STORAGE DEVELOPER CONFERENCE
SDC 22
Fremont, CA
September 12-15, 2022
BY Developers FOR Developers

A SNIA Event

Redfish Adoption at Google

Redfish development and modelling for Google's data center management

Presented by Derek Chan (dchanman@google.com)

Contributions from Mingyang Sun (smy@google.com)

Main challenges for Redfish adoption

Software Development

- Simplify the cross-org client adoption of Redfish
- Track Redfish usage across the organization

Modelling

- Normalizing telemetry to legacy telemetry schemes (e.g. ID schemes)
- Upstreaming and generalizing the gaps in upcoming usecases

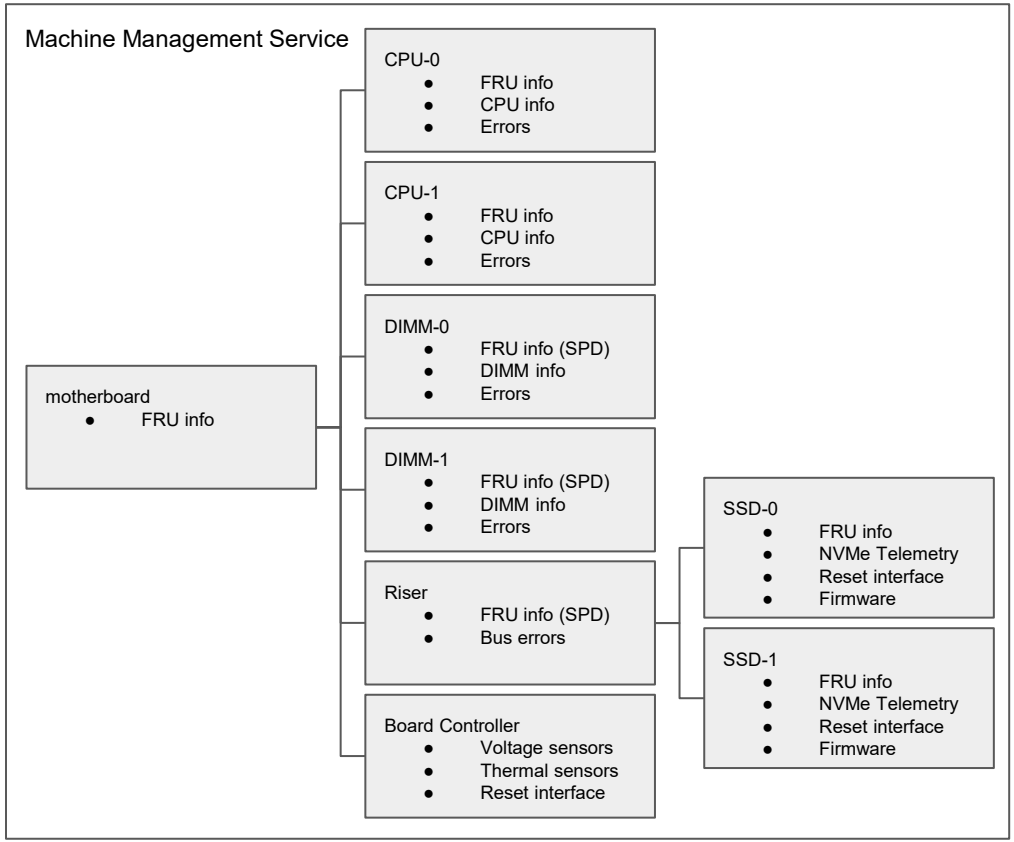
Background

Migrating onto Redfish

Background

Google has been using a proprietary management service prior to Redfish.

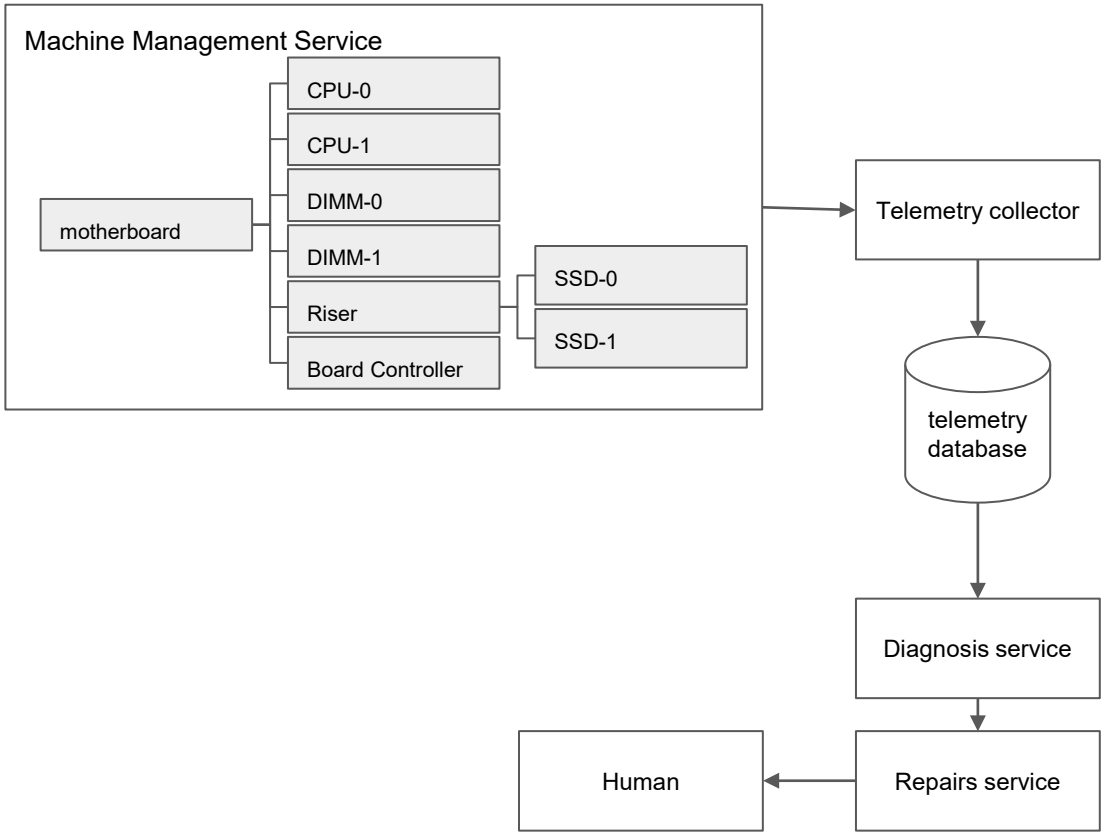
The service roughly implements an Entity-Component design pattern to represent hardware and their control/telemetry capabilities in a system model.



Background

Google depends heavily on application services using data from this model to automate the machine lifecycle.

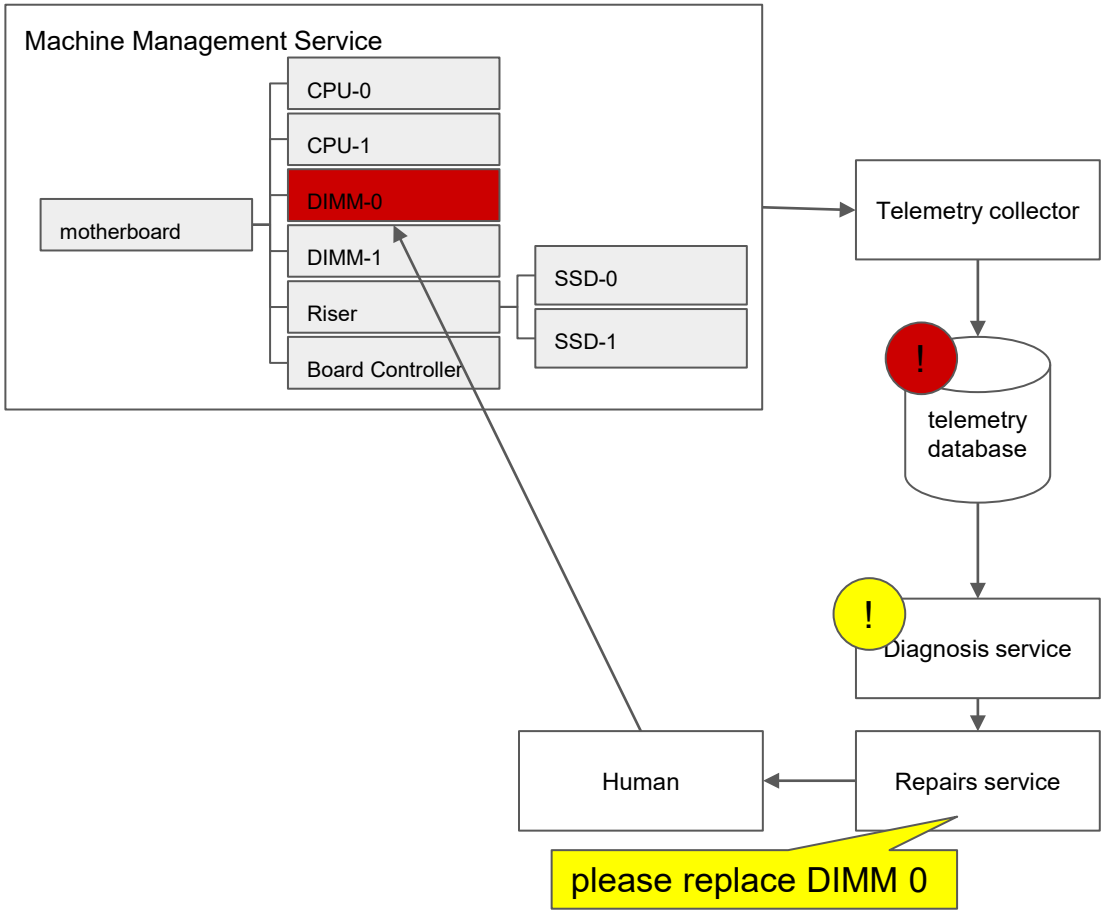
e.g. Automation will diagnose machine issues, attempt restart actions, and only dispatch a human to perform specific directed actions.



