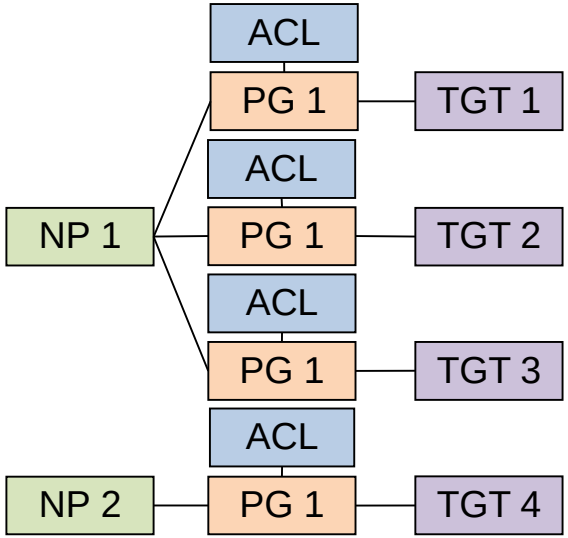
- iSCSI supports MC/S. MC/S can span over multiple network portals.
- These network portals can be grouped together into a PG.
- Each network portal belongs to exactly one PG within a node.
- PGs are identified by a PG tag, which is unique within a node.



1-3 iSCSI Target Topology Compliant with iSCSI Specification

HITACHI
Inspire the Next

- When allocating a network portal to a target node, create a PG in the target node first and then add the network portal to the PG.
- Accessibility control is done per session, i.e. per PG.



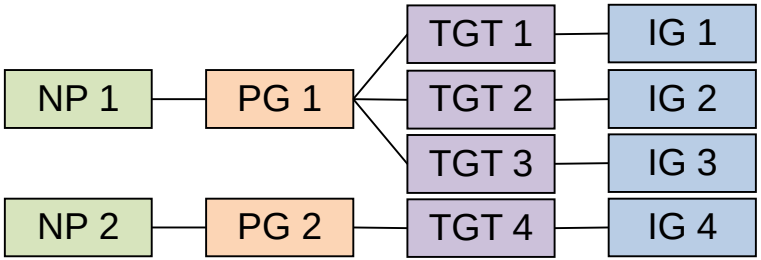
NP: Network Portal
ACL: Access Control List
PG: Portal Group
TGT: Target Node

- PG is necessary for MC/S. But SPDK iSCSI doesn't support MC/S
- When MC/S is not used, PG may not be meaningful concept.

1-4 iSCSI Target Topology in SPDK

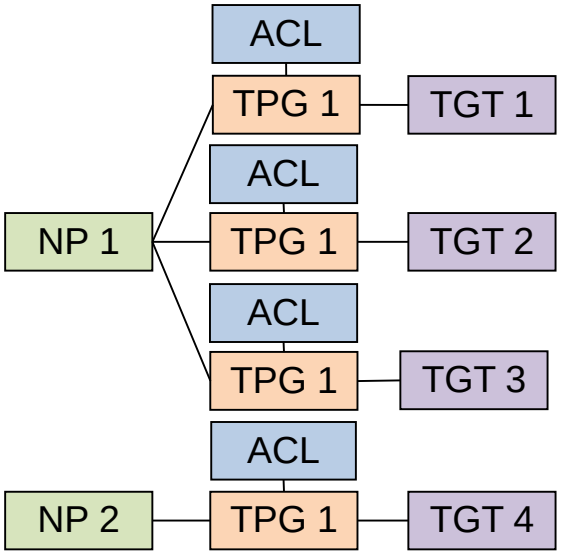
HITACHI
Inspire the Next

- The PG tag is globally unique in a host and is used as follows:
  - When creating each network portal, create a PG and then a PG-portal pair.
  - Use PG tag as the index of the network portal.
  - When allocating a network portal to a target node, specify the PG tag of the portal.
- Initiator group (IG) control accessibility. IG is based on the same idea as PG.
- The IG tag is globally unique in a host and is used as follows:
  - When creating each target node, create an IG and then a IG-target pair.
  - Use IG tag as the index of the target node.
  - When using a target, specify the IG tag of the target node.
  - When adding an initiator to a target node, the initiator is added to the IG of the target.

