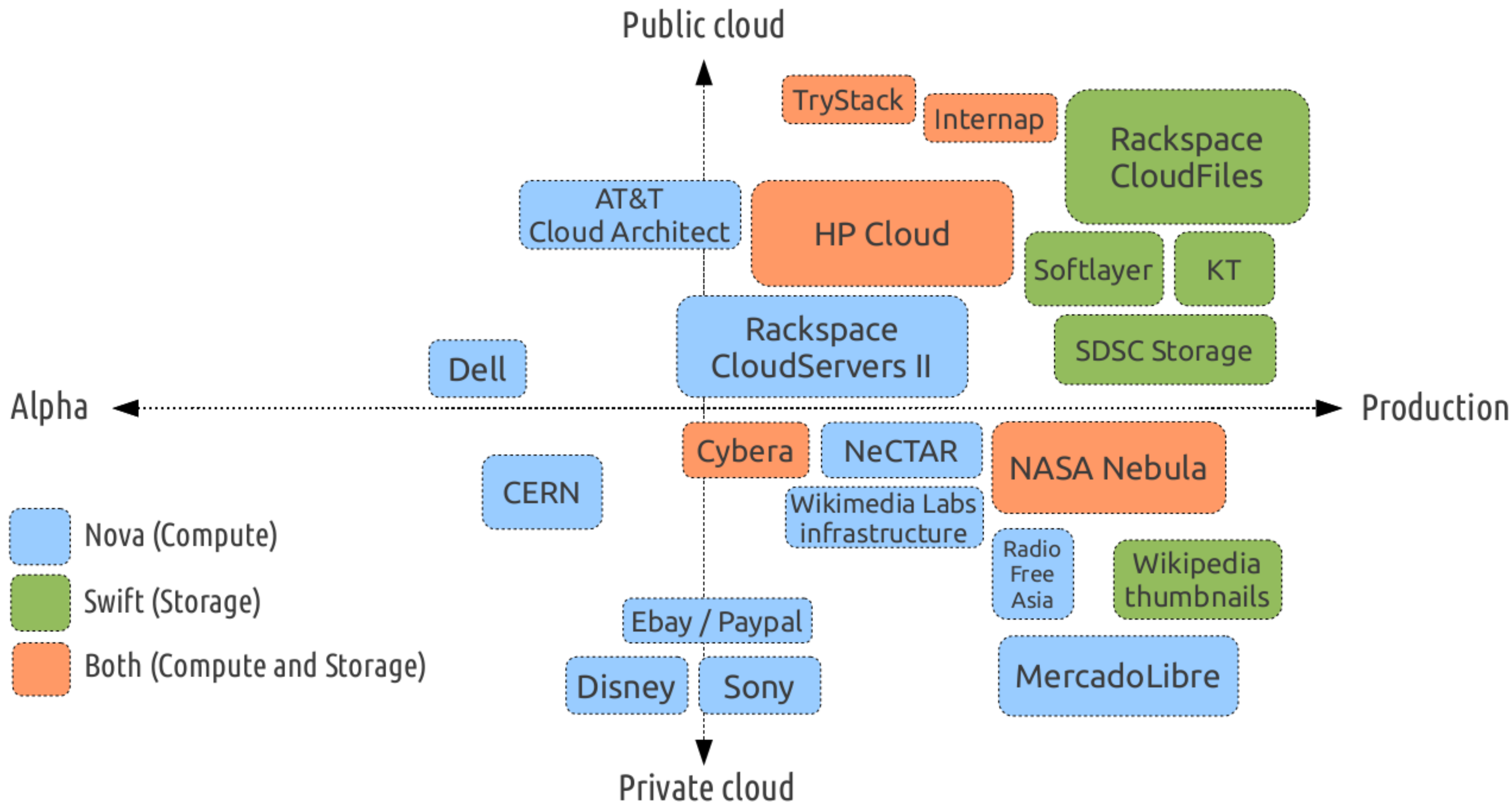
On Fault Resilience of Open Stack

Subho Banerjee

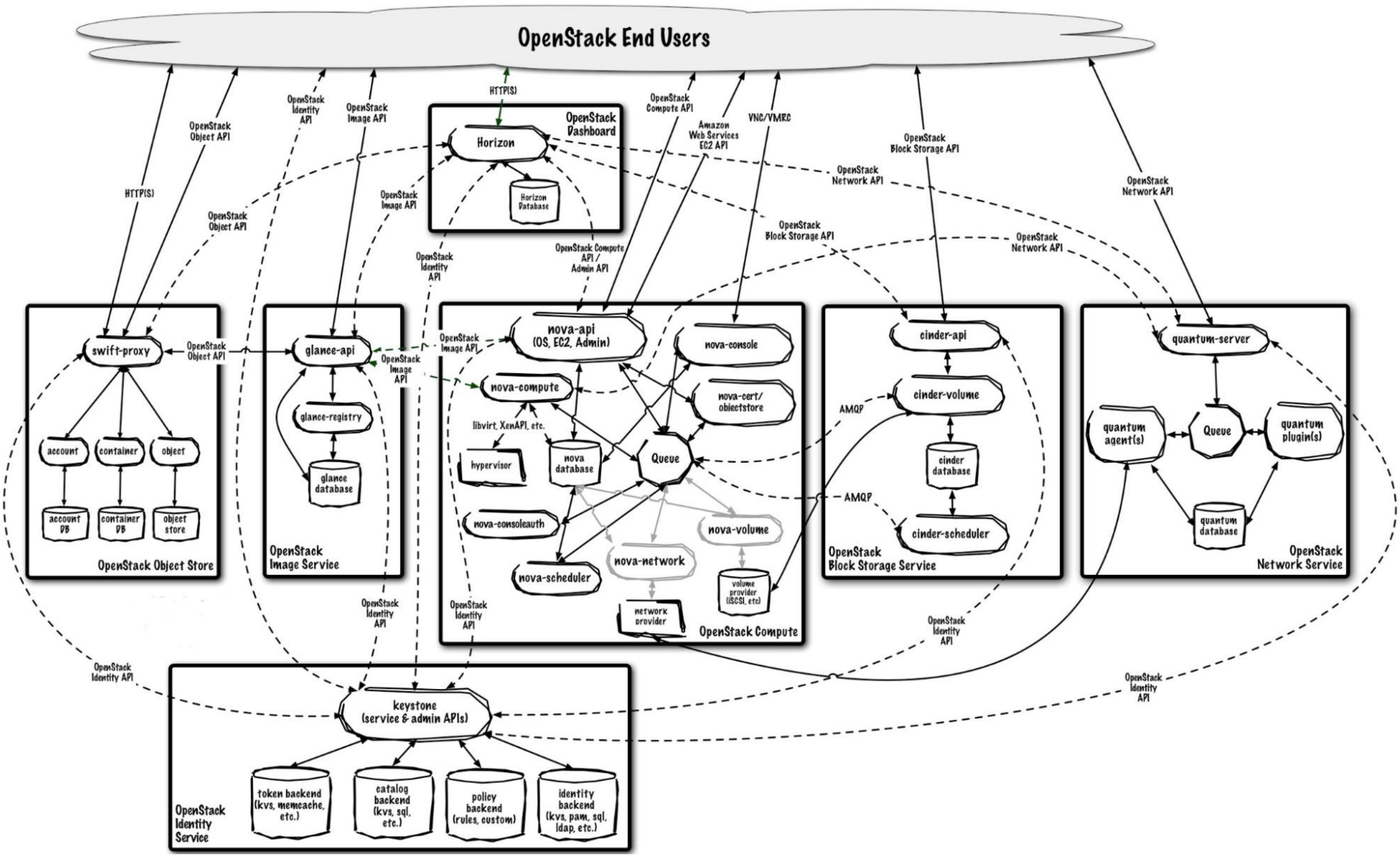
OpenStack

- Control and automate pools of resources
- Efficiently allocate resources
- Empower admins & users via self-service portals
- Provide APIs to make apps cloud-aware

Broad Commercial Support



Grizzly Logical Architecture



Overview