Background

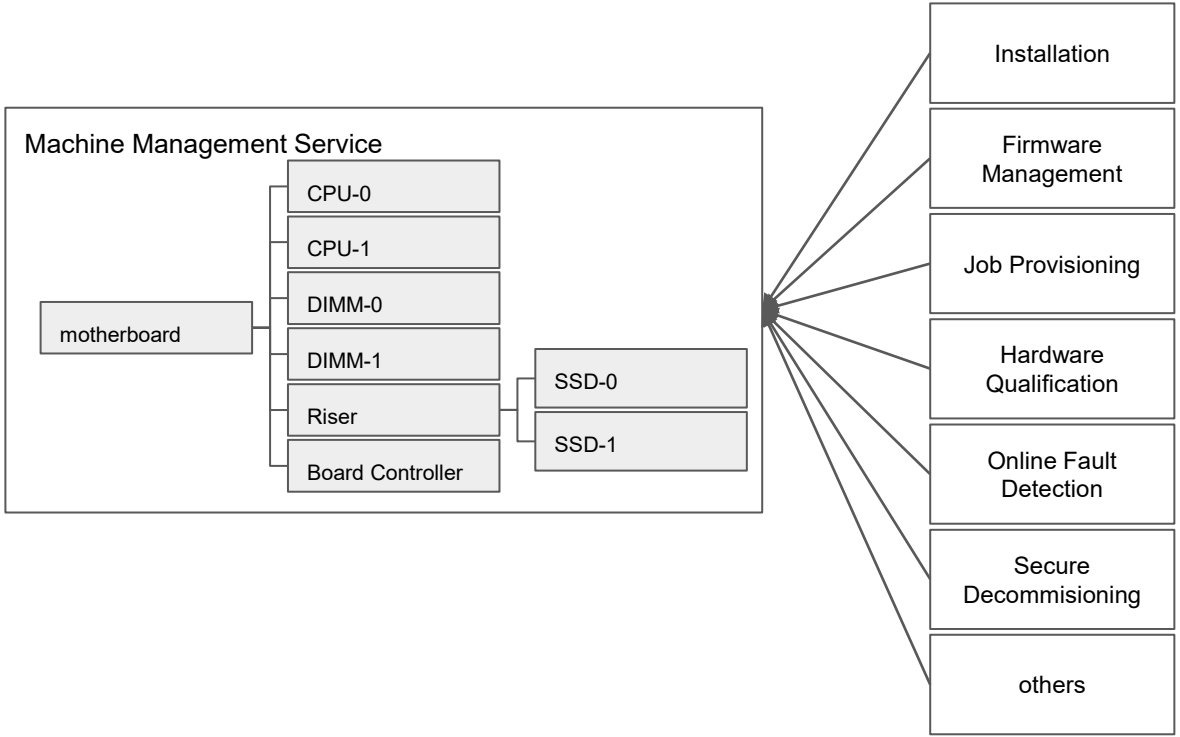
Google depends heavily on application services using data from this model to automate the machine lifecycle as much as possible.

e.g. Automation will diagnose machine issues, attempt restart actions, and only dispatch a human to perform specific directed actions.



Background

Aside from fault detection and repairs, many other automated services depend on the same Machine Management Service.



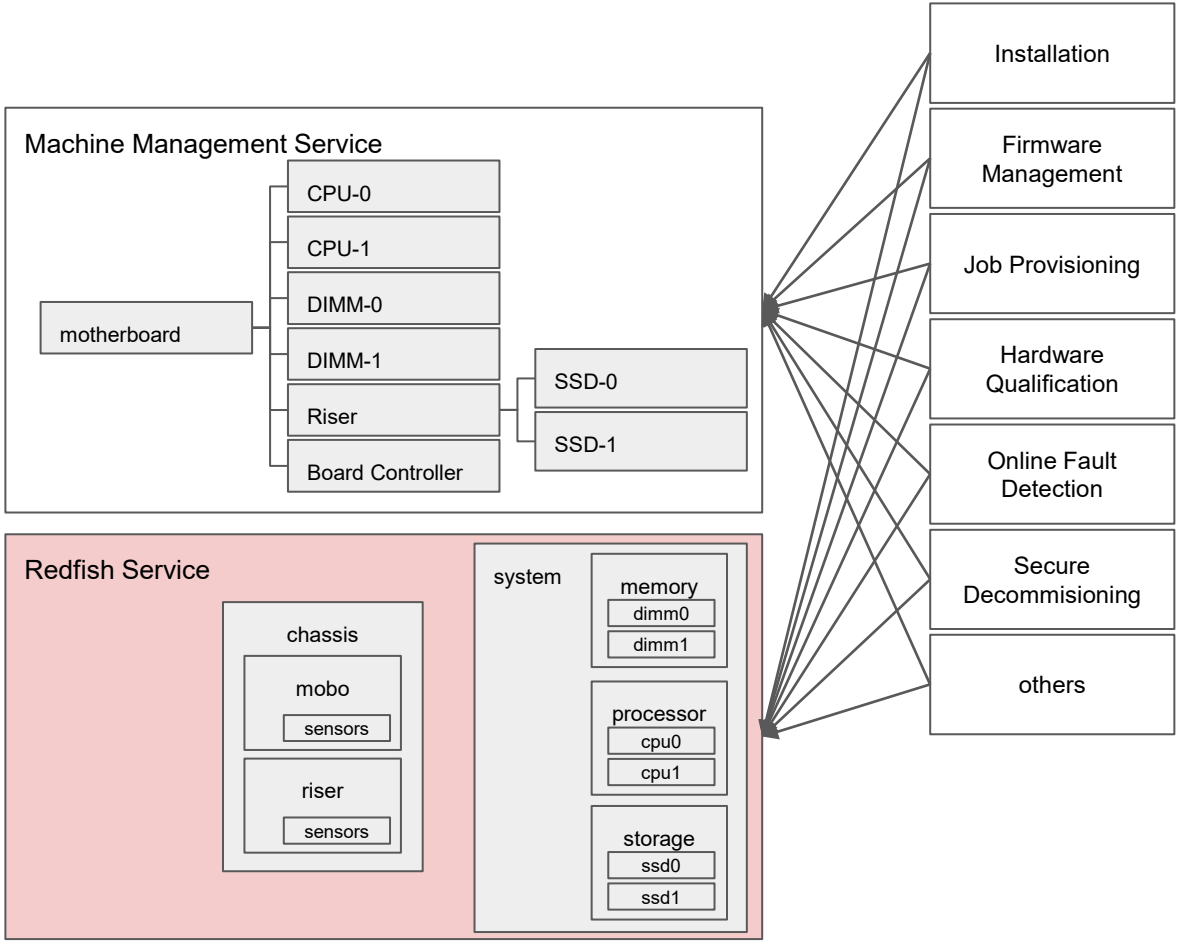
Challenges

Software engineering challenges:

- business viability depends on maximizing automation over human processes
- ownership of these client systems fall under different organizations within Google

Modelling challenges:

- clients need backwards compatibility with the legacy service
- modelling requirements are based on what the legacy service could do, not what Redfish could do



Redfish usage

Client development perspective

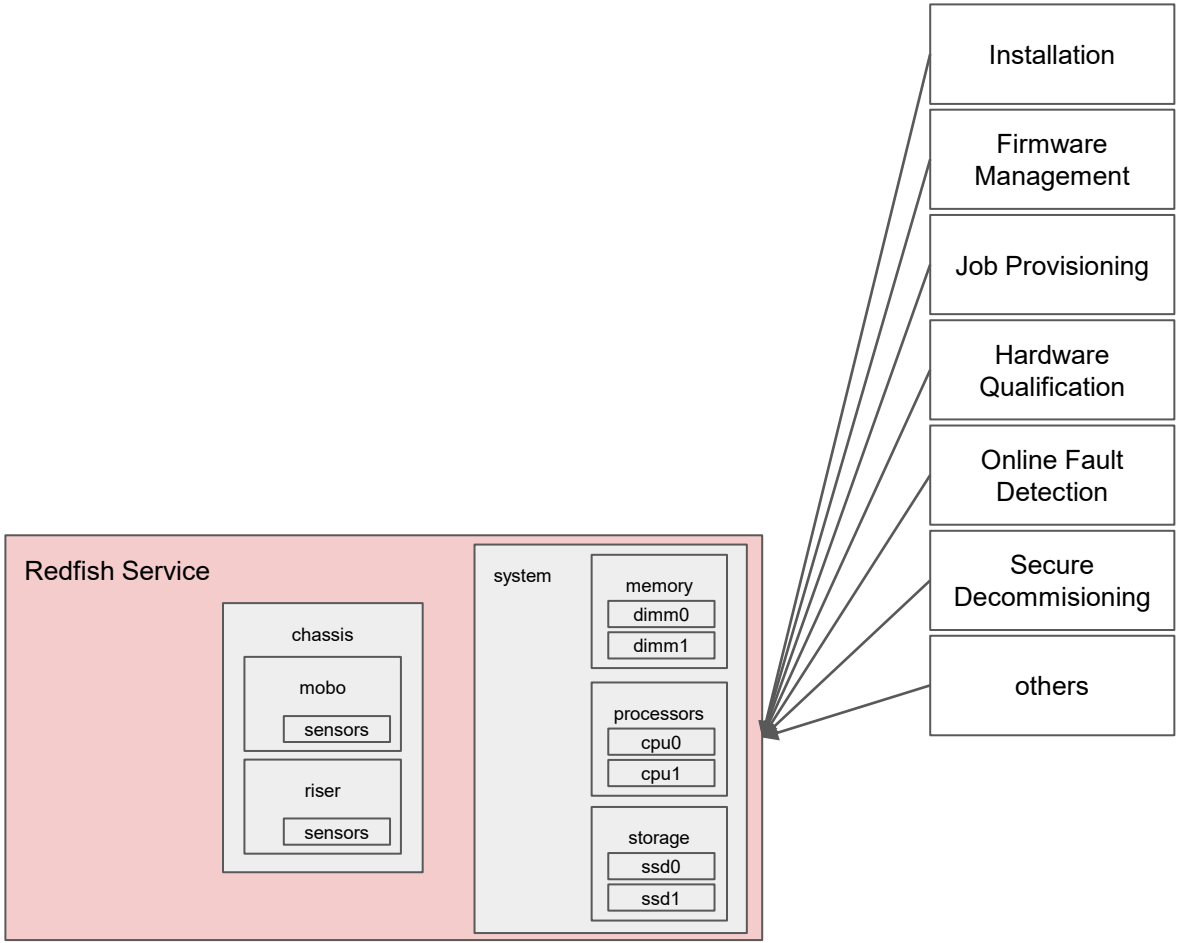
What do we have today at Google?

