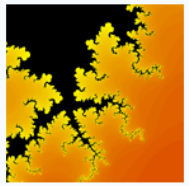
KERLABS

Run-Time Configurable Scheduler Framework

Louis.Rilling@kerlabs.com
Marko Novak
Marko.Novak@xlab.si

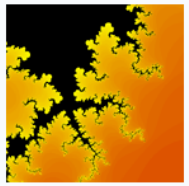
Kerrighed
Linux clusters made easy

Outline



- Rationale
- Big picture
- Example
- User-level interface
- Kernel-level API (samples)
- Status

Outline



- Rationale
- Big picture
- Example
- User-level interface
- Kernel-level API (samples)
- Status

Definitions

- A (global) scheduler runs given applications at given times on given nodes according to a scheduling policy
- A scheduling policy decides when and where applications should run
- Example of policies
  - CPU load balancing
  - Swap avoidance
  - Memory affinity
  - Disk IO balancing
  - Scratch space balancing
  - ...

Needs

- Different applications need different scheduling policies
  - Bag of tasks: CPU load balancer
  - MPI: handle separately compute processes and control processes
  - Parallel make: balance compiler invocations
  - ...

  ➔Admin can restrict a scheduler to some processes

  ➔Help building schedulers with re-usable components
- For a same application the policy can change over the time
  - Cluster changing from shared mode to partitioned mode
  - ...

  ➔At run-time, modify a scheduling policy

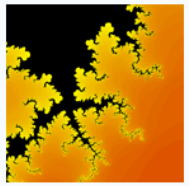
  ➔At run-time, replace a scheduling policy

Needs (2)

- Elements of policies are common to all applications
  - Node removal for maintenance
  - Failure prevention
  - ...