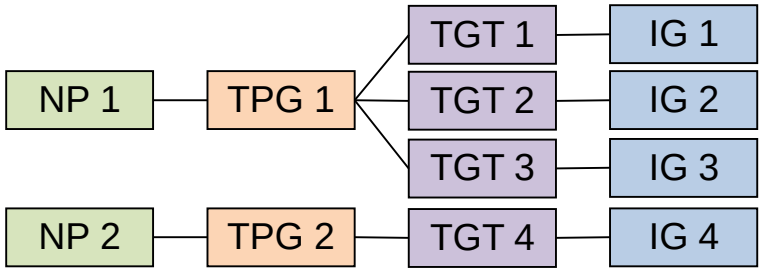


1-5 Comparison of iSCSI Target Topology

- SPDK iSCSI target is simple and can be used easily.
  - Use PG and IG tag as indexes of portals and target nodes, respectively.
- SPDK iSCSI have been validated enough for single path configuration.



iSCSI Target Topology Compliant with Specification



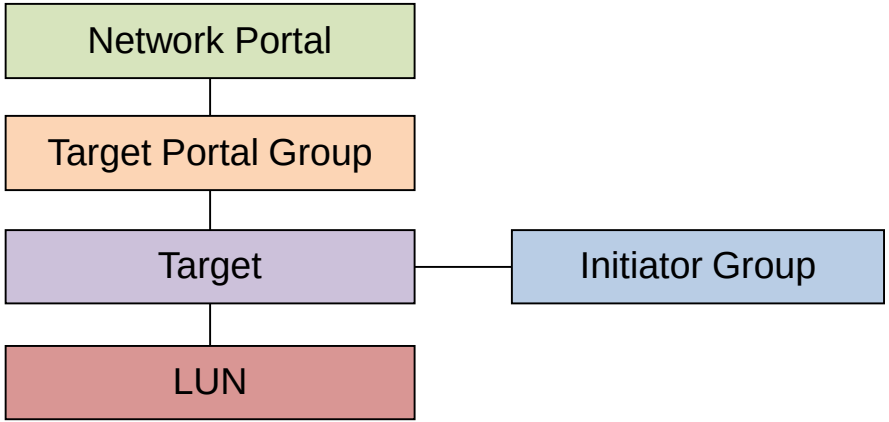
SPDK iSCSI Target Topology

1-6 Dynamic Reconfiguration of SPDK iSCSI Target

HITACHI
Inspire the Next

Now Components of iSCSI Target Can Be Changed Dynamically.

- LUN can be added to the iSCSI target. (Unit attention is not supported yet.)
- IP address of the port of the iSCSI target can be changed dynamically.
- Accessibility to the new host can be added to the iSCSI target.



HITACHI
Inspire the Next

2. JSON Configuration File

2-1 Current SPDK Configuration Management

HITACHI
Inspire the Next

Method to Manage SPDK Configuration

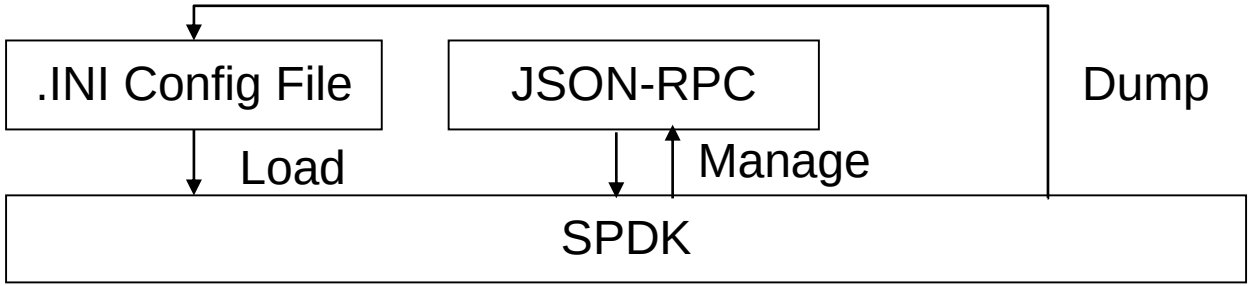
- SPDK provides two methods, .INI configuration file and JSON-RPC.