- Imperative C++ library
- Started with DMTF/libredfish, our C++ library borrows many concepts

```
void GetAllPartInfo(RedfishInterface *intf, T callback) {
    intf    - >GetRoot()[“Chassis”].Each().Do([](auto resource) {
        callback(resource[“PartNumber”], resource[“Manufacturer”]);
    });

    intf    - >GetRoot()[“Systems”].Each()[“Processors”].Each().Do([](auto resource) {
        callback(resource[“PartNumber”], resource[“Manufacturer”]);
    });

    intf    - >GetRoot()[“Systems”].Each()[“Memory”].Each().Do([](auto resource) {
        callback(resource[“PartNumber”], resource[“Manufacturer”]);
    });

    intf    - >GetRoot()[“Systems”].Each()[“Storage”].Each()[“Drives”].Each().Do([](auto resource) {
        callback(resource[“PartNumber”], resource[“Manufacturer”]);
    });
}
```

Anticipating scaling problems

An imperative C++ library doesn't scale when onboarding many cross-org teams



Anticipating s problems

```
void GetAllPartInfo(RedfishInterface *intf, T callback) {
    intf    - >GetRoot()[“Chassis”].Each().Do([](auto resource) {
        callback(resource[“PartNumber”], resource[“Manufacturer”]);
    });

    intf    - >GetRoot()[“Systems”].Each()[“Processors”].Each().Do([](auto resource) {
        callback(resource[“PartNumber”], resource[“Manufacturer”]);
    });

    intf    - >GetRoot()[“Systems”].Each()[“Memory”].Each().Do([](auto res
        callback(resource[“PartNumber”], resource[“Manufacturer”]);
    });

    intf    - >GetRoot()[“Systems”].Each()[“Storage”].Each()[“Drives”].Each().Do([](auto resource) {
        callback(resource[“PartNumber”], resource[“Manufacturer”]);
    });
}
```

golang?

python?

```
void GetDrivePartInfo(RedfishInterface *intf, T callback) {
    intf    - >GetRoot()[“Chassis”].Each().Do([](auto resource) {
        callback(resource[“PartNumber”], resource[“Manufacturer”]);
    });

    intf    - >GetRoot()[“Systems”].Each()[“Storage”].Each()[“Drives”].Each().Do([](auto resource) {
        callback(resource[“PartNumber”], resource[“Manufacturer”]);
    });
}
```

```
void GetCpuStats(RedfishInterface *intf, T callback) {
    intf    - >GetRoot()[“Systems”].Each()[“Processors”].Each().Do([](auto resource) {
        callback(resource[“ProcessorId”]);
    });
```

```
void GetCpuStats(RedfishInterface *intf, T callback) {
    intf    - >GetRoot()[“Chassis”].Each()[“Thermal”].Each().Do([](auto resource) {
        callback(resource[“Reading”]);
    });
}
```

```
void GetCpuThermals(RedfishInterface *intf, T callback) {
    intf    - >GetRoot()[“Chassis”].Each()[“Thermal”].Each().Do([](auto resource) {
        callback(resource[“Reading”]);
    });
    intf    - >GetRoot()[“Chassis”].Each()[“Sensor”].Each().Do([](auto resource) {
        callback(resource[“Reading”]);
    });
}
```

system

memory

dimm0

dimm1

chassis

sensors

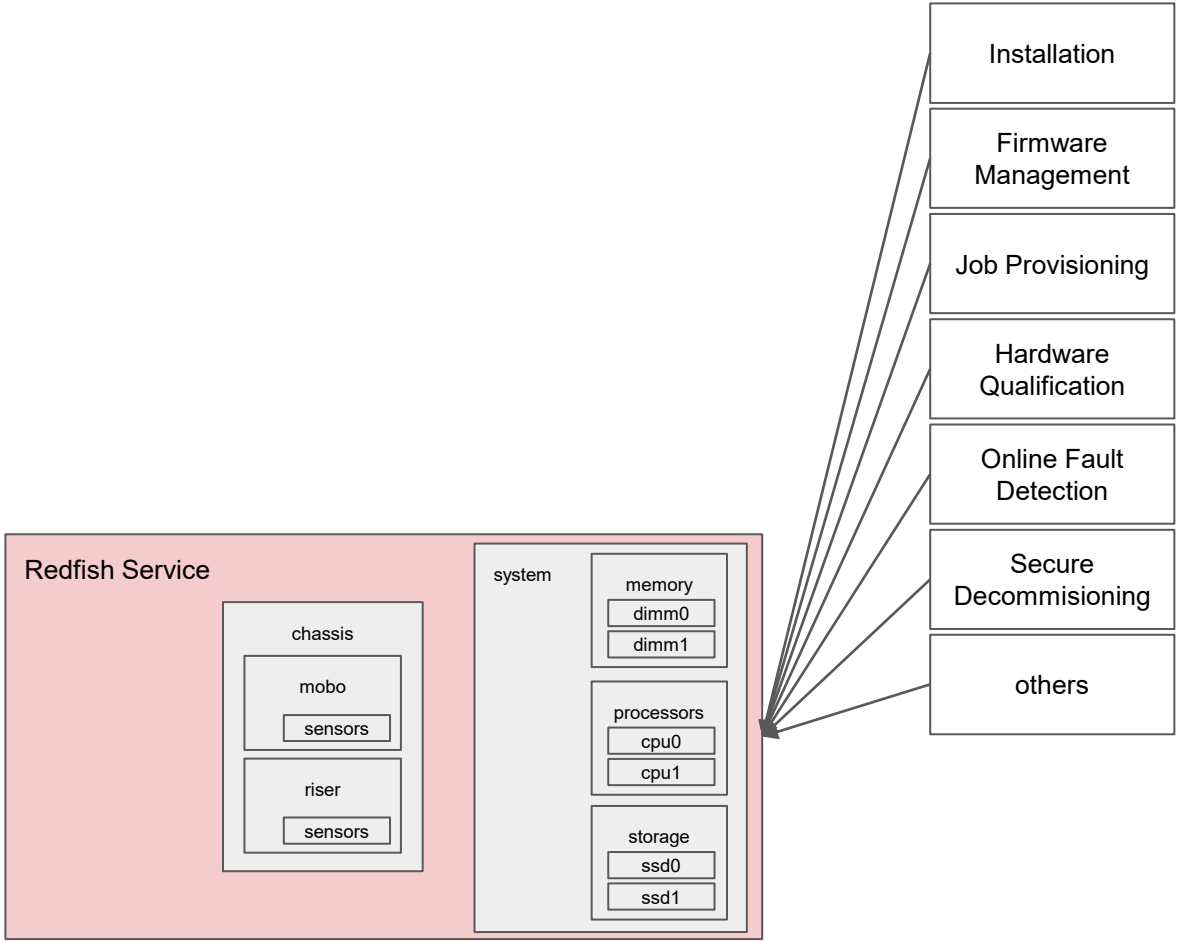
- Installation
- Firmware Management
- Job Provisioning
- Hardware Qualification
- Online Fault Detection
- Secure Decommisioning
- others

Anticipating scaling problems

How do we simplify Redfish usage across the company for many independent teams?

Requirements:

- automate the tracking of Resource and Property requirements
- abstract away the usage of performance features
- provide functional query language that can be programming language agnostic



Redfish query parameters

- Query params are “optional” according to the spec, but they are not optional to Google for automation throughput requirements
  - $expand, $select
  - ETAGs for caching
- Clients need to know when to efficiently use query features if a server has it available

7.3 Query parameters

7.3.1 Query parameter overview

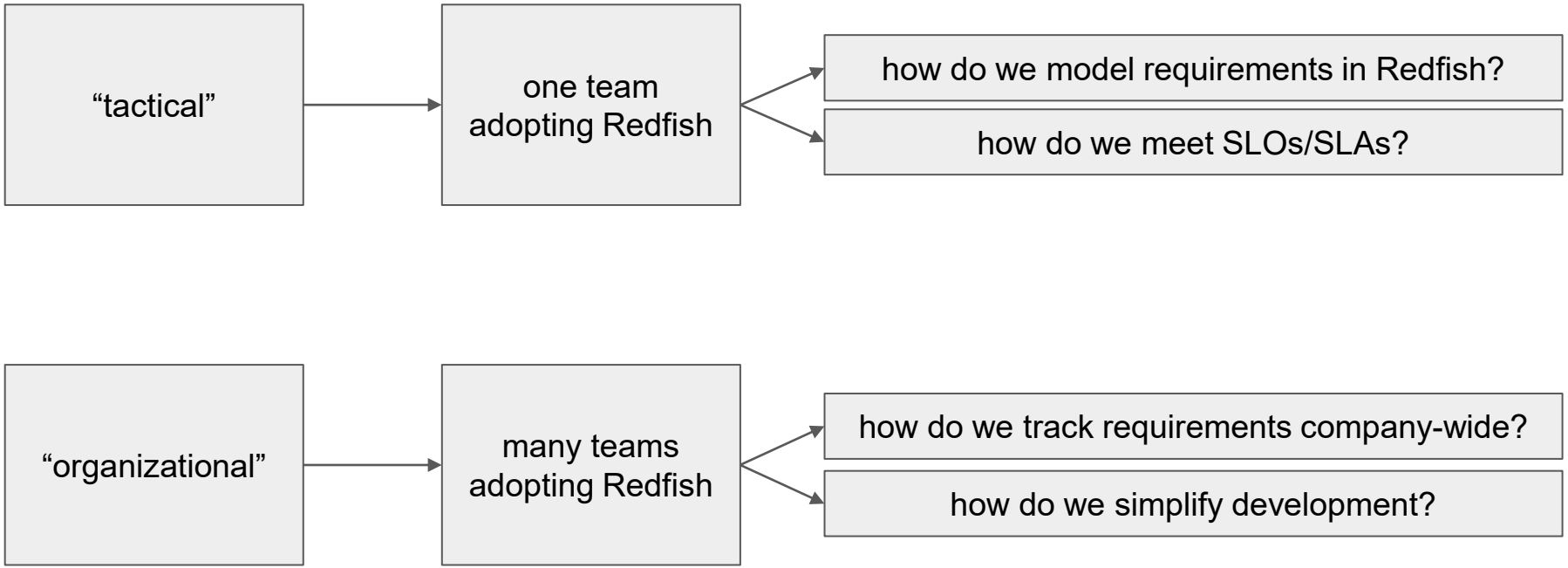
To paginate, retrieve subsets of resources, or expand the results in a single response, clients can include the query parameters. Some query parameters apply only to resource collections.

Services:

- Shall only support query parameters on `GET` operations.
- Should support the `$top`, `$skip`, `only`, and `excerpt` query parameters.
- May support the `$expand`, `$filter`, and `$select` query parameters.
- Shall include the `ProtocolFeaturesSupported` object in the service root, if the service supports query parameters.
  - This object indicates which parameters and options have been implemented.
- Shall ignore unknown or unsupported query parameters that do not begin with `$`.

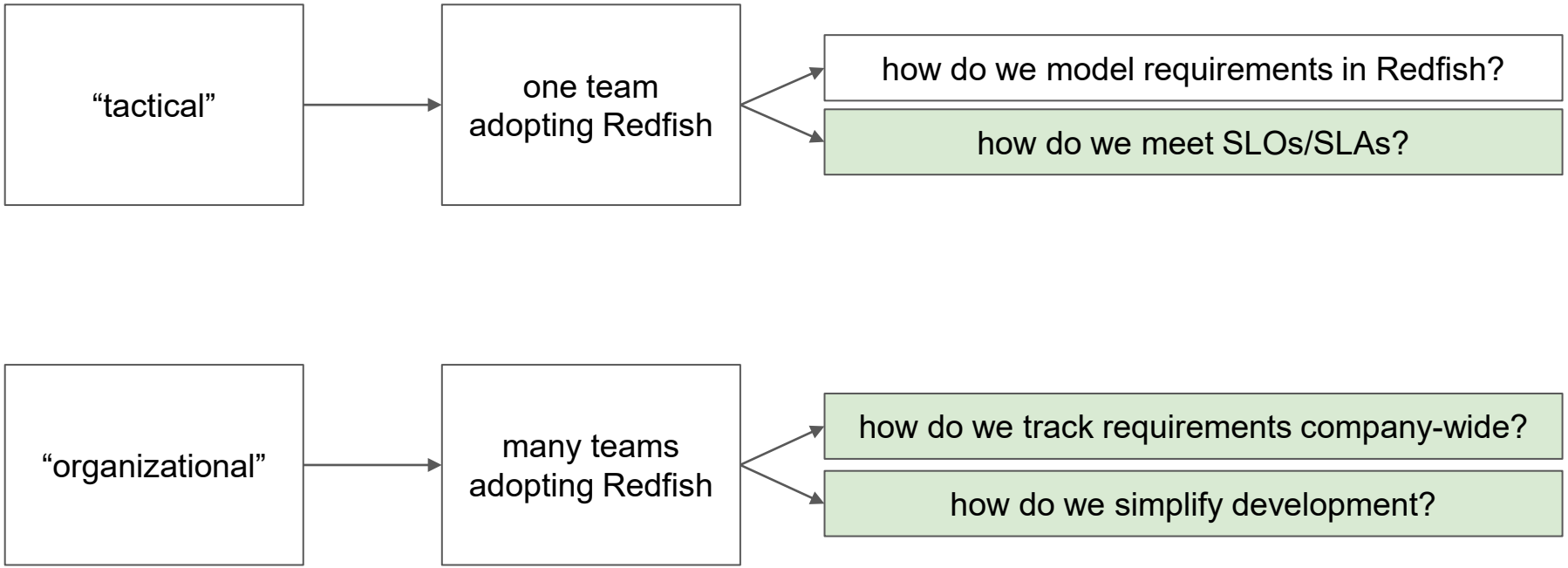
Redfish Specification
Version 1.15.1
April 7 2022

Problems



Redfish Ergonomics Roadmap

Problems



Approach

We plan to provide a common development framework across Google:

- Abstract away Redfish query parameters
- Normalize modelling discrepancies

how do we meet SLOs/SLAs?

how do we simplify development?