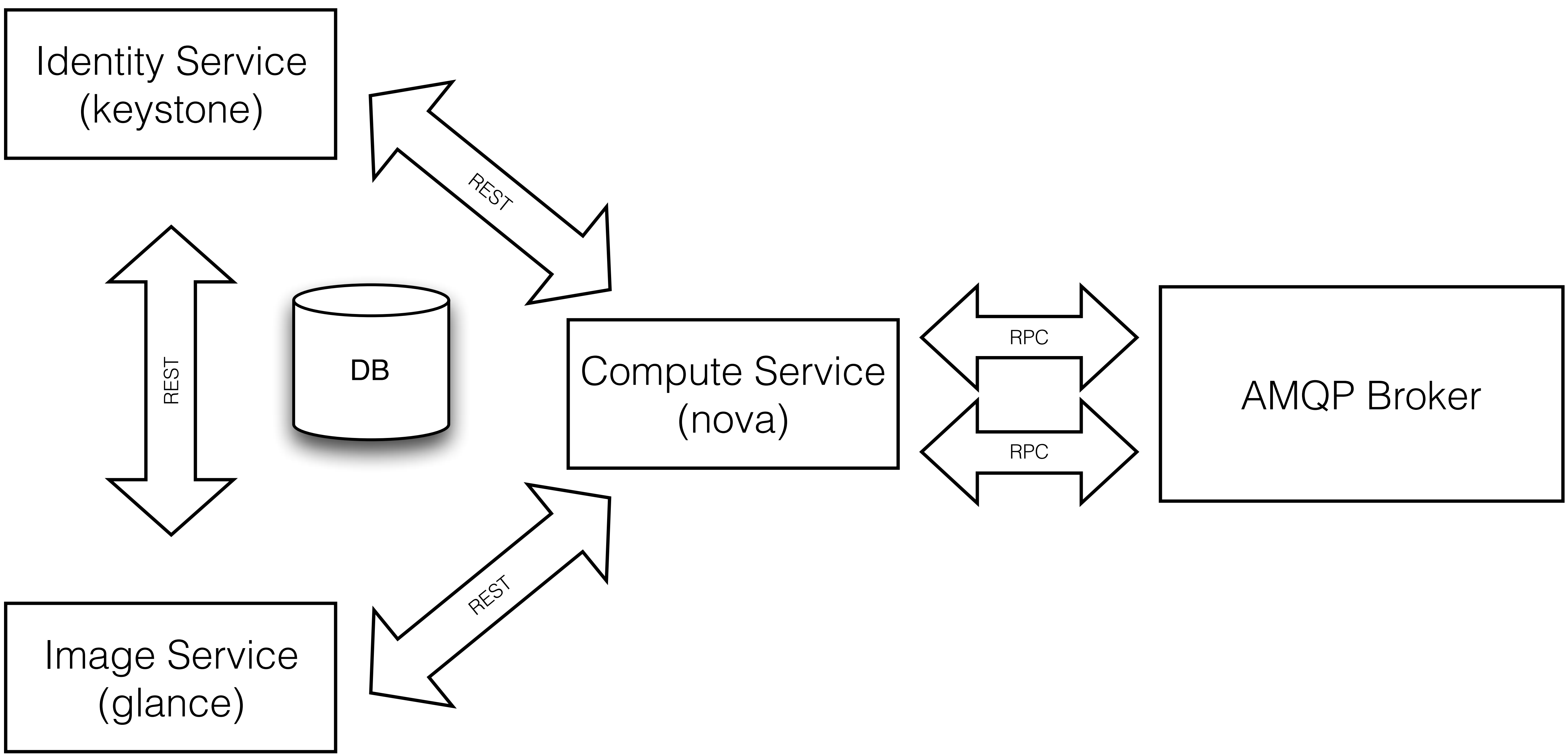
- Fault Injection Framework for OpenStack
    - Target intra/inter service communication during processing of an external request
- Study fault resiliency of OpenStack
    - Fault Resiliency — Maintain correct functionalities under faults
    - OpenStack essex and grizzly
- Categorize bug categories dealing with fault resilience issues