.INI Configuration File

- This is loaded and used at SPDK booting.
- This is used to initialize and configure SPDK subsystems.

JSON-RPC

- This is used to configure and manage SPDK subsystems.
- This is used mainly through Python based command line tool.



2-2 Issue of Current SPDK Configuration Management

HITACHI
Inspire the Next

Format Is Not Unified between API and Config File

- Users use SPDK mainly by JSON-RPC and Python tool. Unification to JSON will be valuable for many.

Current Config File Is Not Easy to Export/Edit/Import

- Exporting, editing, and importing config file at runtime are very usual operation.
- Current config file supports dump function but it is not easy to use.

Current JSON-RPC Cannot Control Initialization of Subsystems

- Most operations can be done by JSON-RPC but initialization of subsystems can be done only by .INI config file.

Current Config File Is Not Aware of Persistent Metadata.

- If metadata is persistent, the owner of it should be skipped at the next reboot.

2-3 Motivation to JSON Config File

HITACHI
Inspire the Next

Usual Operation Can Be Done by JSON Config File

1. Export the JSON config of the components you want to edit.
2. Edit the JSON config.
3. Import the JSON config.

JSON-RPC and Config File Can Be Used Depending on the Situation

- JSON-RPC can modify the attributes of the running components.
- JSON config file can be saved and used to be affected at the next reboot.
- Users will be able to get better flexibility by this combination.

Verification of JSON Config File

- JSON config file is easy to verify before loading.

Easy Deployment of SPDK Application Configuration

- JSON config file will be helpful to deploy SPDK application easily.

2-4 High-Level Design of SPDK JSON Config File

HITACHI
Inspire the Next

Format of JSON Config File

- JSON config file is made of a sequence of JSON-RPC requests.
- Sequence of JSON-RPC requests is ordered by dependency.

Addition of New JSON-RPCs

- Add new JSON-RPCs to initialize SPDK subsystems. (Configuring SPDK subsystems can be done by existing JSON-RPCs.)
- Add new JSON-RPCs to dump configuration of SPDK subsystems. Dump is made of a sequence of JSON-RPCs to restore the configuration.

(Continued …)

2-4 High-Level Design of SPDK JSON Config File (Continued)

HITACHI
Inspire the Next

Disable Auto-Start of SPDK Booting

- Add a command line option to disable auto-start of SPDK booting.
- When SPDK receives the option, it waits for JSON-RPCs to initialize and configure SPDK subsystems and then start SPDK application.