As a byproduct of this framework, can we create common development platforms?

- property usage tracking
- performance monitoring

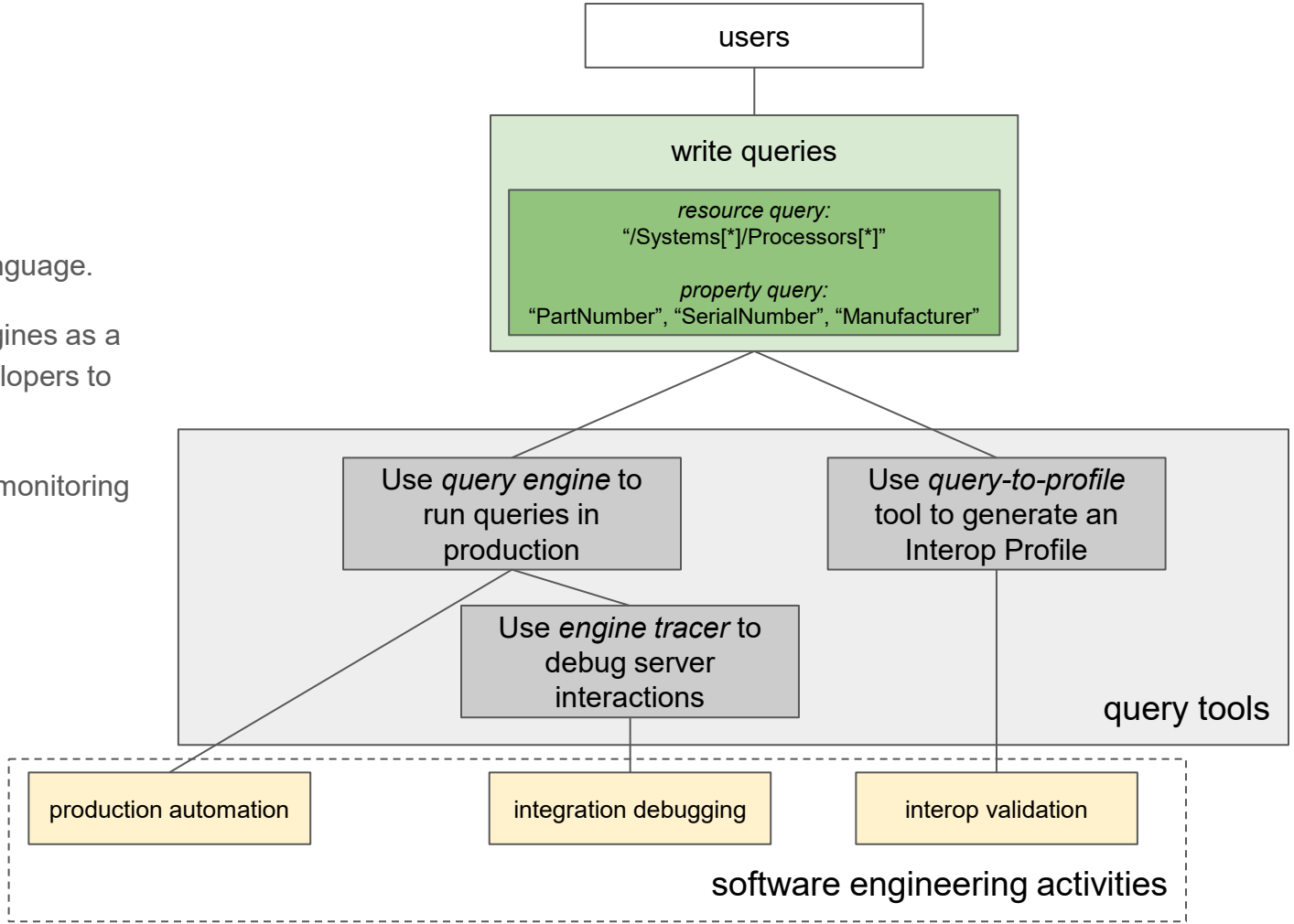
how do we track requirements company-wide?

Proposal

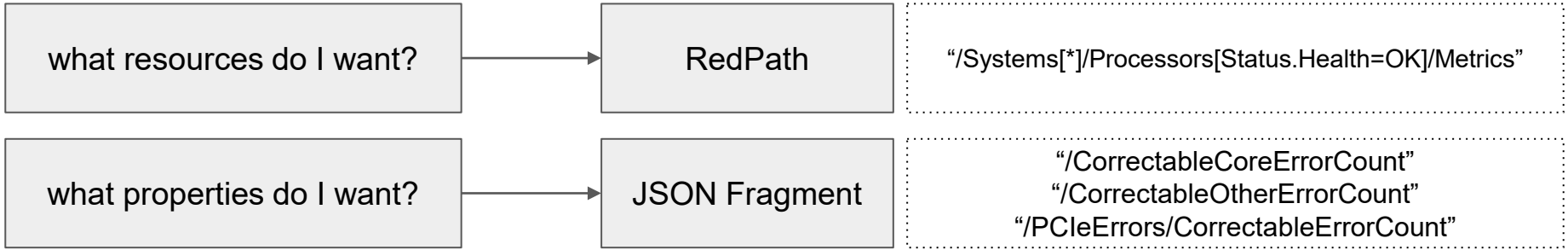
Define a Redfish query language.

Provide Redfish query engines as a common platform for developers to use.

Provide analysis tools for monitoring Redfish client usage.



What’s in a query?



What’s in a query?

```
{
    “QueryName”: “ProcessorErrors”,
    “Subqueries”: [
        {
            “RedPath”: “/Systems[*]/Processors[Status.Health=OK]/Metrics”,
            “PropertyPath”: [
                “/CorrectableCoreErrorCount”,
                “/CorrectableOtherErrorCount”,
                “/PCIeErrors/CorrectableErrorCount”
            ]
        }
    ]
}
```

RedPath

By Patrick Boyd from Dell Technologies

https://github.com/DMTF/libredfish#redpath

Some examples:

- /Chassis[1] - Will return the first Chassis instance
- /Chassis[SKU=1234] - Will return all Chassis instances with a SKU field equal to 1234
- /Systems[Storage] - Will return all the System instances that have Storage field populated
- /Systems[*] - Will return all the System instances
- /SessionService/Sessions[last()] - Will return the last Session instance
- /Chassis[Location.Info] - Will return all the Chassis instances that have a Location field and a Info subfield of Location
- /Systems[Status.Health=OK] - Will return all System instances that have a Health of OK

JSON Fragment

RFC6901: JSON Pointer
https://datatracker.ietf.org/doc/html/rfc6901

The same RFC is used in the Redfish Specification DSP0266 for fragments.

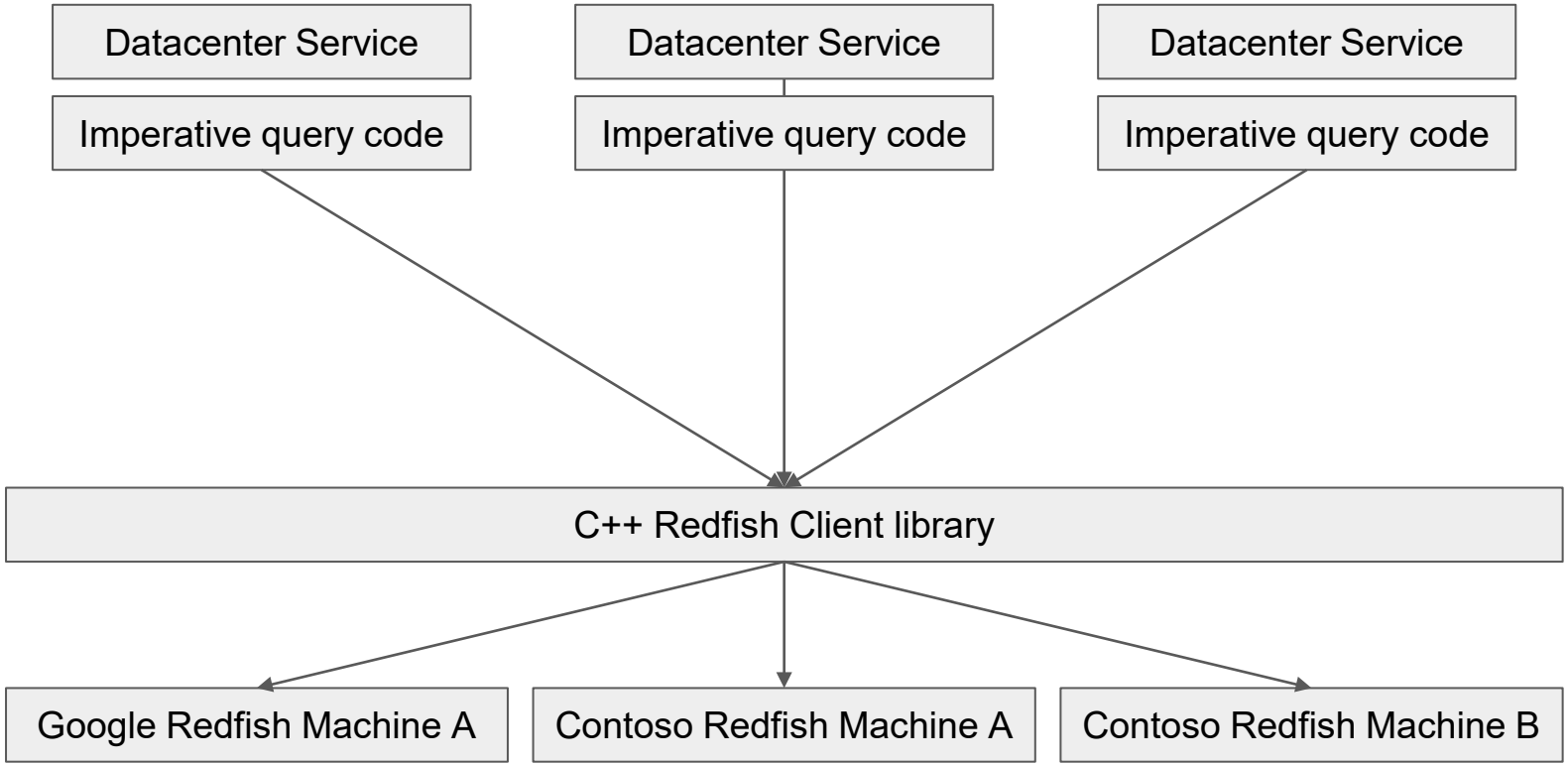
For example, given the JSON document

```
{
   "foo": ["bar", "baz"],
   "": 0,
   "a/b": 1,
   "c%d": 2,
   "e^f": 3,
   "g|h": 4,
   "i\\j": 5,
   "k\"l": 6,
   " ": 7,
   "m~n": 8
}
```