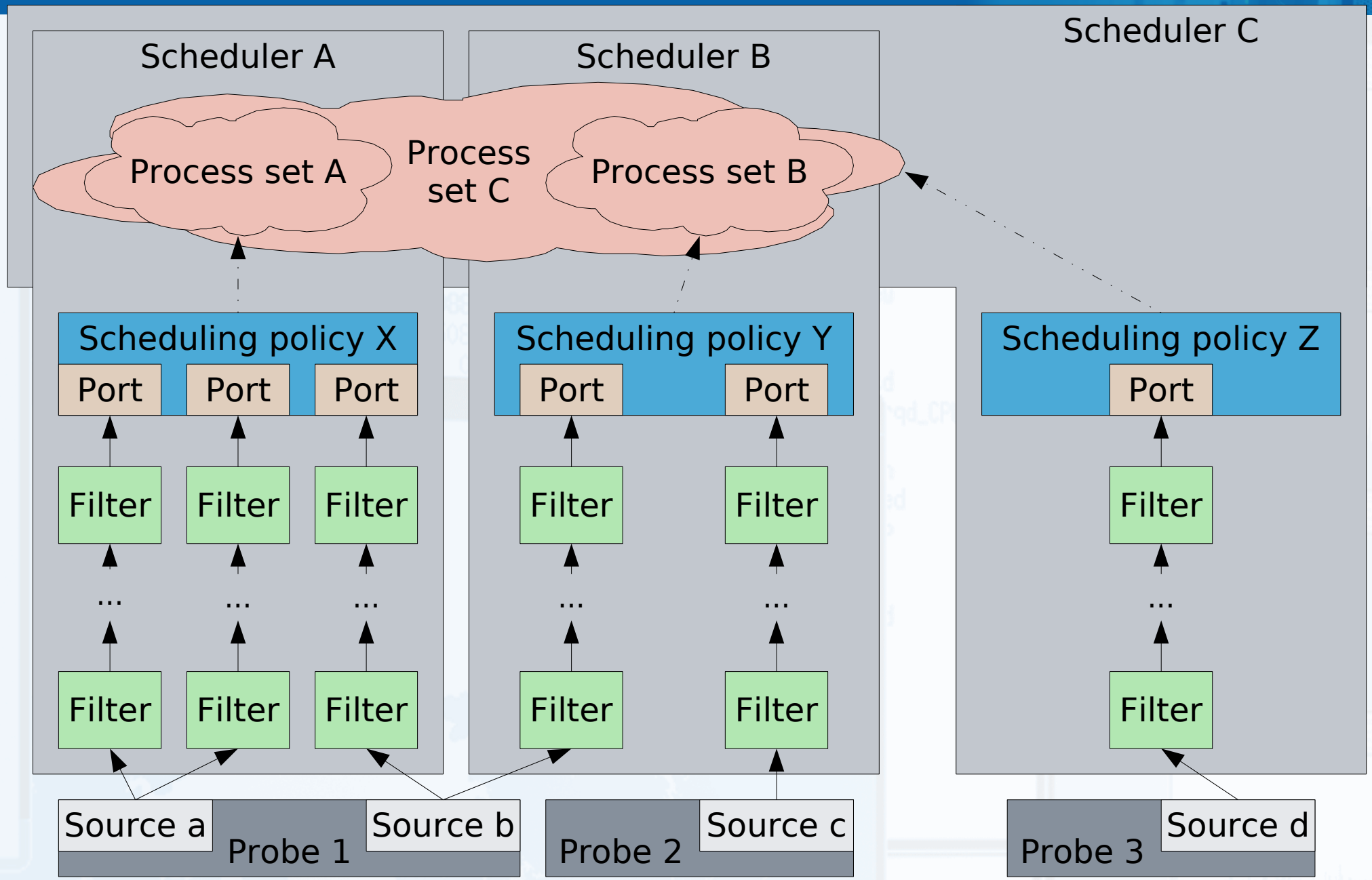
  ➔Stackable policies

Outline



- Rationale
- Big picture
- Example
- User-level interface
- Kernel-level API (samples)
- Status

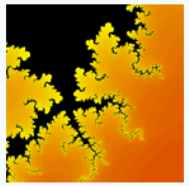
Components layout replicated on all nodes



Features

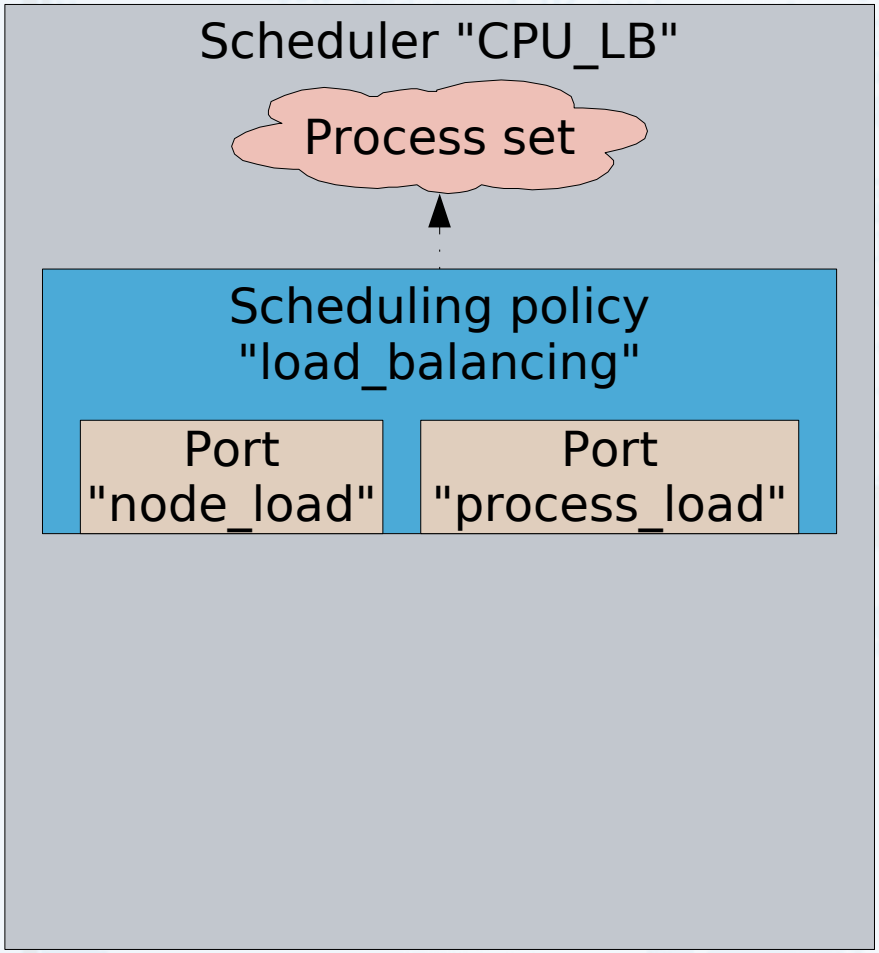
- User-level interface
  - ConfigFS-based configuration and access to data
  - Global configuration from any node
- Kernel-level interface
  - Update notifications (publish-subscribe)
  - Typed data
  - Remote data queries
  - Type-checked component linkage
  - Process set iterators
  - New task placement policies (remote clone)
  - Use existing EPM API (migration, checkpoint)
- Automatic module loading