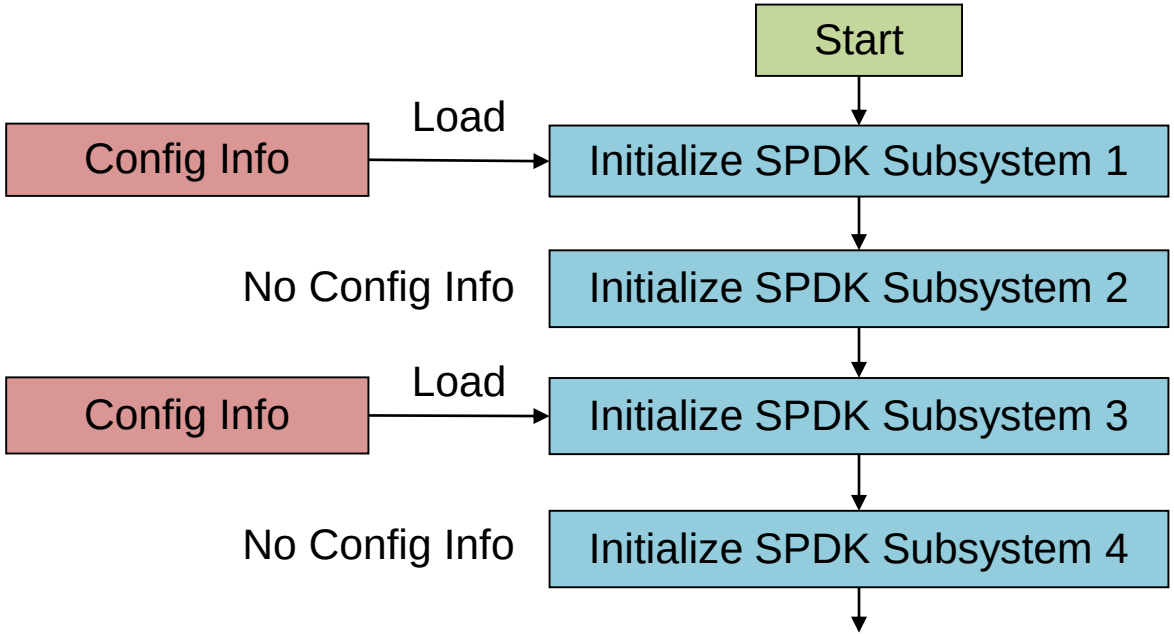
Python Client Controls Progress of SPDK Booting

- Python client iterates JSON config file and sends JSON-RPCs synchronously and sequentially.
- Python client notifies the end of JSON config file by JSON-RPC.
- When SPDK receives the notification, SPDK starts the application.

2-5 Comparison of the Approach to SPDK Boot Flow – .INI Config File

HITACHI
Inspire the Next

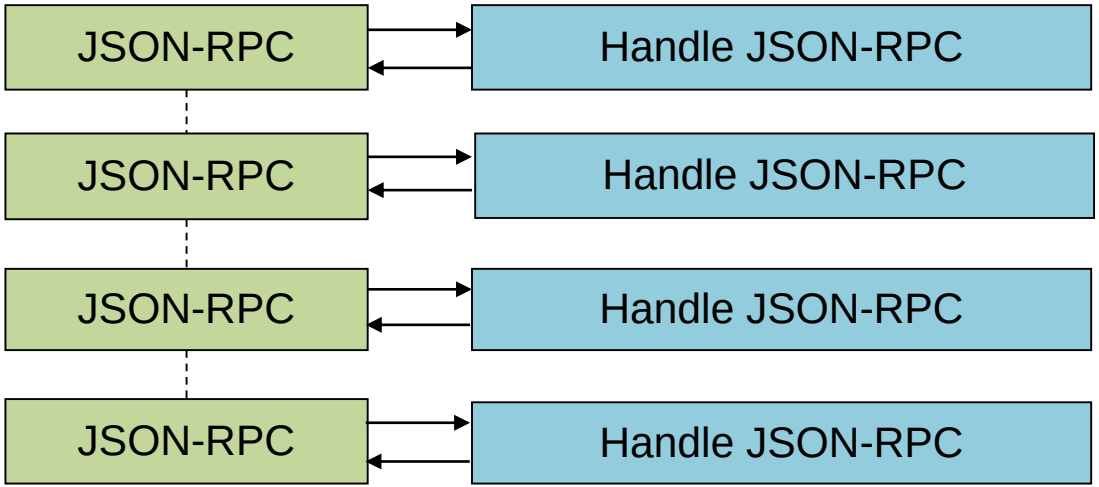
- SPDK proceeds subsystem initialization actively by itself.
- SPDK knows dependency relationship among subsystems.
- Whether config info is loaded or not depends on each subsystem.