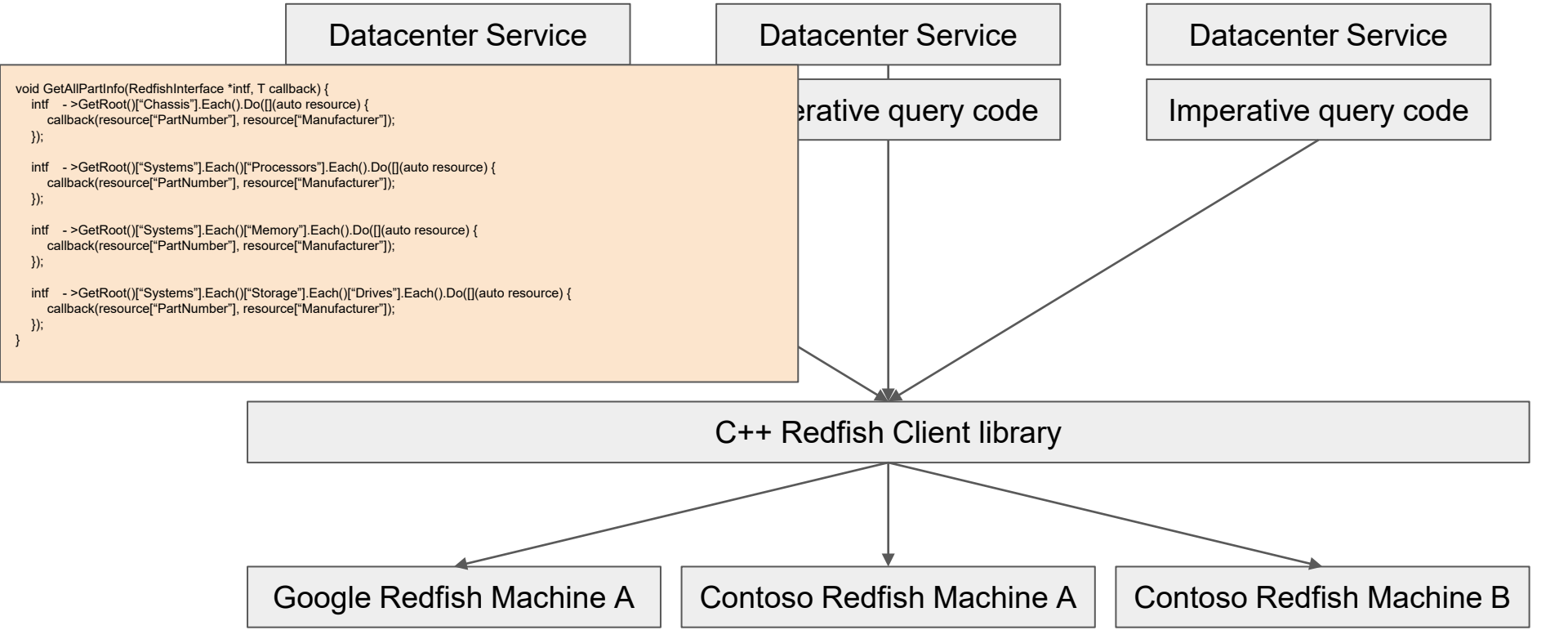
The following JSON strings evaluate to the accompanying values:

```
 ""           // the whole document
 "/foo"       ["bar", "baz"]
 "/foo/0"     "bar"
 "/"          0
 "/a~1b"      1
 "/c%d"       2
 "/e^f"       3
 "/g|h"       4
 "/i\\j"      5
 "/k\"l"      6
 "/ "         7
 "/m~0n"      8
```

