oVirt

Monitoring At Scale:

What was Recently Done and What's Next in oVirt

Arik Hadas
Principal Software Engineer
Red Hat
23/10/17

What Do We Mean by “Monitoring”

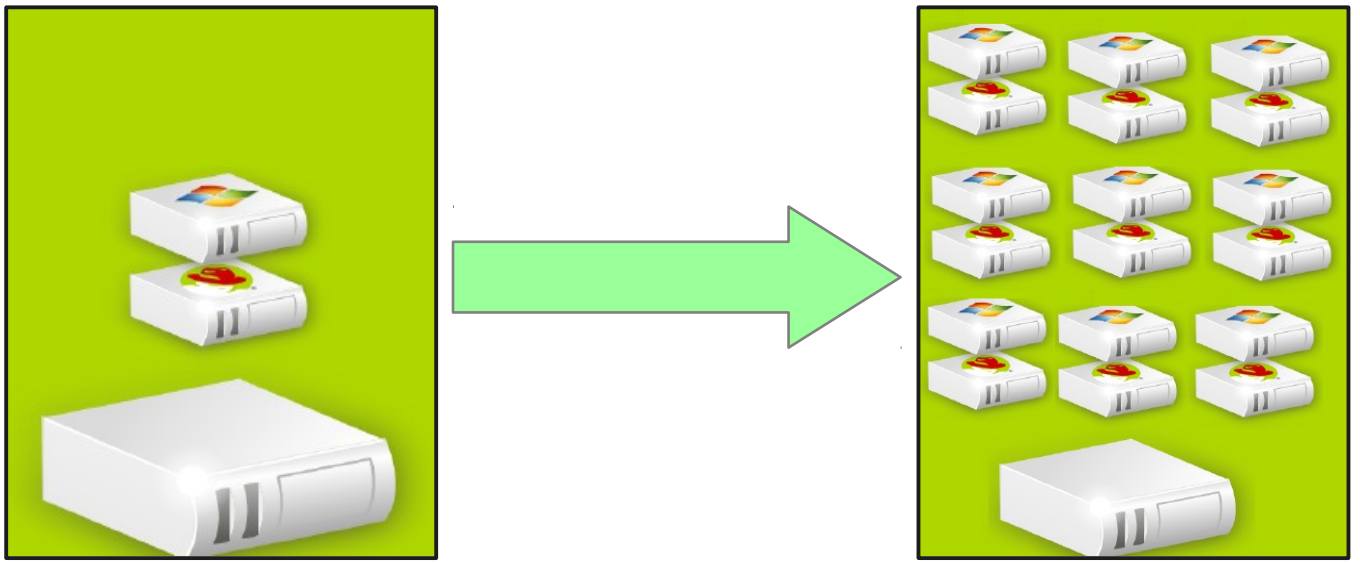
- Identifying the status of active entities
  - VMs, Hosts, Storage domains
- Tracking resource consumption
  - Memory, CPU, Disk space, ...
- Retrieving dynamic properties
  - Client IP, Device addresses, ...

Why is Monitoring Important

- Reflects the up-to-date status of the system
- Affects system responsiveness
- Provides data for automatic processes
  - High availability
  - Load balancing
  - ...

Monitoring At Scale

- The more entities to monitor, the more:
  - Data to collect
  - Data to process
  - Data to store



Problem: Low Performance

- Monitoring
  - Continuous operation
  - Runs in the background
- In large scale deployments monitoring may consume a lot of resources
  - Leads to various anomalies

Our Solution

- Relatively simple changes
  - No architectural change
  - No major change in technology
- We noticed a significant improvement

Outline

- **Introduction to oVirt**
- VMs monitoring in large scale deployments
- Improving the monitoring process
- Measurements
- Future work

What Is oVirt?

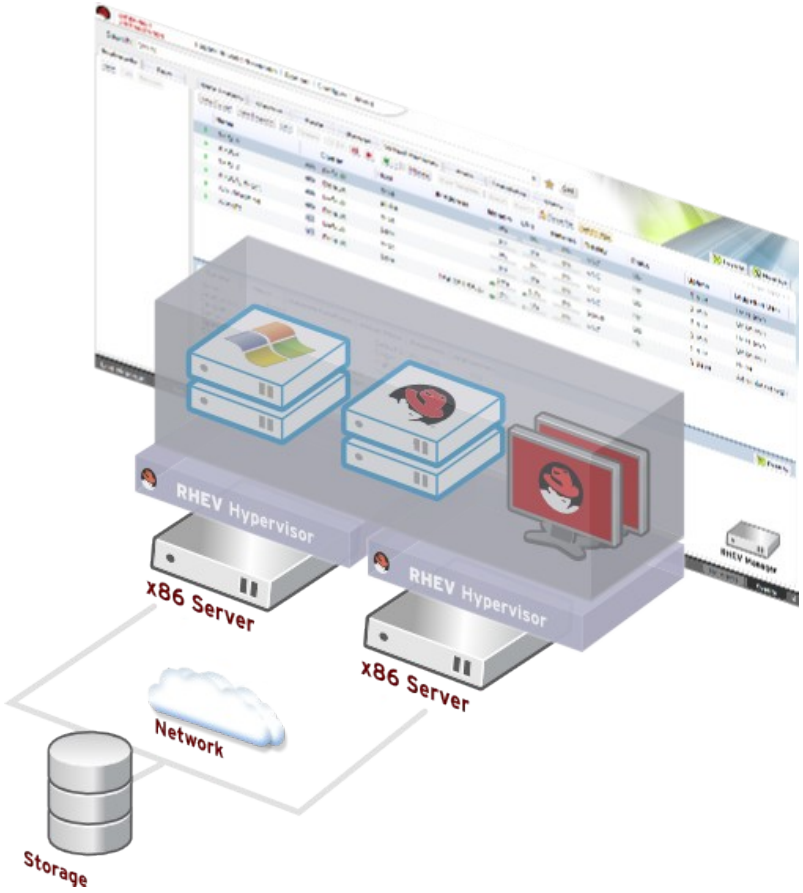
Large scale, centralized management for server and desktop virtualization