2-5 Comparison of the Approach to SPDK Boot Flow – JSON-RPC

HITACHI
Inspire the Next

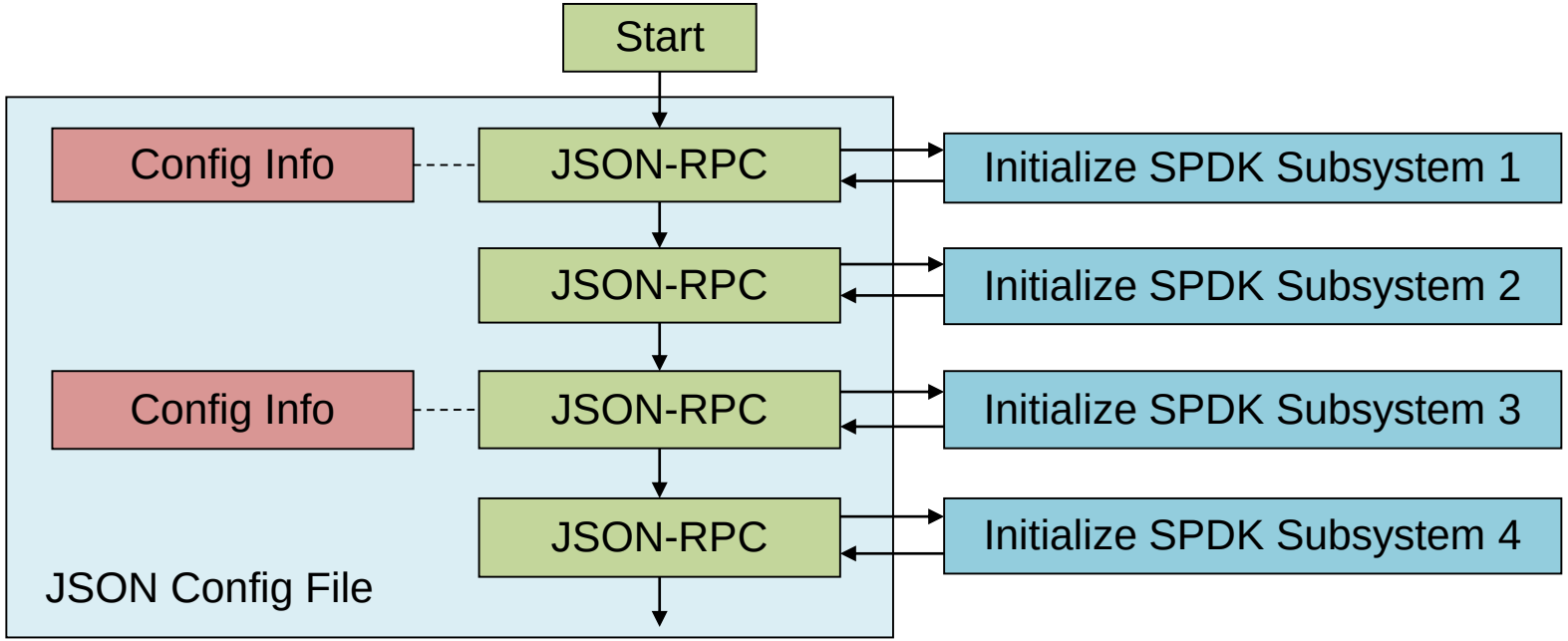
- Many users operate SPDK through JSON-RPC.
- JSON-RPC is used mainly not directly but through Python client.
- Most operations can be done through JSON-RPC.
- JSON-RPC is synchronous and unidirectional (Python client -> SPDK).
- Sequence of JSON-RPCs proceeds passively for SPDK.