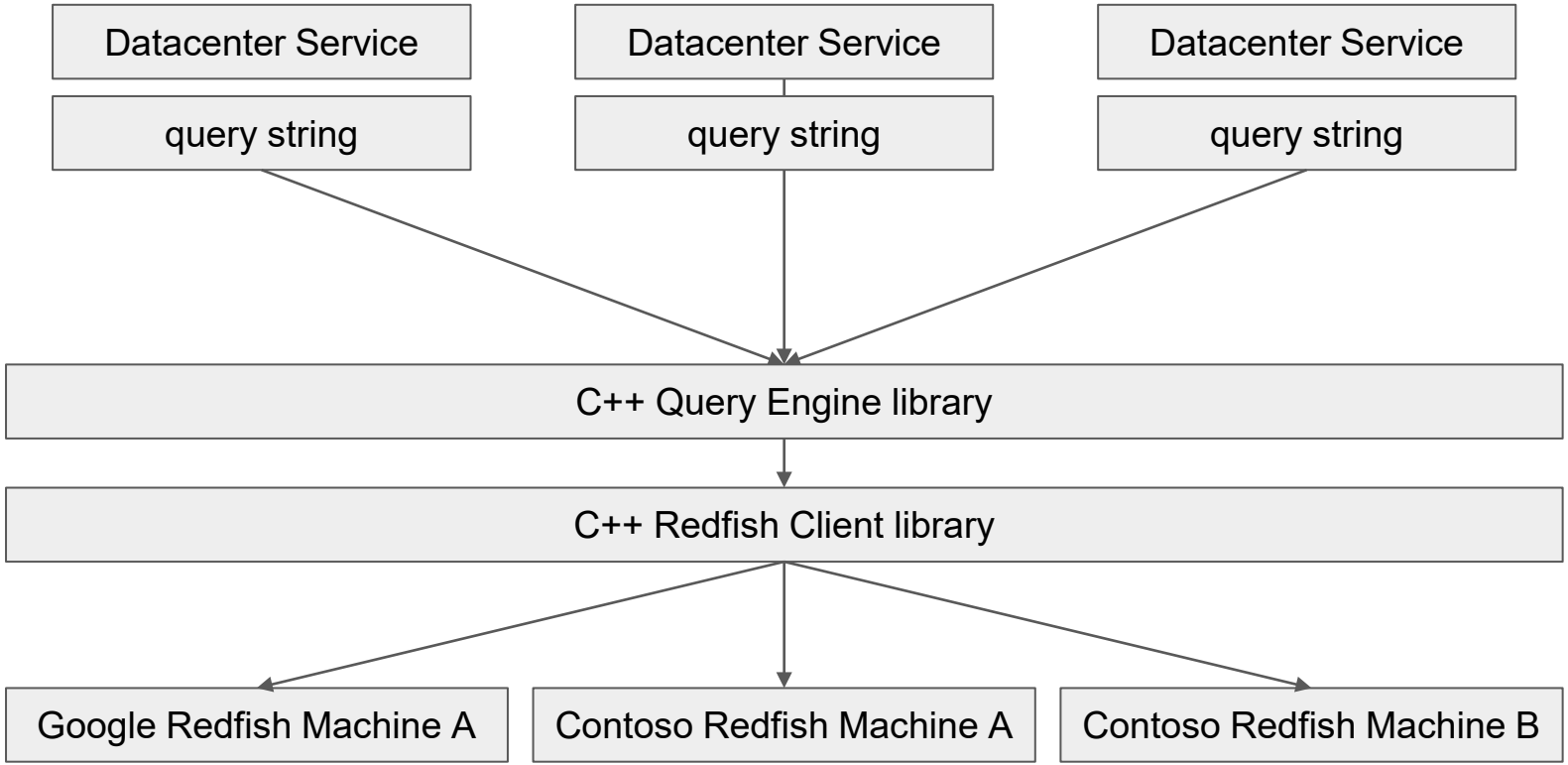
Queries are more portable than code



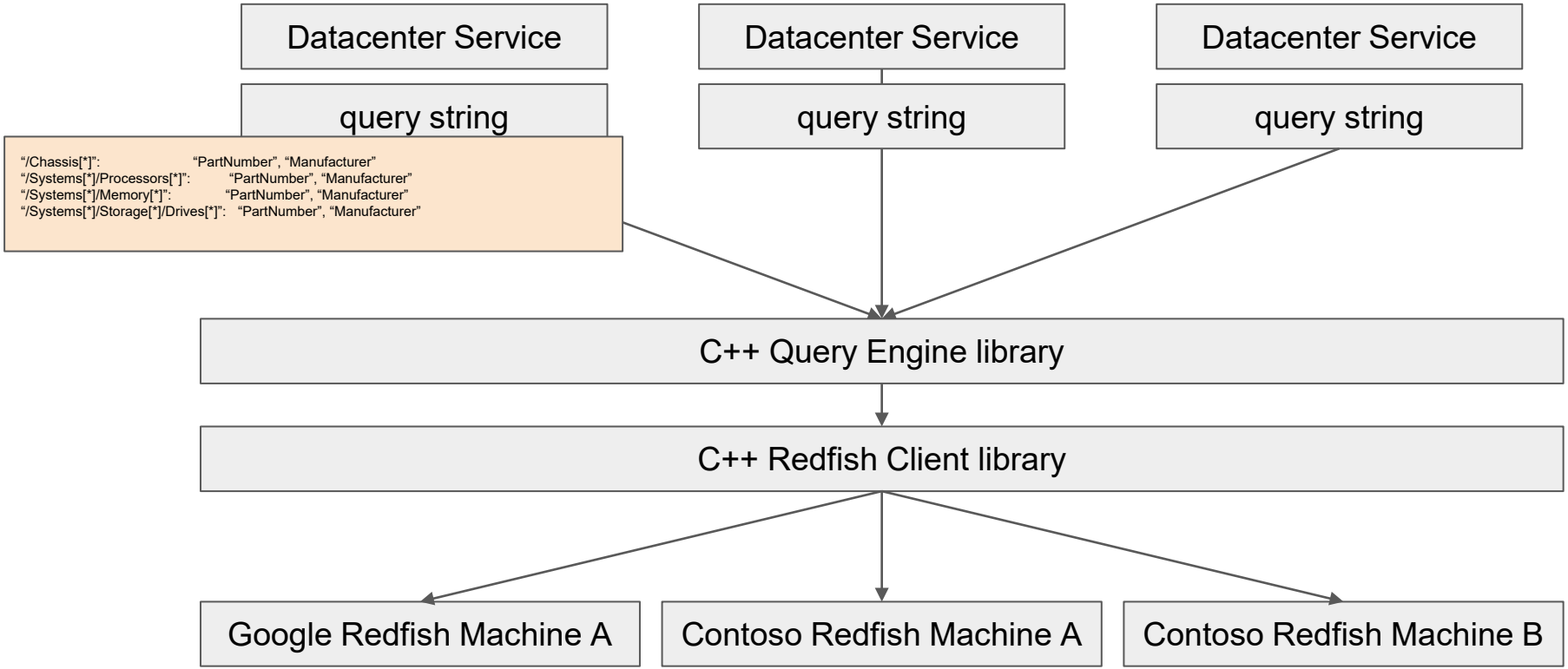
Queries are more portable than code



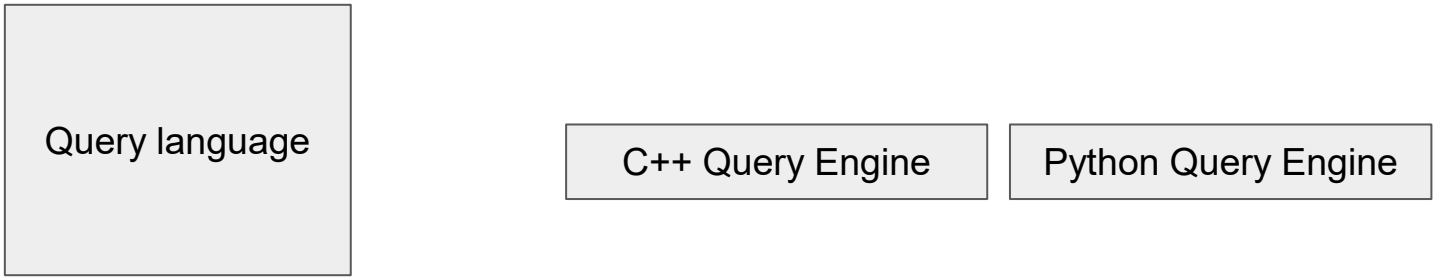
Queries are more portable than code



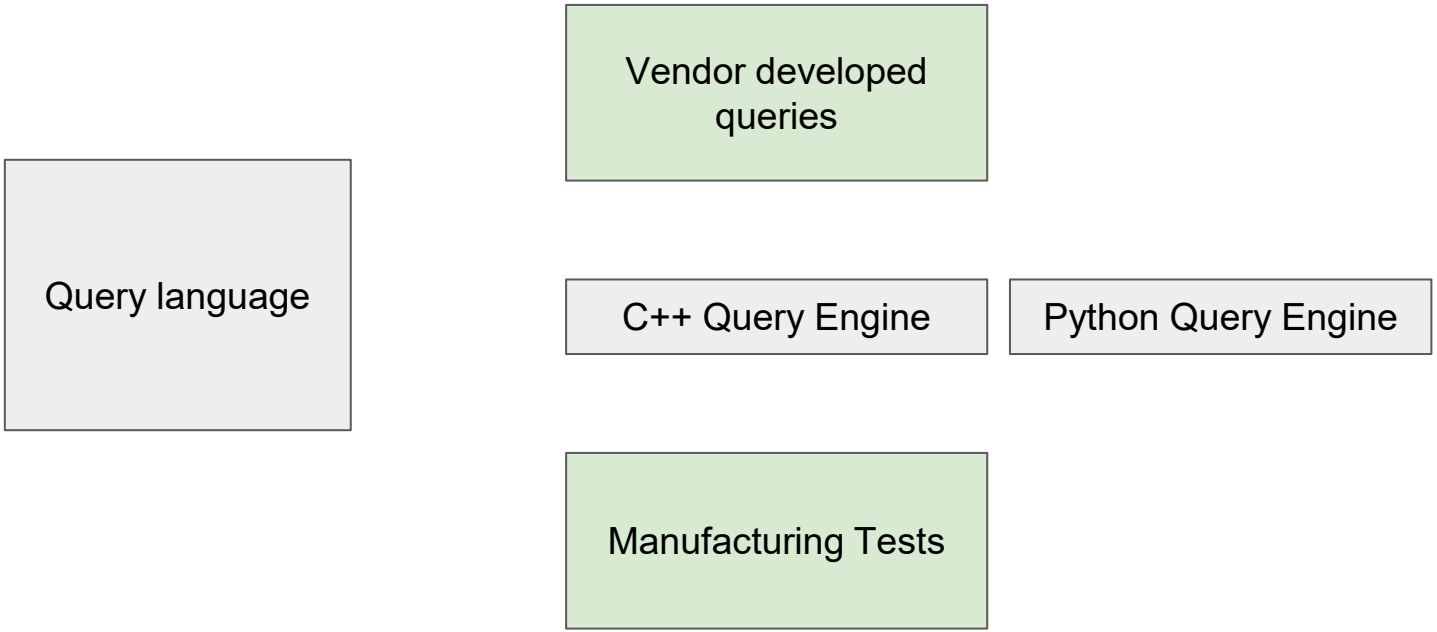
Queries are more portable than code



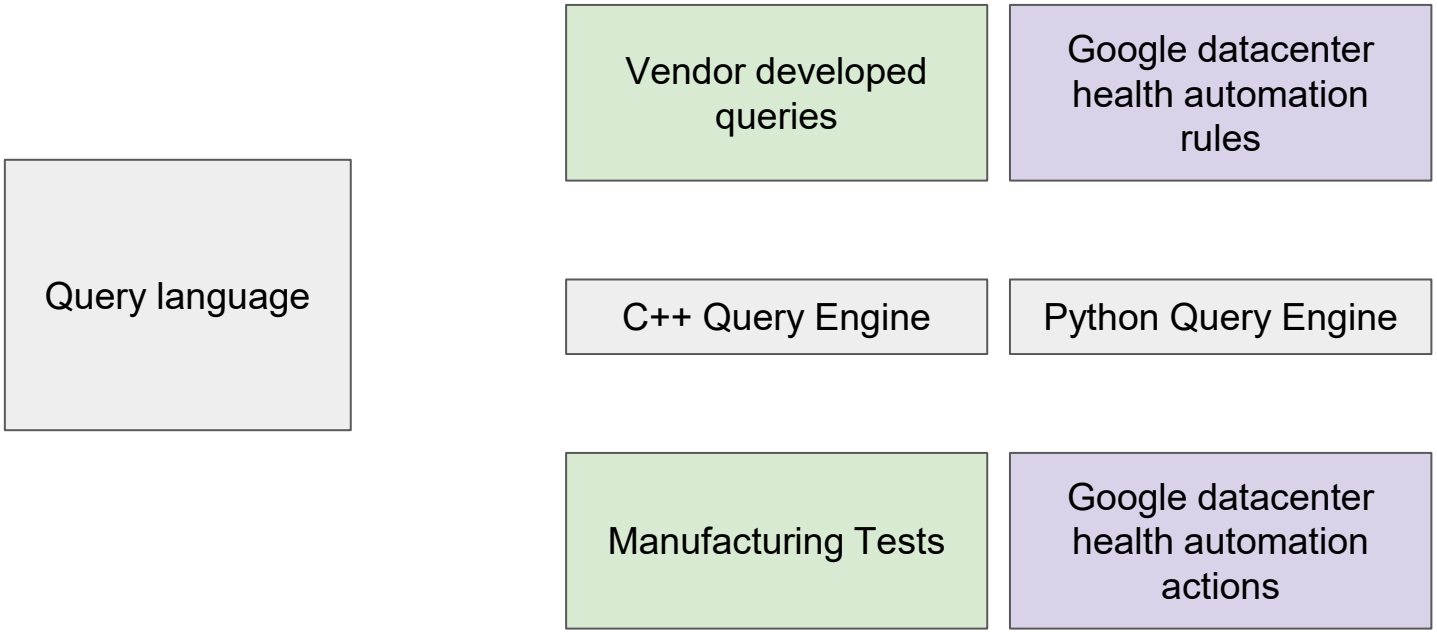
Queries are more portable than code



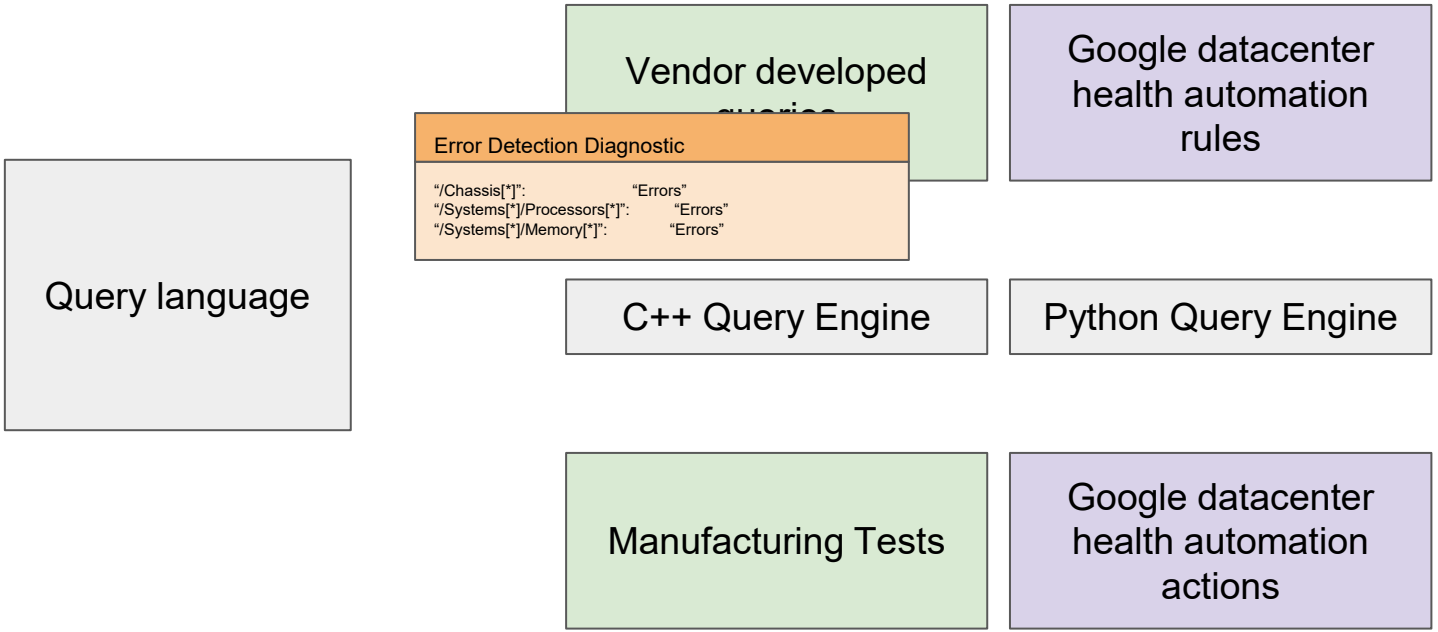
Queries are more portable than code



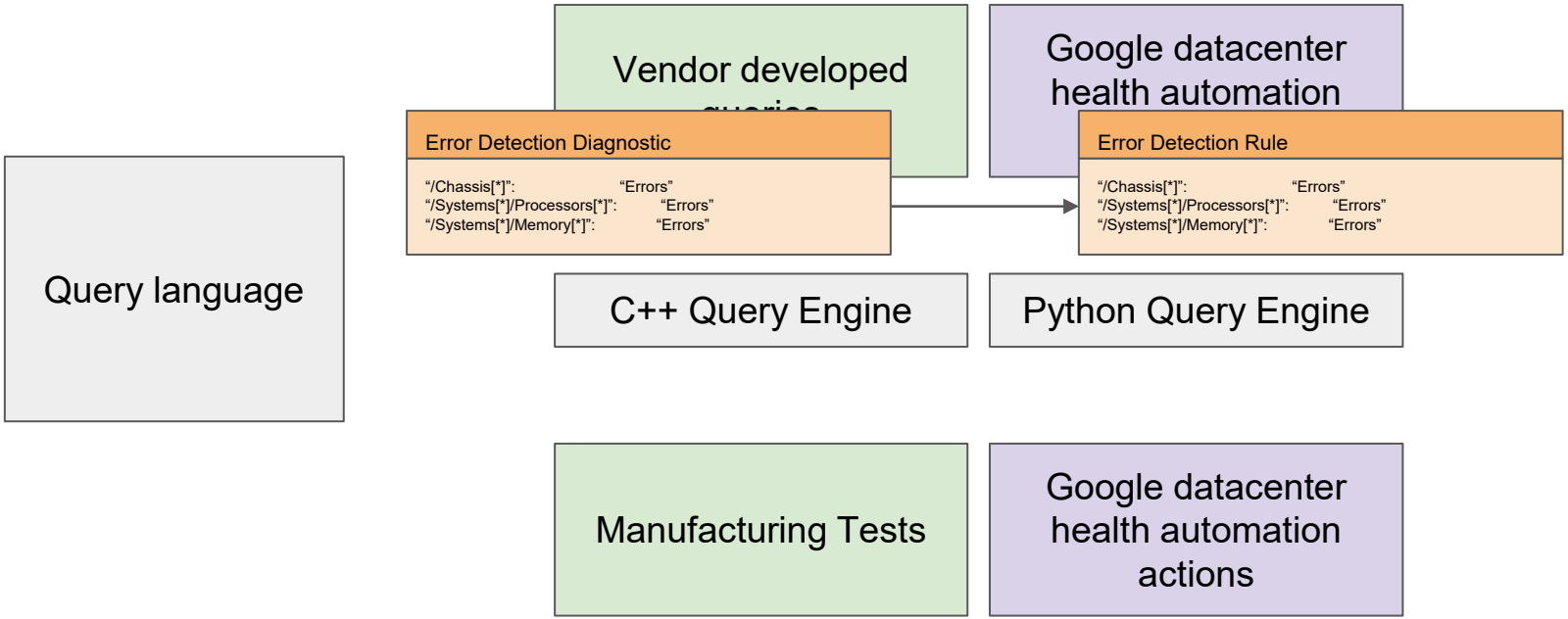
Queries are more portable than code