Fault Injection Framework

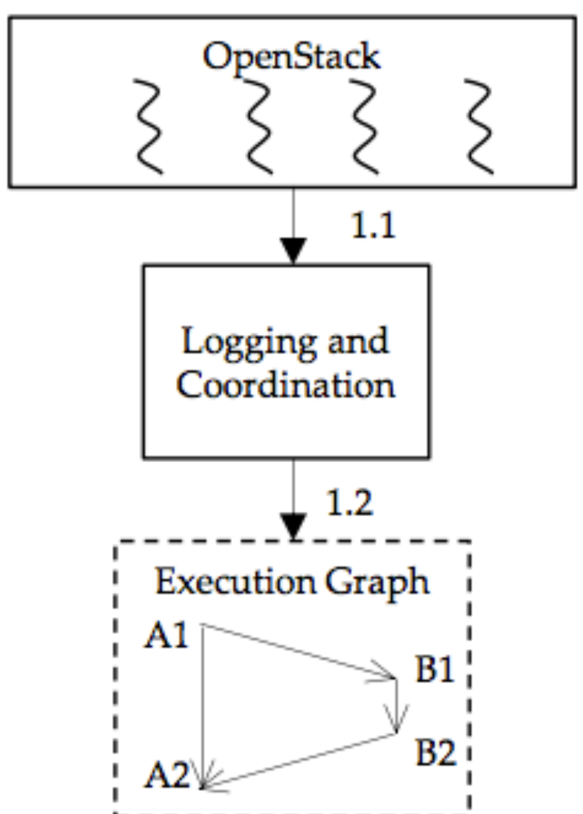
Design Principles

- Inject faults in communication flows
- Expose OpenStack's high level semantics to the fault injection module
- Consider 11 most commonly used API endpoints