Based on leading performance, scalability and security infrastructure technologies

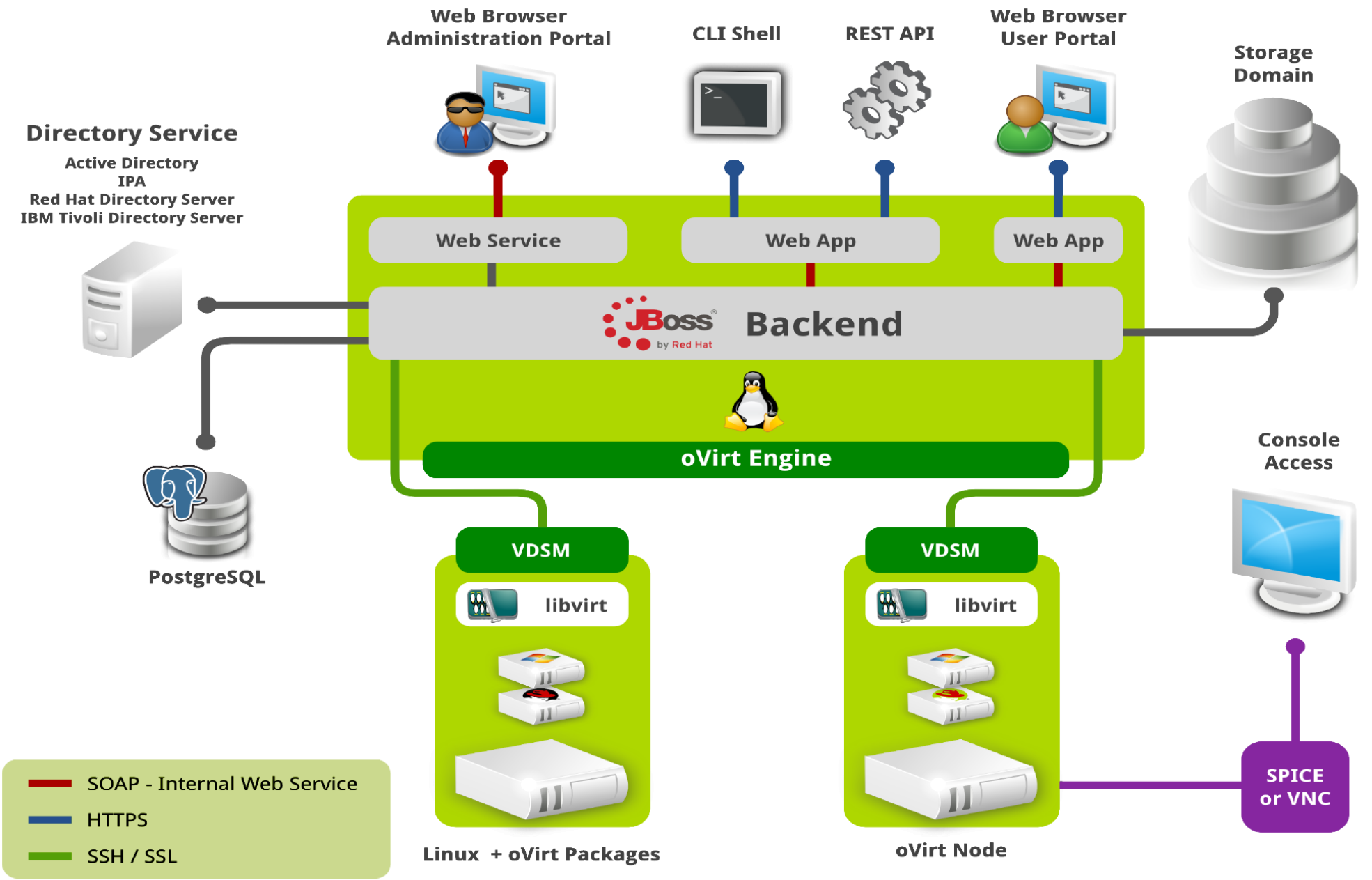
Provide an open source alternative to vCenter/vSphere