Outline



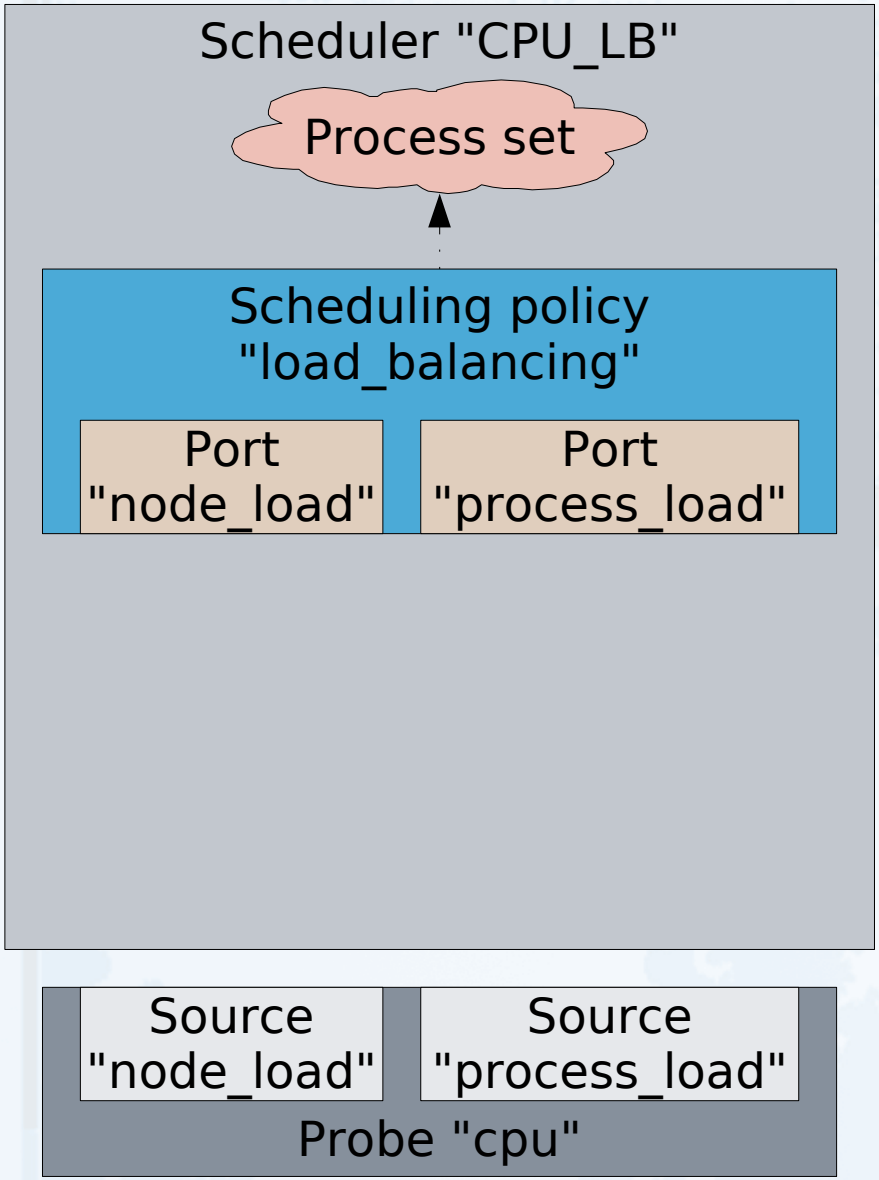
- Rationale
- Big picture
- Example
- User-level interface
- Kernel-level API (samples)
- Status