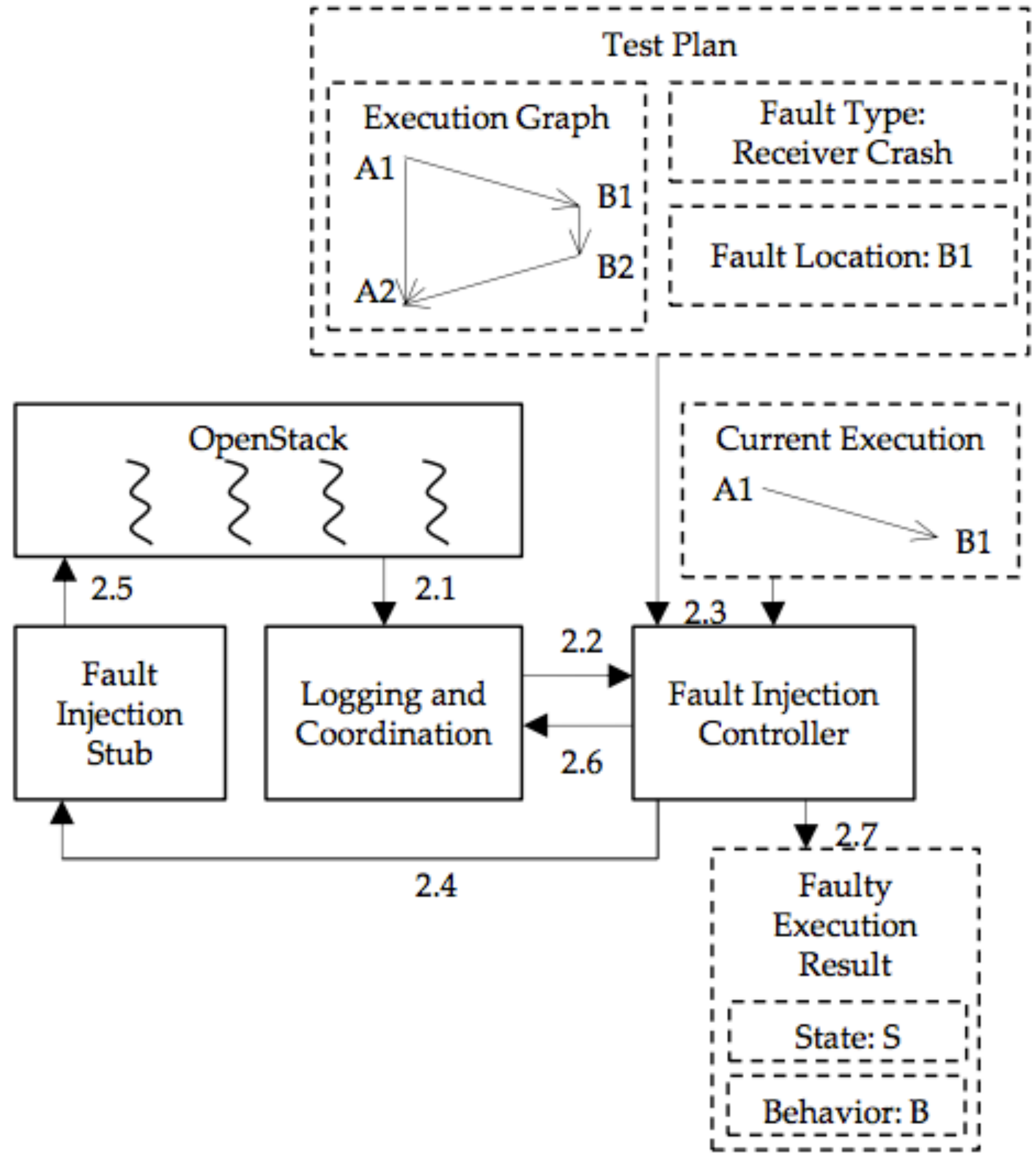
Parts of the stack being tested



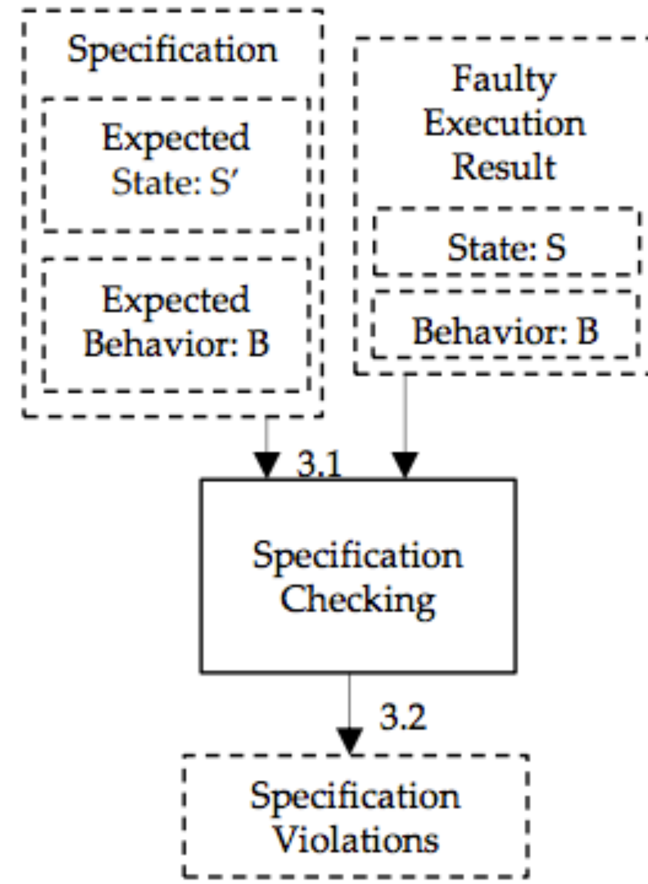
1. Construct execution graphs for external requests