Focus on KVM for best integration/performance

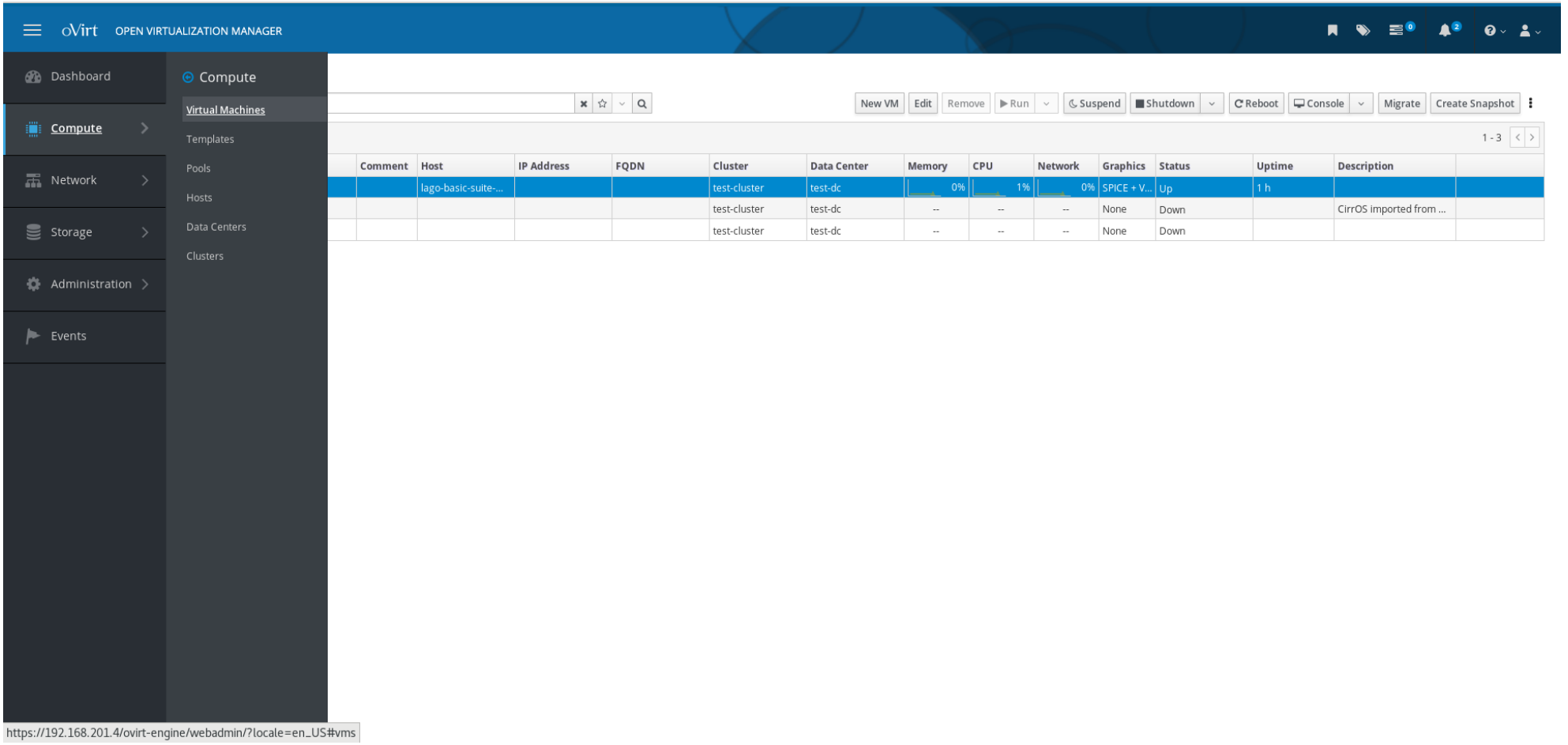
Focus on ease of use/deployment



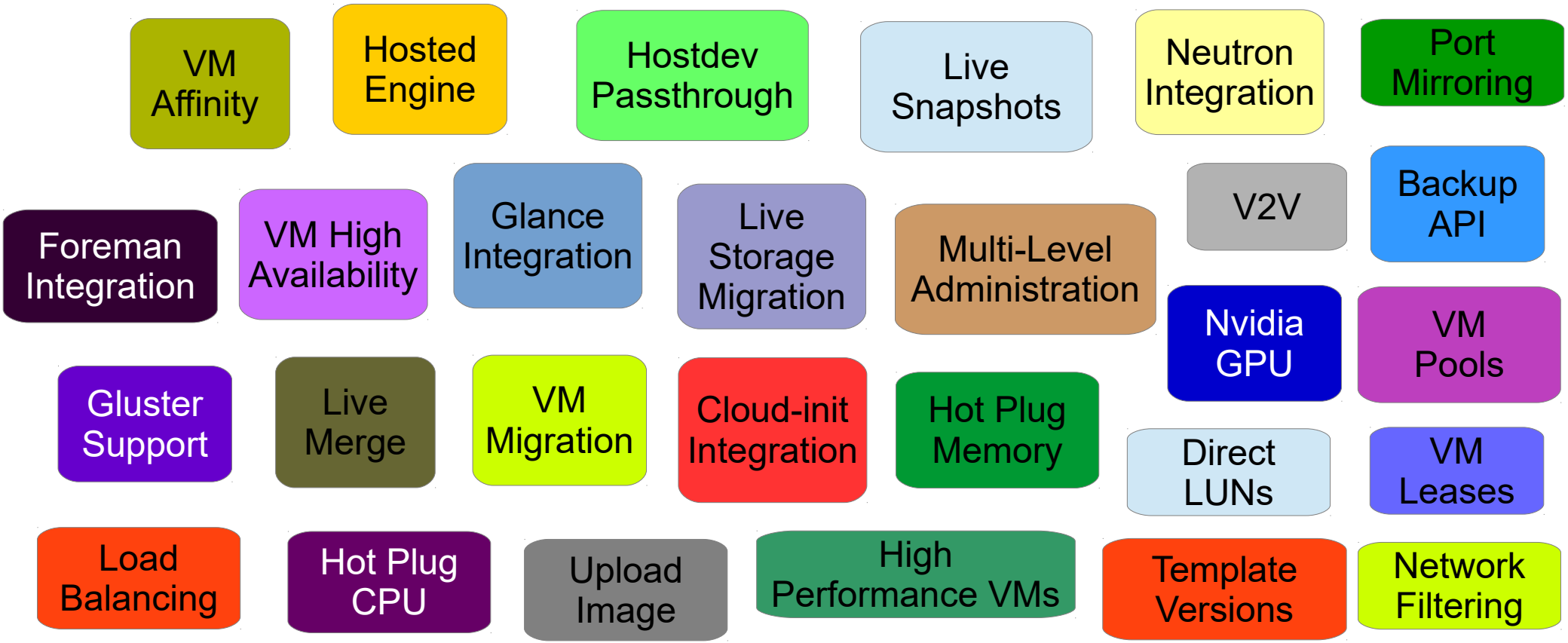
oVirt Architecture View



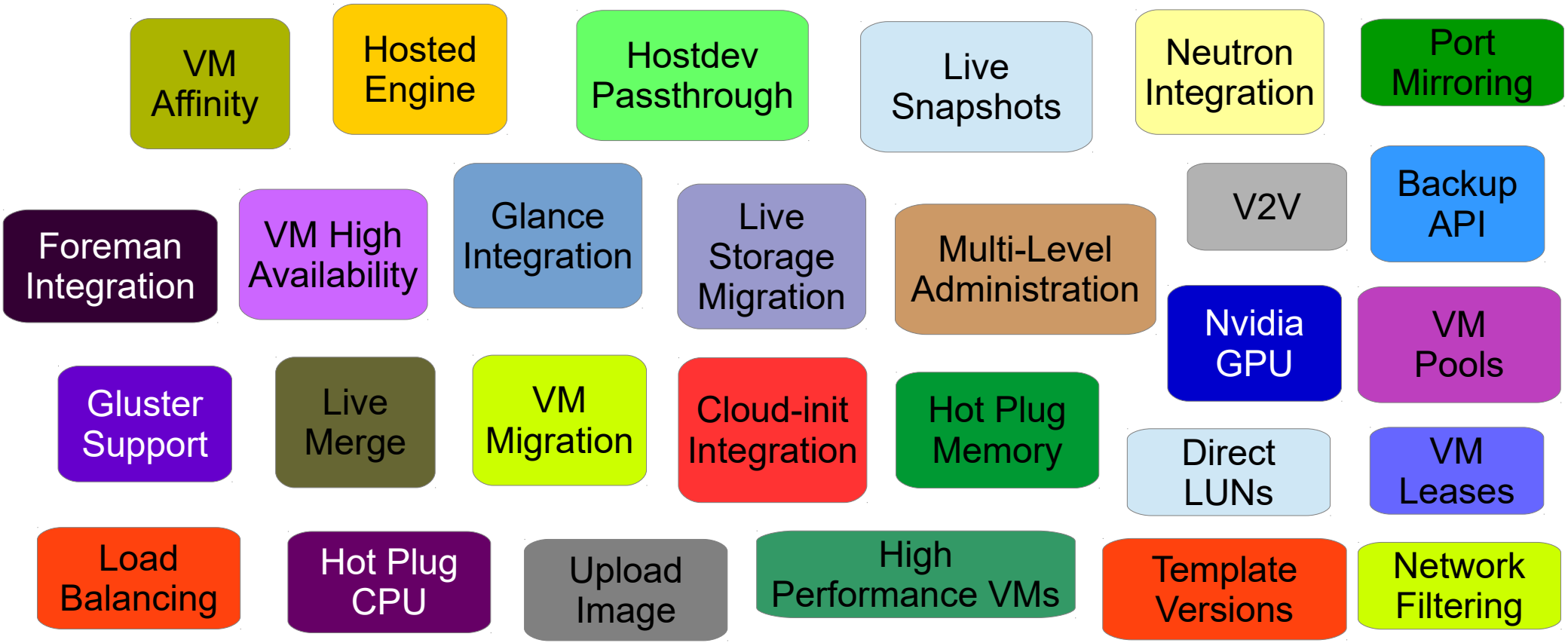
Webadmin - Screenshot



oVirt Feature-Rich Platform



Feature-Rich Platform



- Less attention to scale

Outline

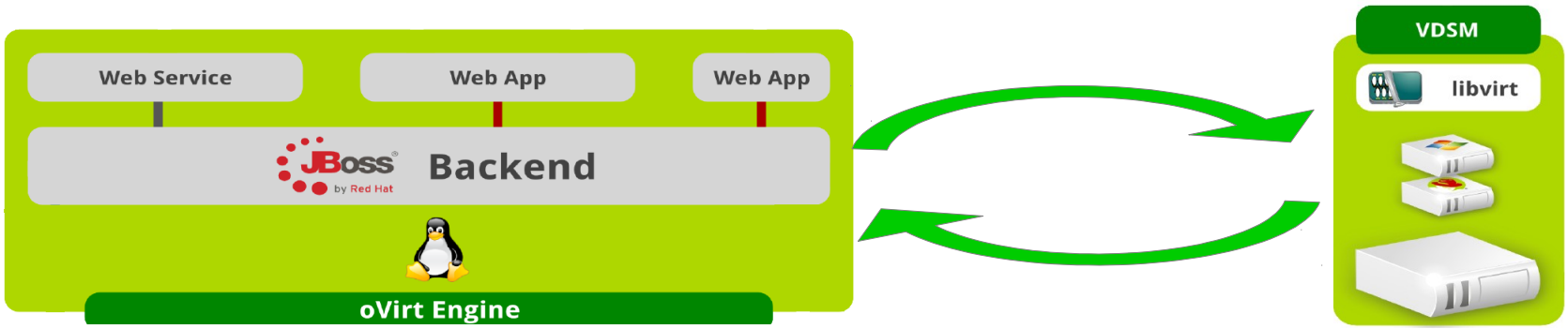
- Introduction to oVirt
- **VMs monitoring in large scale deployments**
- Improving the monitoring process
- Measurements
- Future work

VMs Monitoring

- Focus on monitoring of virtual machines
  - Far more instances than any other entity
- This includes:
  - Status
  - Dynamic properties (i.e., client IP)
  - Devices information
  - Statistics

VMs Monitoring Model Before v3.6

- Polling based mechanism
- Every 3 sec, for each host:
  - The engine queries VMs from the database
  - The engine polls information on running VMs
  - The engine persists data that has changed
- Every 5th cycle includes statistics



VMs Monitoring Model Before v3.6 (2)

- Hosts are locked during monitoring cycles
  - To prevent operations on VMs in parallel
- Dynamic properties are compared via reflection
- VM statistics are not being compared
  - They almost always change
- Devices are polled separately when their hash changes

VMs Monitoring Model Before v3.6 (3)

- Problems in very large scale deployments
  - Monitoring cycles were skipped
  - High CPU consumption
  - High load on the database