2-5 Comparison of the Approach to SPDK Boot Flow – JSON Config File

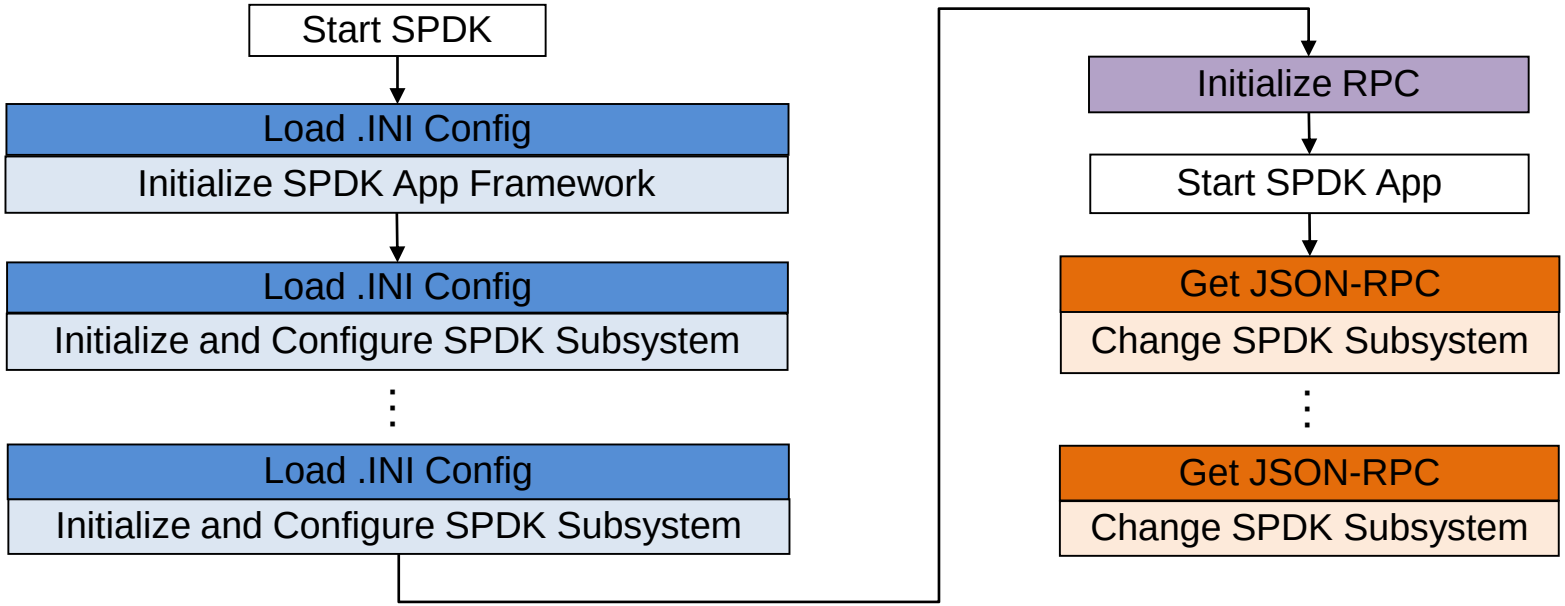
HITACHI
Inspire the Next

- Subsystem initialization is done passively by JSON-RPC.
- Python client iterates JSON-RPCs one-by-one synchronously