CPU load balancing



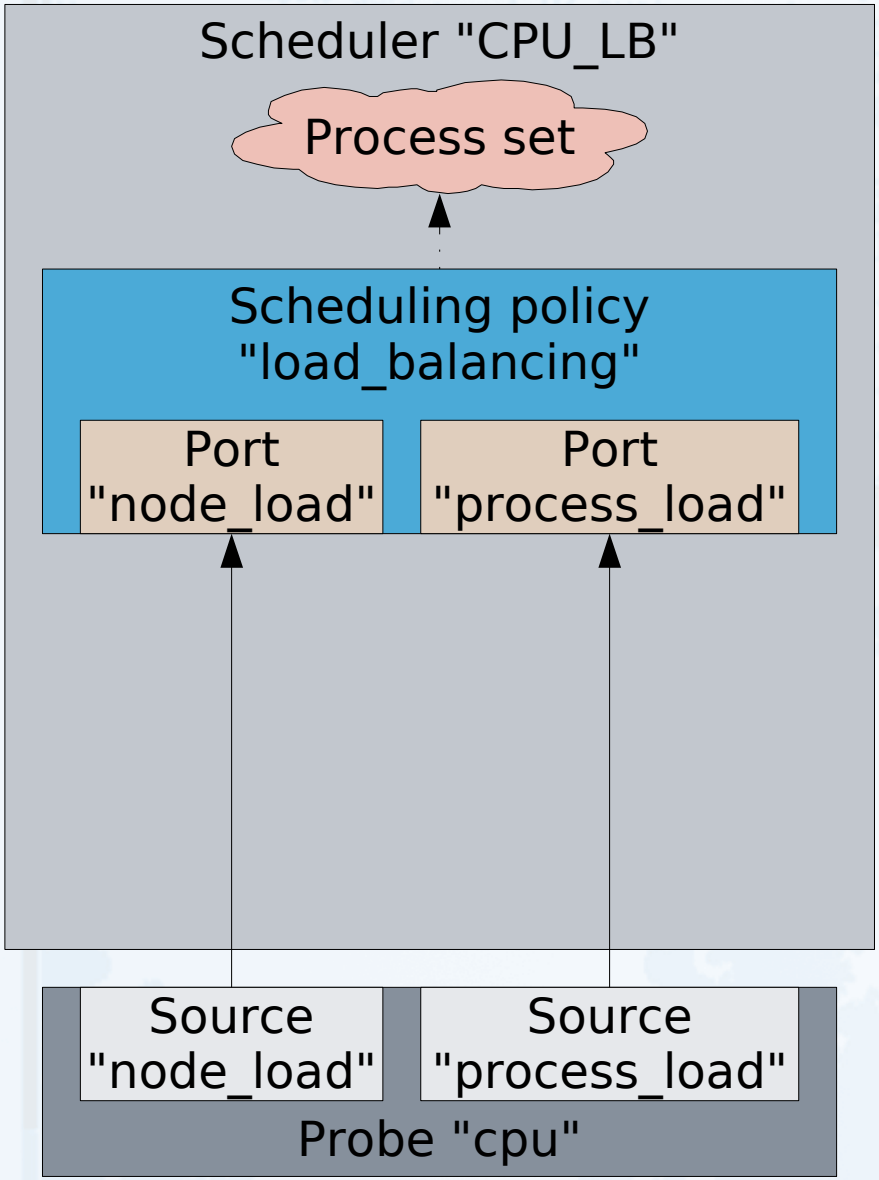
- *Scheduling policy* migrates processes to balance some load
  - *Port* **`node_load`** collects loads (local and remote)
  - *Port* **`process_load`** collects loads induced by processes