2. Performing fault injection and collecting results

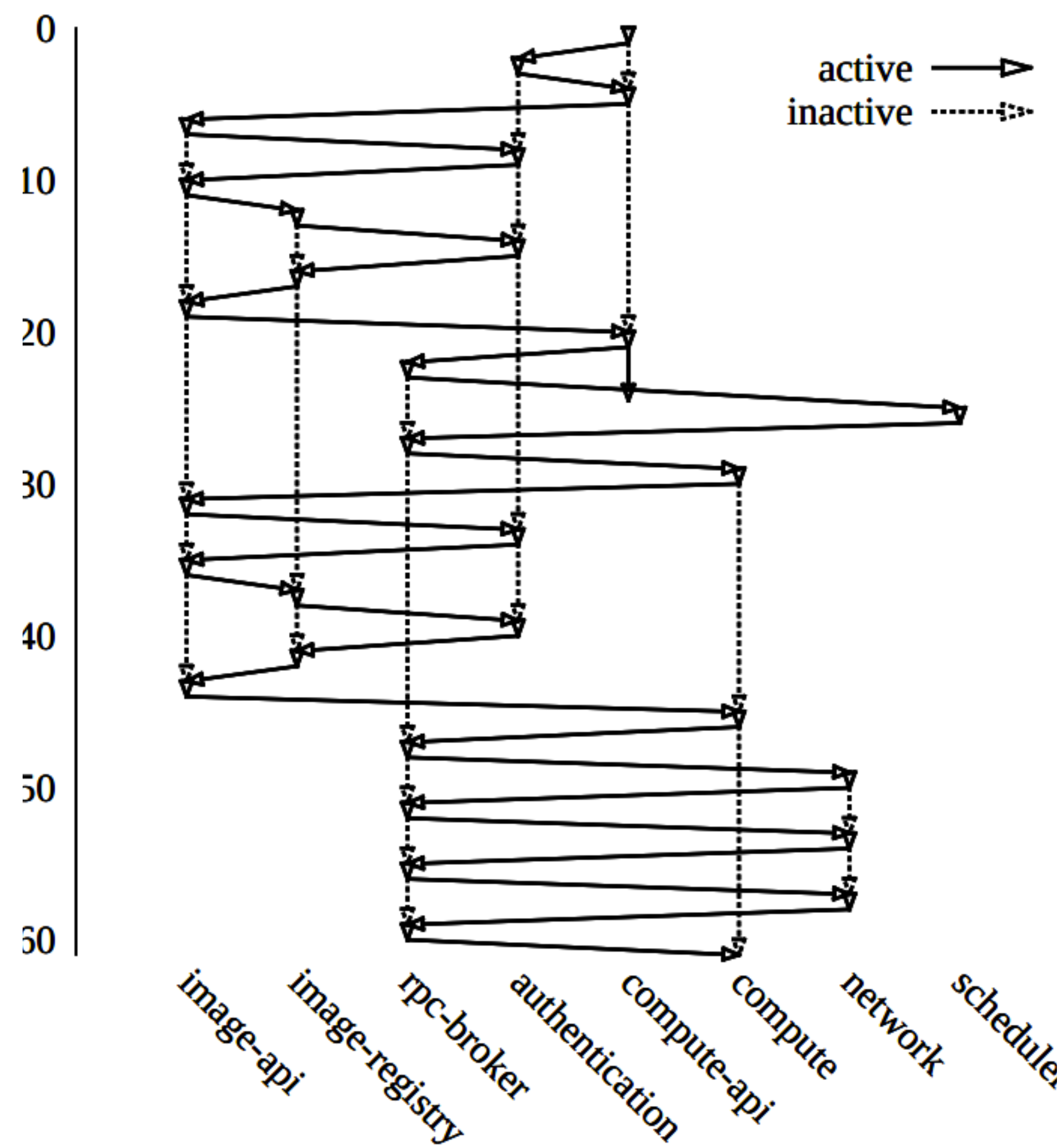


3. Specification checking

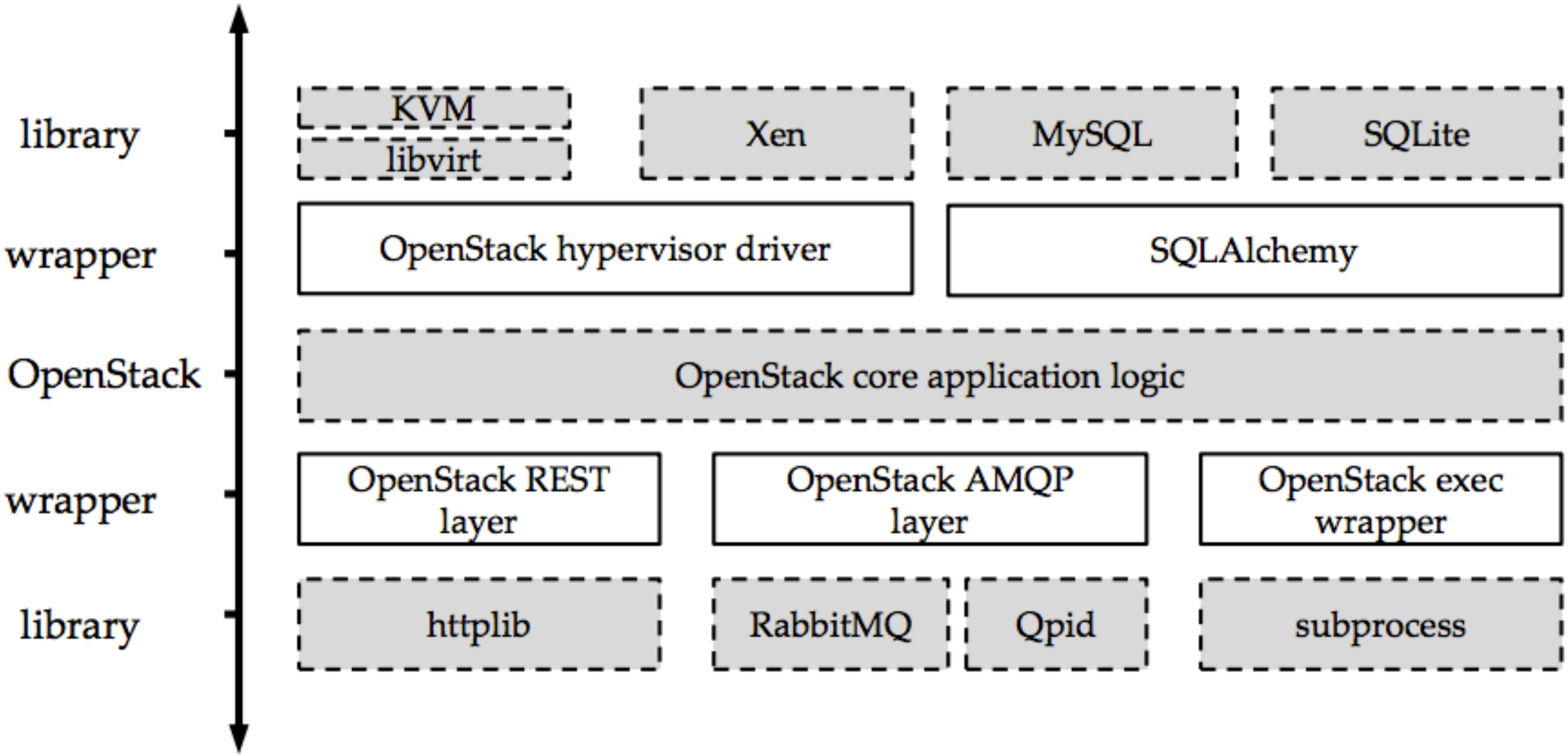


1. Construct Execution Graph (Logging and Coordination Framework)

- System-wide unique tag assigned to each external request
- Introduce tag fields *request context* and *thread-local storage*
- Trace processing of tag within scope of stack