Proposed Solutions

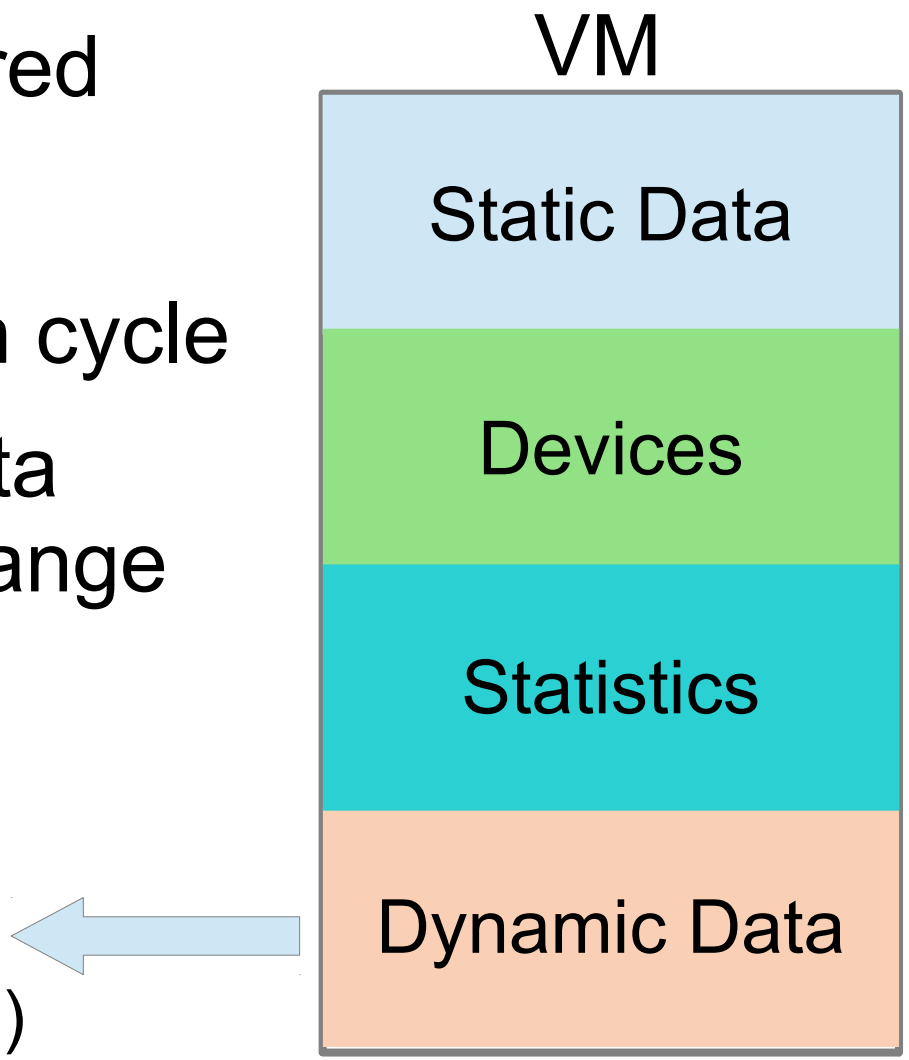
- Add a global caching layer
  - To reduce interactions with the database
  - Does not solve the high CPU consumption
- Distribute the monitoring process
  - Addresses the high CPU consumption
  - Does not reduce the load on the database
- Both solutions were too complex

Outline

- Introduction to oVirt
- VMs monitoring in large scale deployments
- **Improving the monitoring process**
- Measurements
- Future work

Issue #1: Too Many Writes to DB

- Static data is not monitored
- Devices rarely change
- Statistics change in each cycle
- Some of the dynamic data (reported data) might change
  - Not often though



Reported data (i.e., client IP) +
Not reported data (i.e., stop reason)

Reduce Number of Writes

- Introduce @UnchangeableByVdsm
  - Marks properties that are not reported

```
private String currentCd;
@UnchangeableByVdsm
private String stopReason;
private VmExitReason exitReason;
```

- Move frequently changed fields to the stats
  - E.g., guest memory cached/buffered/free

Separate Out Devices Monitoring

- Devices hash was stored with the dynamic data
  - Consequently, change of one device triggered persistency of all dynamic data
- Solution: store the devices hash separately

Issue #2: Too Many Reads from DB

- Many connections with DB are used
- Long time is spent on quering the DB
- Even when no data (except stats) is changed!

Eliminate Redundant Queries

- Optimize the code to skip unneeded data processing (including queries from DB)
- For example, skipping redundant VM numa nodes processing eliminated the following DB interactions:

Average time (micro-sec)

- 261 to get numa nodes by host
- 259 to get assigned numa nodes
- 255 to get numa node CPU by host
- 246 to get numa node CPU by VM
- 242 to get numa nodes by VM

Overall time (micro-sec)

- Getting numa nodes by host—3% (48,546 msec)
- Getting assigned numa nodes—3% (48,201 msec)
- Getting numa node CPU by host—3% (47,569 msec)
- Getting numa node CPU by VM—2% (45,918 msec)
- Getting numa nodes by VM—2% (45,041 msec)

Memorization

- Apply memoization to repeated queries

```
public class MemoizingSupplier<T> implements Supplier<T> {
    private final Supplier<T> delegate;
    private boolean initialized;
    private T value;

    public MemoizingSupplier(Supplier<T> delegate) {
      this.delegate = delegate;
    }

    public T get() {
      if (!initialized) {
        value = delegate.get();
        initialized = true;
      }
      return value;
    }
}
```

Smart Caching

- Cache only relevant entity's properties
  - E.g., static properties used by the monitoring
- Cache only relevant entities
  - E.g., VM jobs (limited number of instances)
- Use DB for persistency, not as a bus of data
  - E.g., VM statistics