CPU load balancing



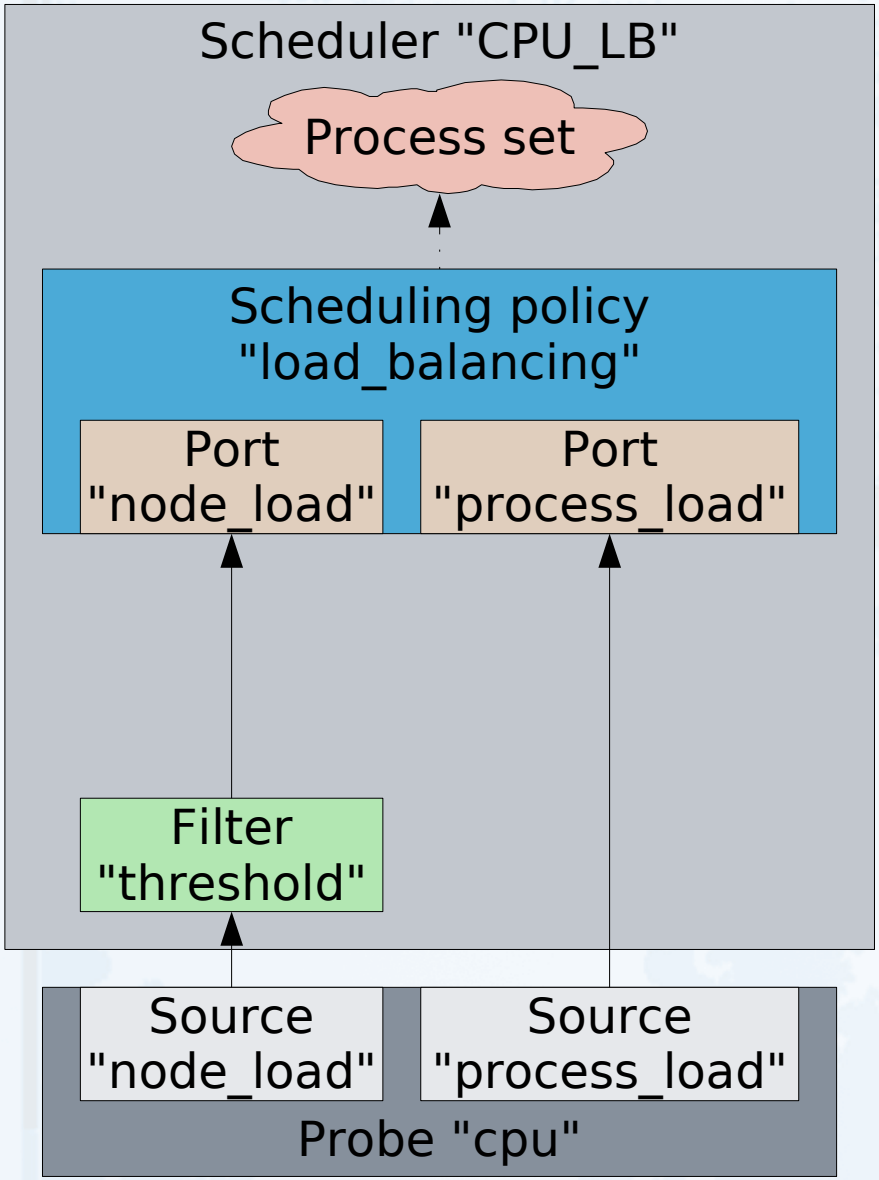
- *Scheduling policy* migrates processes to balance some load
  - *Port* **node_load** collects loads (local and remote)
  - *Port* **process_load** collects loads induced by processes
- *Probe* **cpu** provides local **CPU** loads
- *Probe source* **node_load** notifies updates periodically