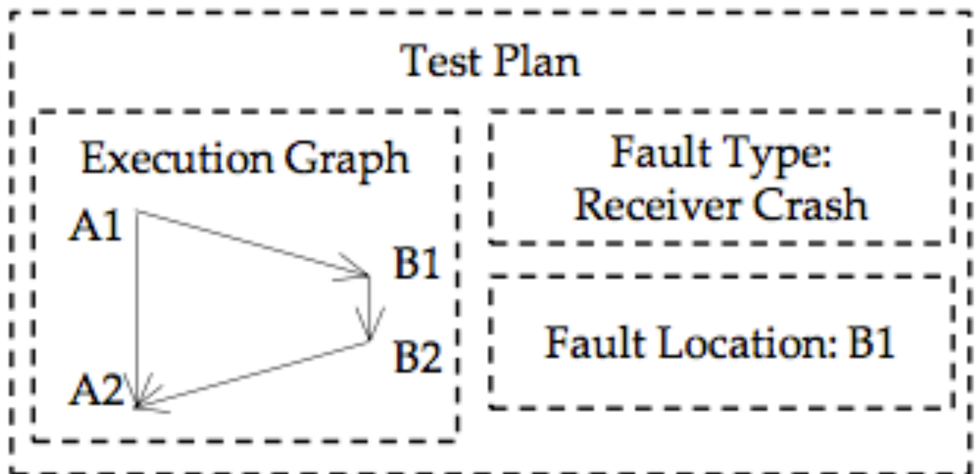
Execution graph for VM_Create



- Layers in white are instrumented with the new tag based logging scheme.
- Consolidate communication between services
- Shared across the stack