Lighter, Dedicated Queries

- Complicated queries take time
- Attempt #1: narrow down ‘vms’ view

> explain analyze select * from vms where ...
Planning time: 2.947 ms
Execution time: 765.774 ms

> explain analyze select * from vms_monitoring_view where ...
Planning time: 0.387 ms
Execution time: 275.600 ms

Lighter, Dedicated Queries (2)

- Attempt #2: query only dynamic data

> explain analyze select * from vms_monitoring_view where ...
Planning time: 0.405 ms
Execution time: 275.850 ms

> explain analyze select * from vm_dynamic where ...
Planning time: 0.109 ms
Execution time: 2.703 ms

Issue #3: Locks Contention

- High contention between monitoring threads and those executing operations on VMs
- During the execution of VM operations, the host was locked to avoid monitoring the VM
  - To prevent conflicts

Split VMs and Hosts monitoring

- Replaced host-level locks with VM-level locks
  - VM operations lock VMs rather than hosts
  - Monitoring locks each VM running on the host
    - And skips those that cannot be locked
- That reduces contention rate on operations-intensive deployments

Issue #4: High UNIX Load

- The overall backend load was relatively high
  - Even in stable deployment
- The monitoring was an immediate suspect

Events-Based Communication

- Replaced the polling-based backend<->host protocol with events-based protocol
  - Based on JSON-RPC instead of XML-RPC
- Hosts send events upon VM changes
  - Less monitoring cycles and data to process
- Keep polling statistics cycles
  - Statistics always change
  - Compensate missing events

Outline

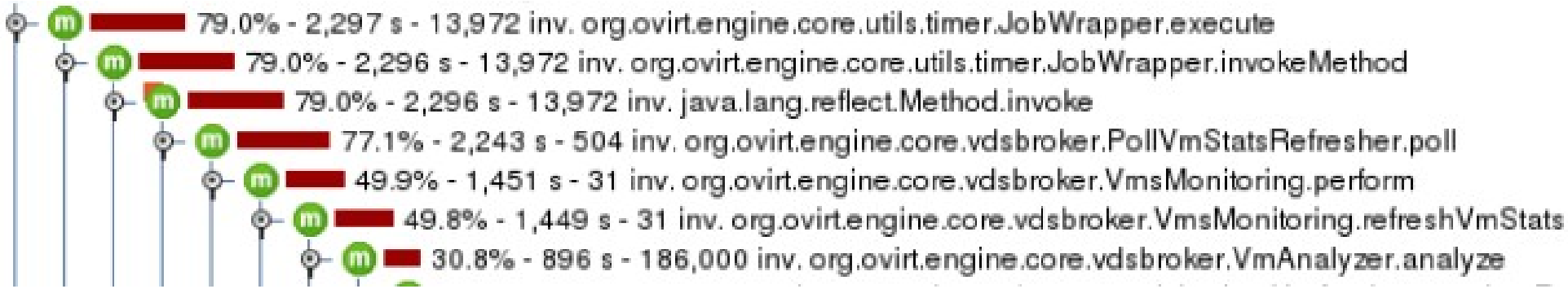
- Introduction to oVirt
- VMs monitoring in large scale deployments
- Improving the monitoring process
- **Measurements**
- Future work

Case Study

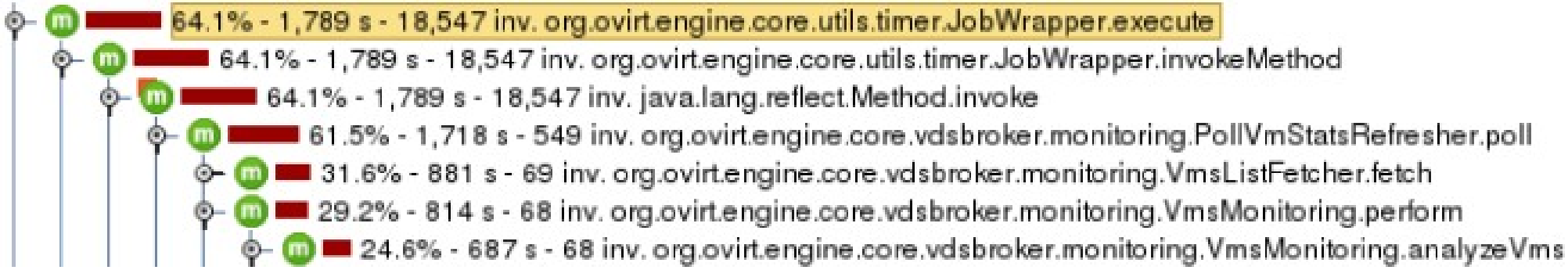
- Deployment with 1 host running 6000 VMs
  - ‘Fake VMs’
- Stable deployment
  - No operation is done
- Measured 1 hour of uptime
- Compared versions 3.6 and 4.1
  - Both used events

CPU

3.6


79.0% - 2,297 s - 13,972 inv. org.ovirt.engine.core.utils.timer.JobWrapper.execute
79.0% - 2,296 s - 13,972 inv. org.ovirt.engine.core.utils.timer.JobWrapper.invokeMethod
79.0% - 2,296 s - 13,972 inv. java.lang.reflect.Method.invoke
77.1% - 2,243 s - 504 inv. org.ovirt.engine.core.vdsbroker.PollVmStatsRefresher.poll
49.9% - 1,451 s - 31 inv. org.ovirt.engine.core.vdsbroker.VmsMonitoring.perform
49.8% - 1,449 s - 31 inv. org.ovirt.engine.core.vdsbroker.VmsMonitoring.refreshVmStats
30.8% - 896 s - 186,000 inv. org.ovirt.engine.core.vdsbroker.VmAnalyzer.analyze