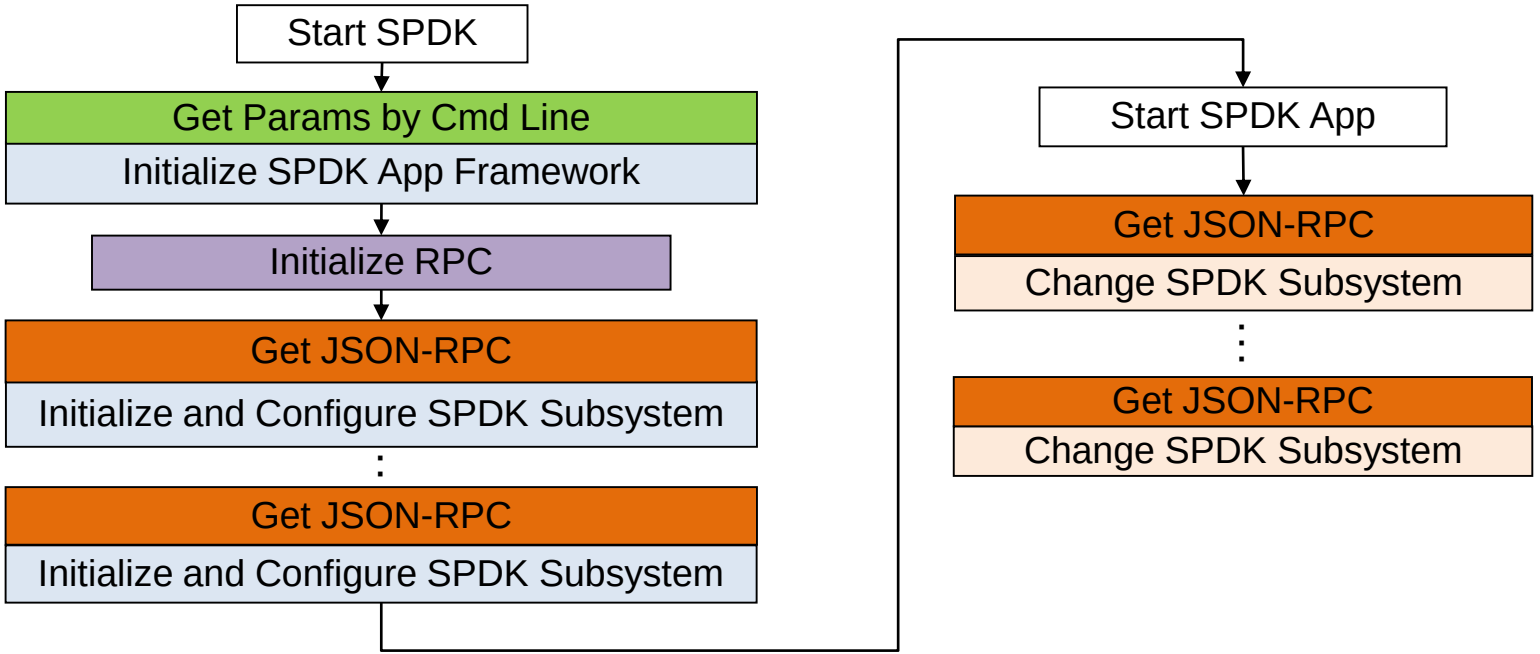
- Single step execution or breakpoint function will be available.

2-6 Comparison of the SPDK Boot Flow – .INI Config File



- Initialization of SPDK App Framework needs .INI Config File.
- JSON-RPC must be initialized after SPDK App Framework.
- JSON-RPC must start after completion of SPDK subsystem initialization to avoid corruption due to conflict between .INI config file and JSON-RPC.

2-6 Comparison of the SPDK Boot Flow – JSON Config File



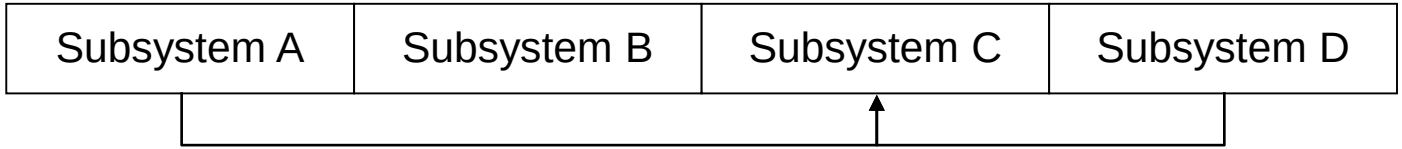
- Initialize JSON-RPC before starting SPDK subsystem initialization.
- Subsystem initialization proceeds by JSON-RPC driven.
- SPDK allows only JSON-RPCs for subsystem init before starting SPDK App.