Queries are more portable than code

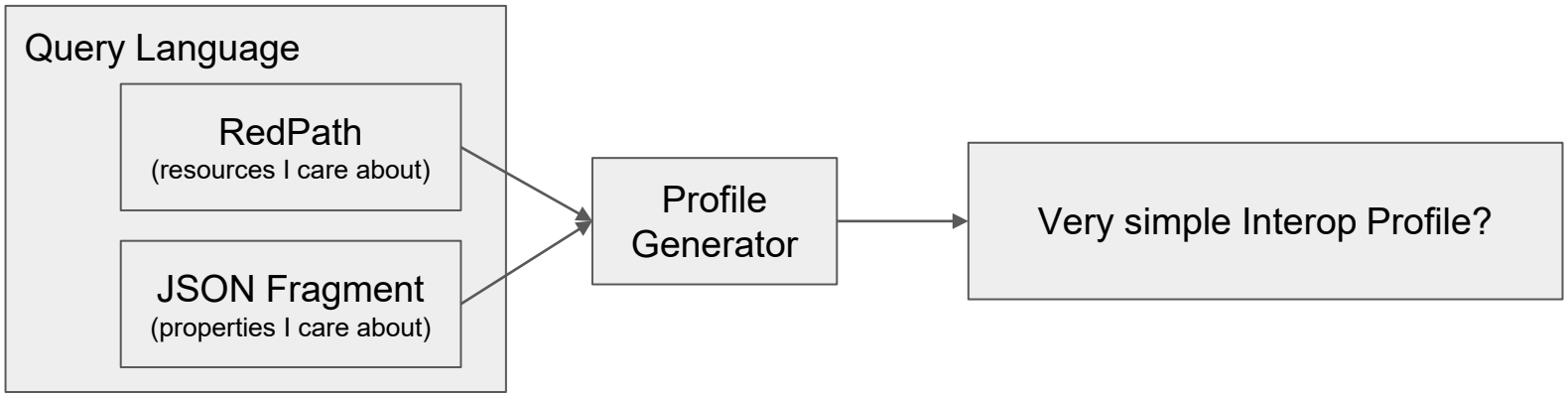


Queries are more portable than code

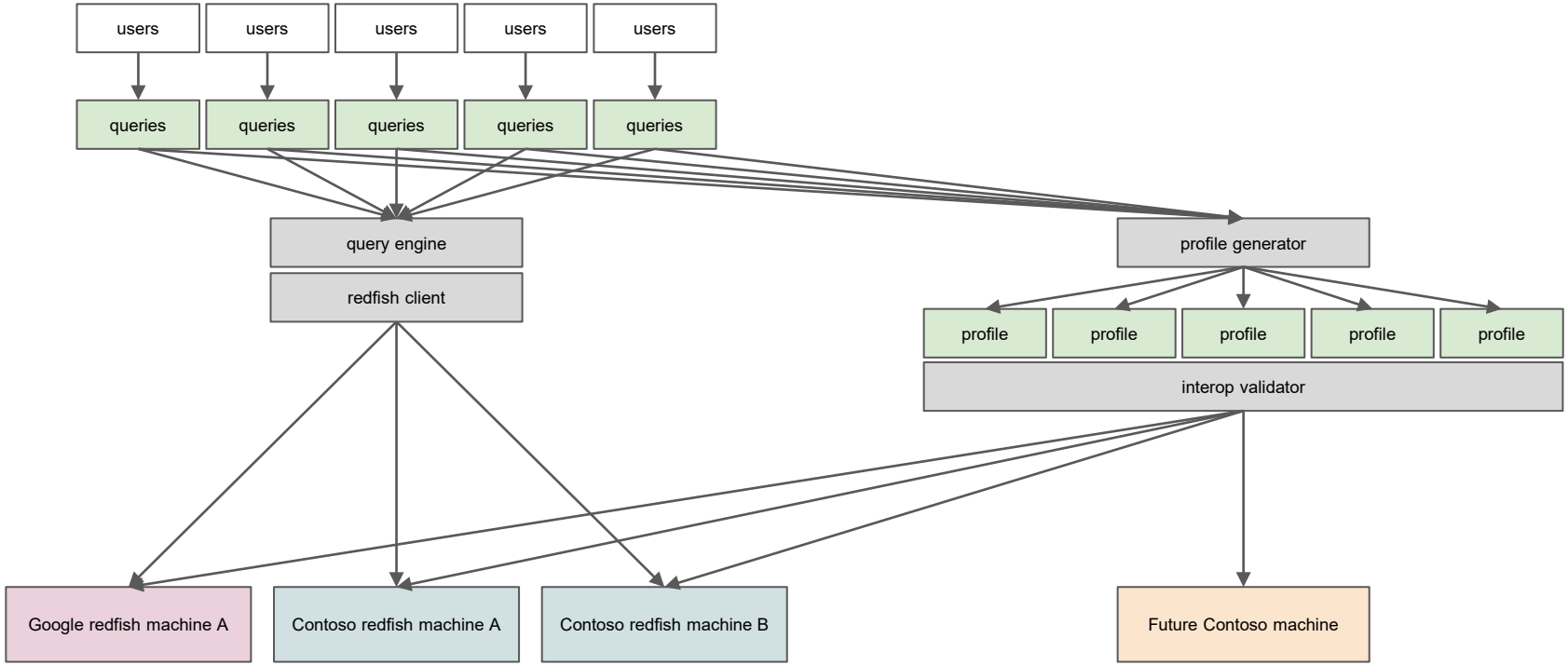


Usage tracking

- Hard to keep track of different property/resource requirements, especially with more users onboarding to Redfish
- Nobody likes to write Interop Profiles
- Could we generate Profiles automatically from queries?



Our ergonomics roadmap



We’re looking for feedback

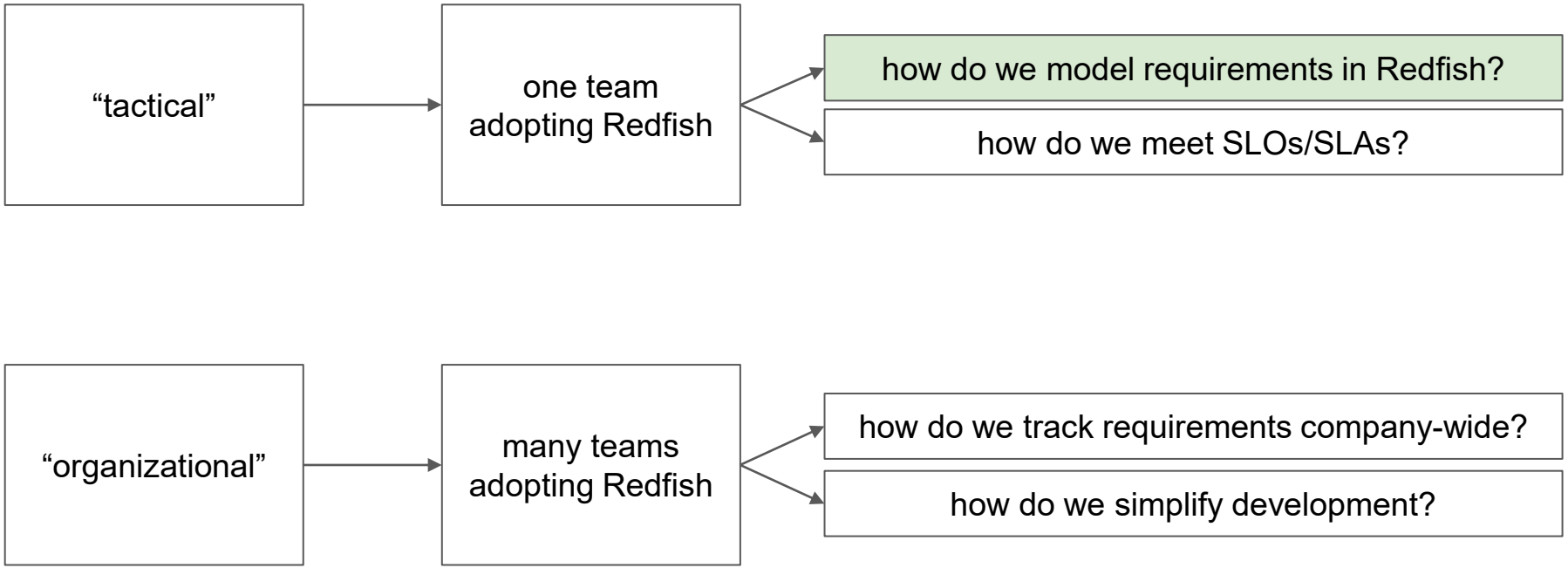
- Can we standardize on these client interfaces?