2. Fault Injection

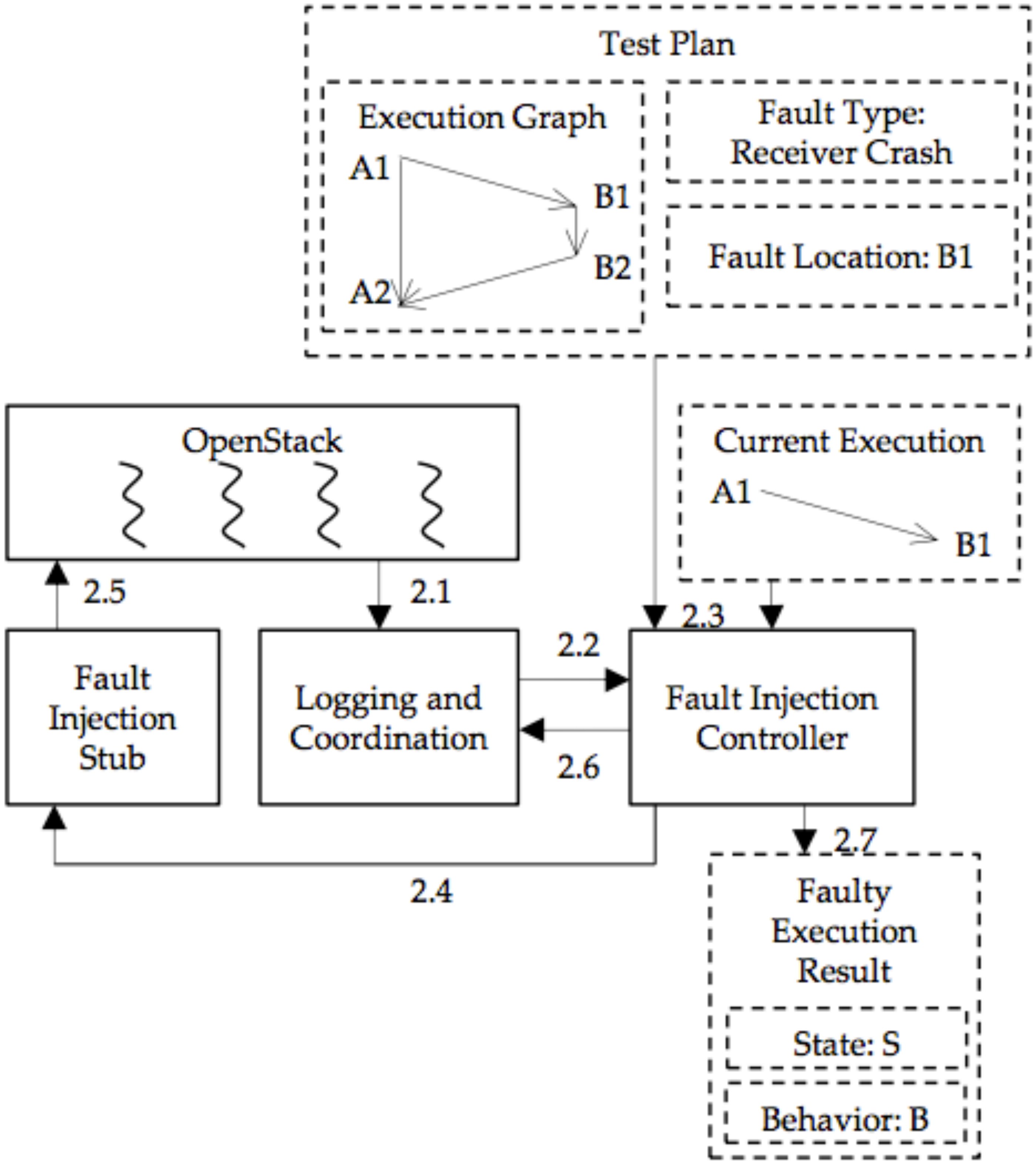
Constructing Test Plans



Procedure 1 Test Plan Generation

test_plans←an empty list
for all *node* in *exe_graph* **do**
  for all *fault* in *fault_specs* **do**
    if *fault* can be injected to *node* **then**
      new_plan←TestPlan(*exe_graph*, *node*, *fault*)
      test_plans.append(*new_plan*)
return *test_plans*

Inject Faults



- Only single faults injected
- Two types of faults considered
    - Service Faults
    - Network Partition Faults
- Suggest clustering nodes on execution graph

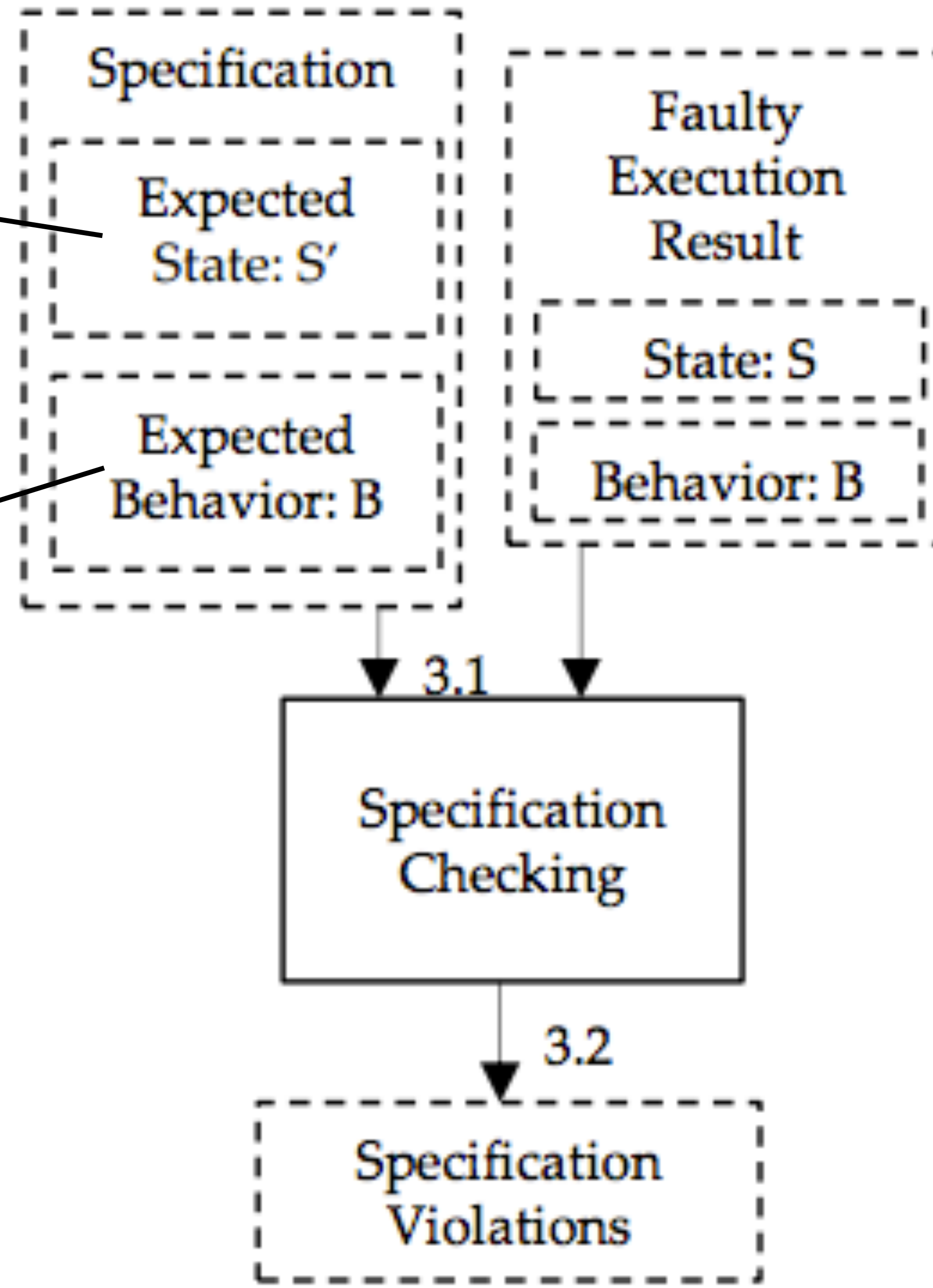
3. Specification Checking

Specification 1 VM State Stabilization Specification

query = select *VM* from *compute_database*
where *VM.state* in *collection*(*VM unstable states*)
if *query*.count() = 0 **then**
return *Pass*
return *Fail*