4.1

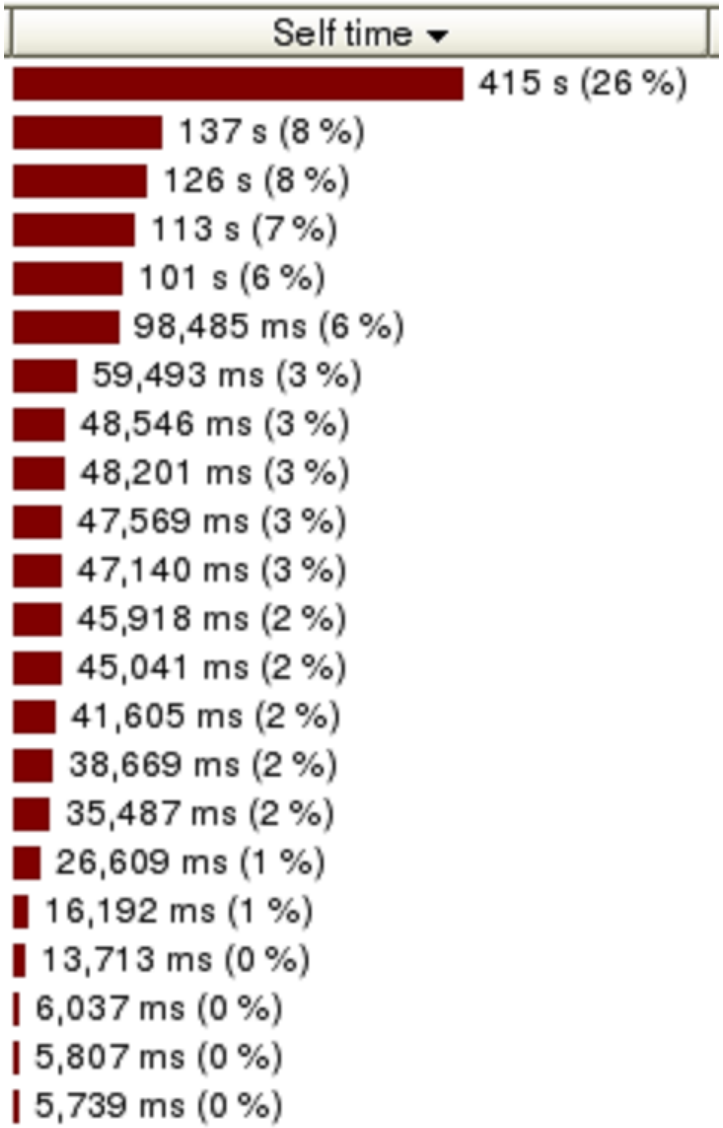

64.1% - 1,789 s - 18,547 inv. org.ovirt.engine.core.utils.timer.JobWrapper.execute
64.1% - 1,789 s - 18,547 inv. org.ovirt.engine.core.utils.timer.JobWrapper.invokeMethod
64.1% - 1,789 s - 18,547 inv. java.lang.reflect.Method.invoke
61.5% - 1,718 s - 549 inv. org.ovirt.engine.core.vdsbroker.monitoring.PollVmStatsRefresher.poll
31.6% - 881 s - 69 inv. org.ovirt.engine.core.vdsbroker.monitoring.VmsListFetcher.fetch
29.2% - 814 s - 68 inv. org.ovirt.engine.core.vdsbroker.monitoring.VmsMonitoring.perform
24.6% - 687 s - 68 inv. org.ovirt.engine.core.vdsbroker.monitoring.VmsMonitoring.analyzeVms

CPU (2)

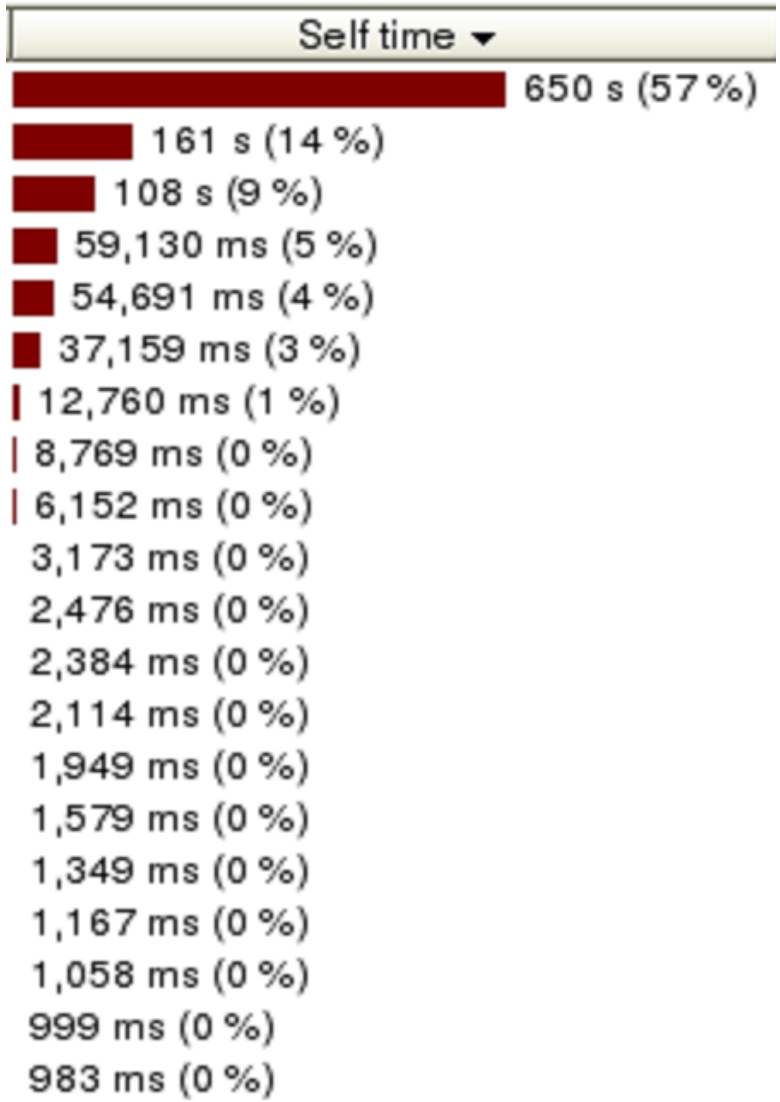
- Total CPU time reduced from 2297s to 1789s (78%)
- Significantly less time in monitoring code
  - Processing time reduced from 896s to 687s
  - Persistence time reduced from 546s to 114s
  - Overall, 814s instead of 1451s (56%)