- Tools will be developed in our GitHub repo for Machine Management
- https://github.com/google/ecclesia-machine-management
- We participate actively in the DMTF Tools Task Force

Data modelling

Normalization with legacy telemetry

Problems

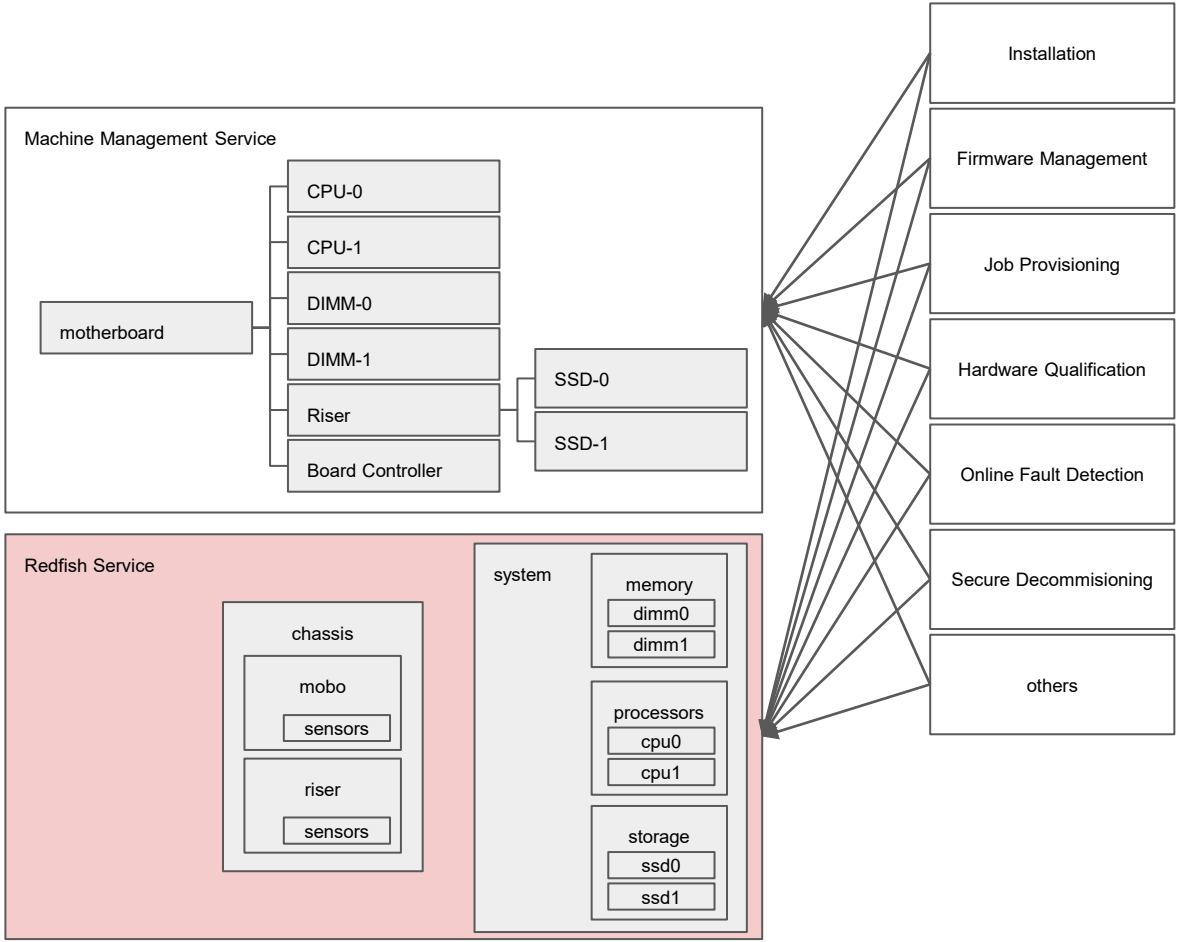


Legacy compatibility

Over a decade of automation has been developed using the legacy management system.

Switching automation to Redfish either means:

- make the legacy API match Redfish
- make Redfish match the legacy API
- have an abstraction layer to make both APIs match



Making Redfish match the legacy API

The legacy API exposes more debug information than Redfish does today

- Low level telemetry requirements (exporting GPIOs)
- Custom telemetry (e.g. custom kernel sysfs files)

This produces some anxiety within client teams at Google:

- Speed of upstreaming concerns
  - On paper, DMTF interactions are scary because they are not in the control of our company
  - In practice, DMTF has been responsive to GitHub issues and pulls
- Generalizability concerns
  - If we are the only exporters of this sort of telemetry, why switch from legacy to Redfish?

Case study: NVMe telemetry

Repair automation rules are based on low-level commands/log page (Identify, Device Self-test, FW Slot Info…).

What to do?

- change legacy services to match Redfish's higher level statuses, and migrate all the clients?
- upstream low-level properties into the Redfish/Swordfish standards?

Figure 196: Get Log Page – Firmware Slot Information Log

Bytes	Description
00	**Active Firmware Info (AFI):** Specifies information about the active firmware revision. Bit 7 is reserved. Bits 6:4 indicates the firmware slot that is going to be activated at the next Controller Level Reset. If this field is 0h, then the controller does not indicate the firmware slot that is going to be activated at the next Controller Level Reset. Bit 3 is reserved. Bits 2:0 indicates the firmware slot from which the actively running firmware revision was loaded.
07:01	Reserved
15:08	**Firmware Revision for Slot 1 (FRS1):** Contains the revision of the firmware downloaded to firmware slot 1. If no valid firmware revision is present or if this slot is unsupported, this field shall be cleared to 0h.
23:16	**Firmware Revision for Slot 2 (FRS2):** Contains the revision of the firmware downloaded to firmware slot 2. If no valid firmware revision is present or if this slot is unsupported, this field shall be cleared to 0h.

NVM Express
Revision 1.4
June 10, 2019

Case study: custom hardware firmware update

Firmware update process requires setting a GPIO to disable write protect on one portion of storage.

Legacy API: expose a GPIO that updater knows to set to HIGH.

Redfish: patch Updateable=True in FirmwareInventory.

What to do?

- change all users of the legacy API to new semantics for *Updateable*?
- keep the legacy API and emulate a GPIO using the Redfish property?

Case study: hardware identification

Hardware ID scheme must be consistent with the legacy system.

What to do?

- Make Redfish servers export Google's IDs?
- Make Google's systems with decades of historical data change their ID scheme to something in Redfish?

Solution:

- Meet in the middle: generalize the Google ID scheme, and make it possible to derive Google IDs using standard Redfish properties
- Concerns: if we are the only folks using this scheme, we cannot rely on the properties being available on arbitrary off-the-shelf systems
- Very interested in knowing what other companies are doing, and whether there is opportunity to standardise on a hardware identification scheme

Repair Path IDs

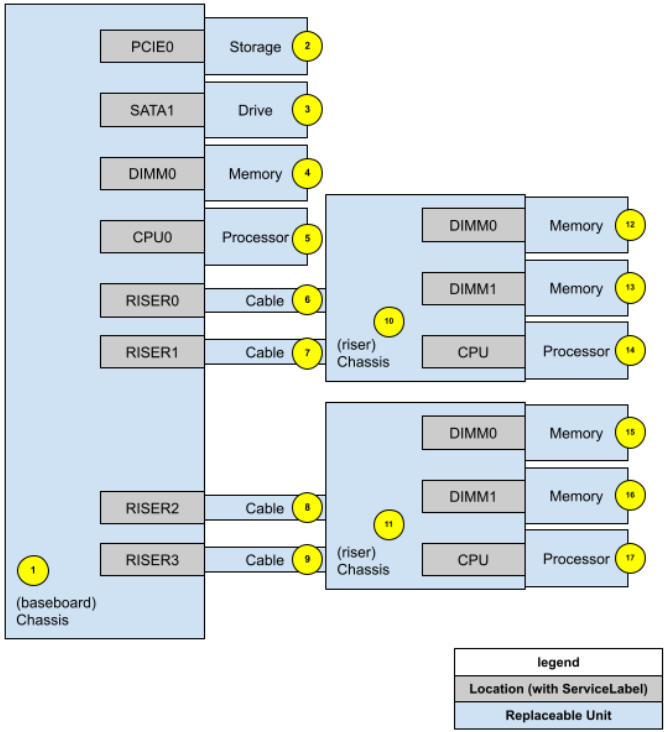
1: “the baseboard” []

5: “the processor with service label CPU0 on the baseboard” [“CPU0”]

14: “the processor with service label CPU on the riser attached to the cable with service label RISER0” [“RISER0”, “DOWNLINK”, “CPU”]

17: “the processor with service label CPU on the riser attached to the cable with service label RISER2” [“RISER2”, “DOWNLINK”, “CPU”]

https://github.com/google/ecclesia-machine-management/blob/master/ecclesia/lib/redfish/g3doc/topology.md



Some upcoming usecases

Storage key management

- Lifecycle
  - Write a cryptographic seed to some hardware
  - Generate keys from the seed
  - Use keys to unlock storage during boot
  - Monitor the health of the key
  - Drainless key rotation
  - Emergency password recovery
- Things needing to be modelled
  - cryptographic seed: logical resource?
  - the hardware(s) storing the cryptographic seed
- Swordfish plans for Google?
  - Want SecuritySend/Receive
  - Maybe propose some action to StorageController? Schema proposal still work in progress.

SATA Disk: TCG, ATA, S.M.A.R.T

- Enterprise TCG security mode feature set
- Hybrid SMR feature sets
- Legacy ATA security feature set for drive telemetry and control
- Use S.M.A.R.T. for failure detection

NVMe-MI

- Enumeration - find all NVMe-MI devices in the system
- Thermal monitoring - GetLogPage
- Firmware update - Need Admin vendor-unique (VU) commands (Prepare, Done)
- Telemetry - GetLogPage, GetFeatures, Identify, VU commands
- TCG passthrough interface

Other misc. low-level telemetry

- Hardware specific fuse states
- Profiling data
- Low level electrical debugging
  - voltage rail status, voltage margining
  - debug mode status, watchdog
  - specific failure signals exposed as GPIOs
- Telemetry decoding
  - Some Redfish endpoints cannot store our decoder, so decoding needs to be done by a Redfish client/proxy
  - Raw telemetry being exposed as LogEntry today; we’re not expecting this solution to be scalable

In summary

Main challenges for Redfish adoption

Software Development

- Simplify the cross-org client adoption of Redfish
- Track Redfish usage across the organization

Modelling

- Normalizing telemetry to legacy telemetry schemes (e.g. ID schemes)
- Upstreaming and generalizing the gaps in upcoming usecases

Get in touch!

- Library development at https://github.com/google/ecclesia-machine-management
- Email: dchanman@google.com