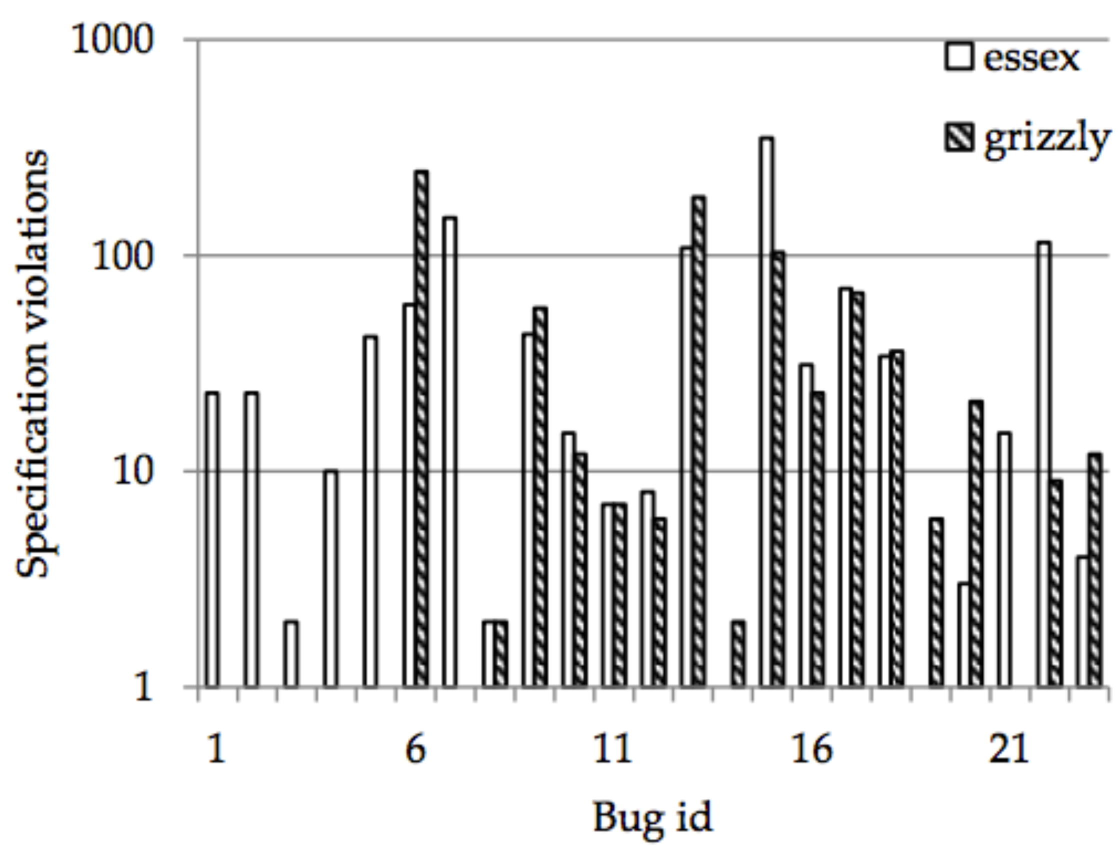
Specification 2 Ethernet Configuration Specification

if (*VM.state* = *ACTIVE*) **and**
((*VM.host.Ethernet* **not** setup) **or**
(*network_controller.Ethernet* **not** setup)) **then**
return *Fail*
return *Pass*



Fault Resilience of OpenStack

Category	Count		
	Common	**essex only**	**grizzly only**
Timeout	1	1	0
Periodic checking	6	4	0
State transition	1	0	1
Return code checking	4	1	0
Cross-layer coordination	0	1	0
Library interference	0	1	0
Miscellaneous	1	1	0
Total	13	9	1



API	Faults				Specification Violations				Bugs			
	essex		grizzly		essex		grizzly		essex		grizzly	
	Crash	Part.	Crash	Part.	Crash	Part.	Crash	Part.	Crash	Part.	Crash	Part.
VM create	217	133	311	229	93	43	150	49	8	6	3	2
VM delete	79	61	102	82	51	15	45	23	9	5	5	2
VM pause	24	17	35	29	16	13	6	4	5	6	2	1
VM reboot	64	36	139	104	9	11	0	5	3	4	0	1
VM rebuild	159	106	242	183	103	67	0	13	5	5	0	1
VM image create (local)	142	119	171	150	59	106	90	79	4	2	3	3
VM image create (HTTP)	107	92	171	150	24	84	79	71	3	2	3	3
VM image delete (local)	59	44	22	15	23	37	12	9	3	2	2	2
VM image delete (HTTP)	59	44	22	15	23	37	10	8	2	2	1	1
Tenant create	7	6	7	6	0	6	0	6	0	1	0	1
User create	7	6	7	6	0	6	0	6	0	1	0	1
Role create	7	6	7	6	0	6	0	6	0	1	0	1
User-role create	9	8	10	9	0	8	0	9	0	1	0	1
Sum	940	678	1246	984	401	439	392	288	42	38	19	20

Types of Errors

- Timeout mechanisms missing in critical OpenStack paths, e.g., REST
  - CreateVM — Compute service querying image service, a network partition blocks system
  - Fixed in grizzly — 600s timeout by default.
- Cross layer coordination — How do you coordinate across layers?
  - OpenStack's AMQP Wrapper waits indefinitely for answer. QPID client drops request after n tries.

- Library Interface —
    - Block compute service — QPID client uses a read/ select which is different from the Python implementation. Hence the client performs read on pipe before it is ready.
- Return Code Checking
    - Incorrect use of return codes
    - Checking disabled

Discussion

- How does the OpenStack configuration affect the fault to failure propagation?
- Test plan creation — What do you do in cases where there are multiple ways faults can be injected?
- Finding the bug is still a manual process
- Does changing system's state make a difference to the result?