Database – Hot Spots

3.6

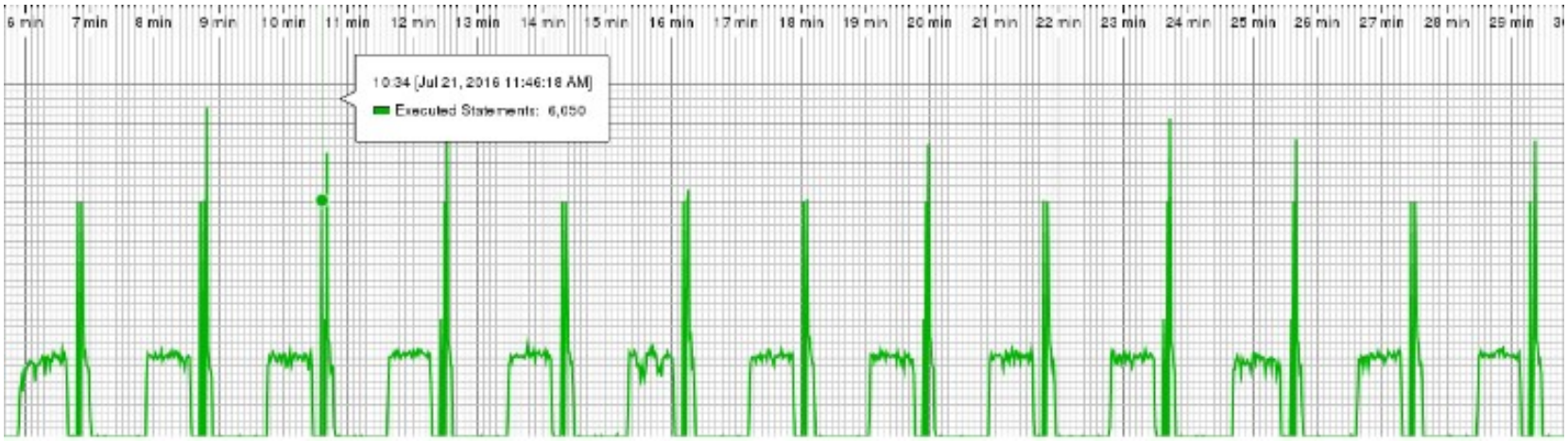


4.1



Database – Executed Statements

3.6



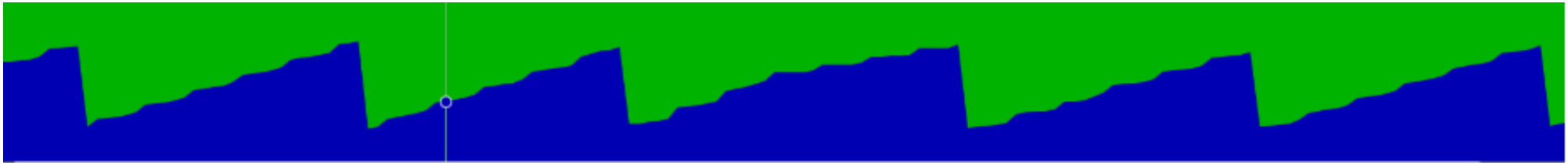
4.1

Database

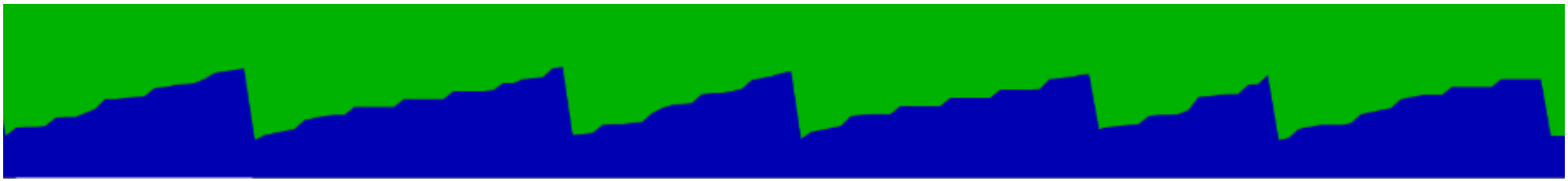
- The time to query all VMs reduced from 3539ms to 909msec (26%)
- The time to save dynamic data in 3.6 was 101 sec (6%, 544 micro-sec on average), 0 in 4.1
  - Similar results for other properties
- In overall, less use of the database

Memory Consumption

3.6



4.1



- Surprisingly, less memory was consumed in 4.1
  - In 3.6 it gets to ~1.45GB
  - In 4.1 it gets to ~1.2GB
- Probably because of caching done by postgres

Outline

- Introduction to oVirt
- VMs monitoring in large scale deployments
- Improving the monitoring process
- Measurements
- **Future work**

Future Work

- Separate out statistics monitoring
- Apply similar principles to host monitoring
- Add caching of more entities
  - Specifically, VM dynamic data (e.g., status)

Conclusions

- Significant improvement shown in a case study
  - All changes are available in version 4.1
- This required deep knowledge of the platform
  - No shortcuts in the form of generic solutions
  - No major technological change
  - No architectural change

THANK YOU!

http://www.ovirt.org
ahadas@redhat.com
ahadas@irc.oftc.net#ovirt