2-7 Dependencies of among Subsystems and Bdevs

HITACHI
Inspire the Next

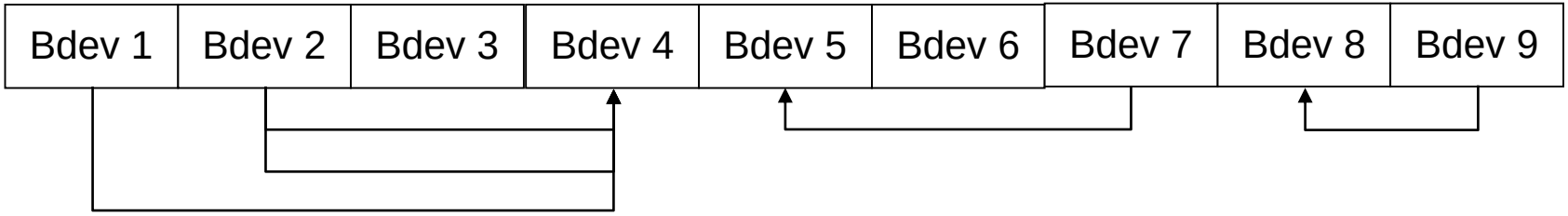
SPDK Subsystems have dependencies among them.

- List of JSON-RPC requests in JSON config file is ordered by dependencies.



Bdevs have dependencies among them and persistent metadata.

- List of JSON-RPC requests in JSON config file is ordered by dependencies.
- Some bdevs have persistent metadata. Supplement information is provided to handle JSON config file appropriately.



HITACHI
Inspire the Next

3. Experience in SPDK Contribution

3-1 Why I Can Do in the SPDK Community Now

HITACHI
Inspire the Next

Why SPDK was the first candidate in our team

- We wanted open source, user space implementation, and non-GPL iSCSI.
- We already started using SPDK.
- SPDK was clean and not monolithic library.
- SPDK might bring us to the latest technology.

Why I Jumped into the SPDK Community

- First I was not so positive for using SPDK.
  - SPDK iSCSI was not matured yet and there were more matured ones.
  - We had to customize SPDK iSCSI but we had not contributed any open source yet.

(Continued …)

3-1 Why I Can Do in the SPDK Community Now (Continued)

HITACHI
Inspire the Next

Why I Jumped into the SPDK Community.

- After consideration and discussion, I jumped into the SPDK community.
  - SPDK looked innovative and exciting.
  - I believed open source would be one of the ways to go for Hitachi.
  - If I do this, it would be OK for us. This was what I had wanted to do.

Why I Can Continue Contribution So Far.

- SPDK community is very supportive and we have good relationship with them.
- I'm not smart enough but may have provided a little complementary capability to SPDK community.
- My contribution to SPDK community will be connected to us directly.

3-2 Our Challenge to SPDK

HITACHI
Inspire the Next

Spreading SPDK Internally is still a challenging task

- Asynchronous programming is very new for us.
- SPDK is counter-intuitive to our previous thought that multi-core and -queue are essential for performance. SPDK utilizes CPU core much efficiently.
- To utilize SPDK well, understanding SPDK well is necessary.
- Some may require that SPDK is included in the OS package.
- Way of thinking is different between open source community and our team.

Working in the SPDK community is good for us

- Our utilization of SPDK is going forward.
- We can work side-by-side with smart engineers and the latest technologies.
- My contribution is spreading among us slowly but steadily.

(Continued …)

3-2 Our Challenge to SPDK (Continued)

HITACHI
Inspire the Next

Our decision of SPDK contribution is absolutely right.

- We will continue contribution to SPDK together with community.
- SPDK is still difficult but we will be able to use SPDK well like our software.

Let’s Join SPDK Community.

- We will not be able to use SPDK well alone.
- I hope SPDK will evolve as open source community and I believe it is valuable for every person, group, and organization that use SPDK.

HITACHI
Inspire the Next

END

Experience in SPDK Contribution – SPDK iSCSI Resource Management and JSON Configuration File

03/23/2018

Shuhei Matsumoto

Scale-out Products Development Department,
IT Platform Products Management Division,

Hitachi, Ltd.

HITACHI

Inspire the Next