CPU load balancing



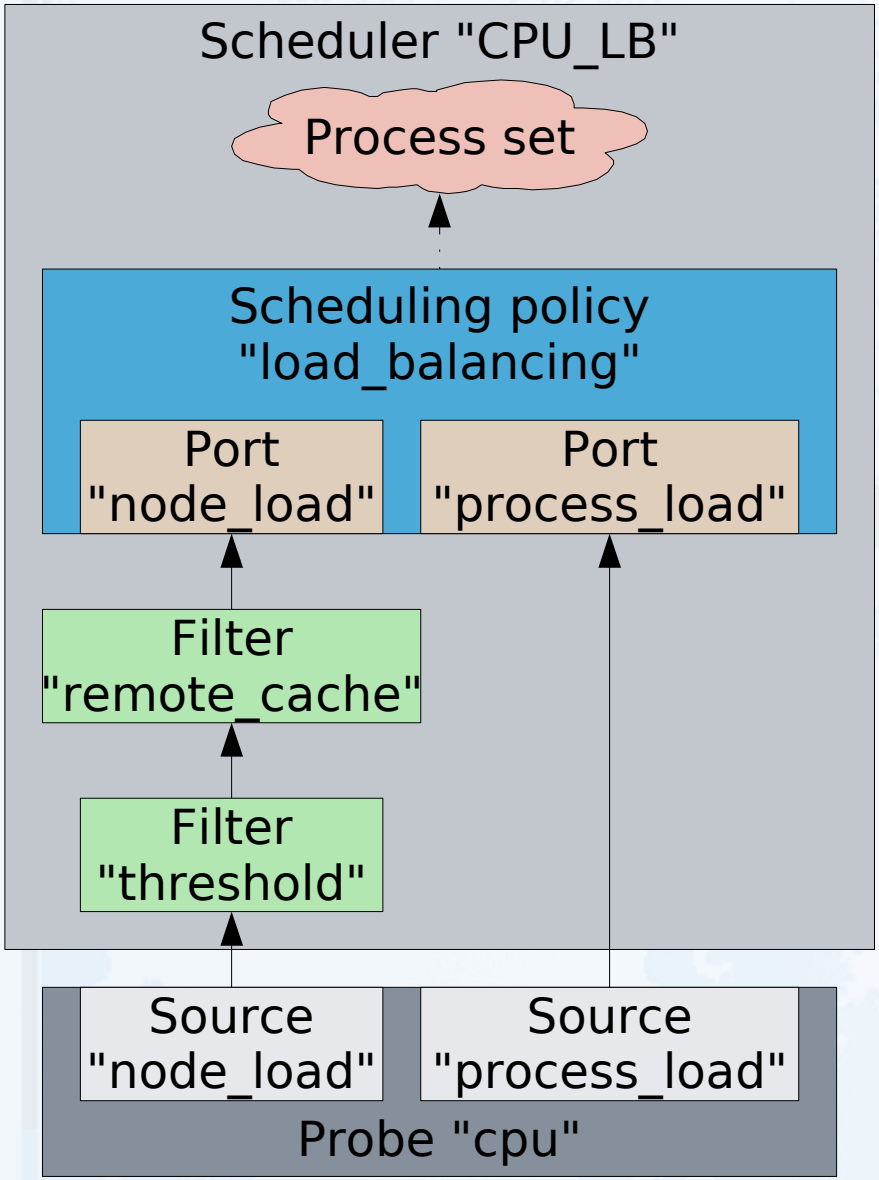
- *Scheduling policy* migrates processes to balance some load
  - *Port* **node_load** collects loads (local and remote)
  - *Port* **process_load** collects loads induced by processes
- *Probe* **cpu** provides local **CPU** loads
- *Probe source* **node_load** notifies updates periodically
- **load_balancing** is activated on local load updates

CPU load balancing



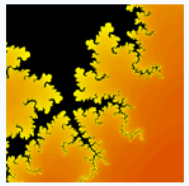
- *Scheduling policy* migrates processes to balance some load
  - *Port* **node_load** collects loads (local and remote)
  - *Port* **process_load** collects loads induced by processes
- *Probe* **cpu** provides local **CPU** loads
- *Probe source* **node_load** notifies updates periodically
- **load_balancing** is activated on local load updates
- Updates for loads > threshold are propagated up to **load_balancing**

CPU load balancing



- *Scheduling policy* migrates processes to balance some load
  - *Port* **node_load** collects loads (local and remote)
  - *Port* **process_load** collects loads induced by processes
- *Probe* **cpu** provides local **CPU** loads
- *Probe source* **node_load** notifies updates periodically
- **load_balancing** is activated on local load updates
- Updates for loads > threshold are propagated up to **load_balancing**
- Remote loads are prefetched and cached

Outline



- Rationale
- Big picture
- Example
- User-level interface
- Kernel-level API (samples)
- Status

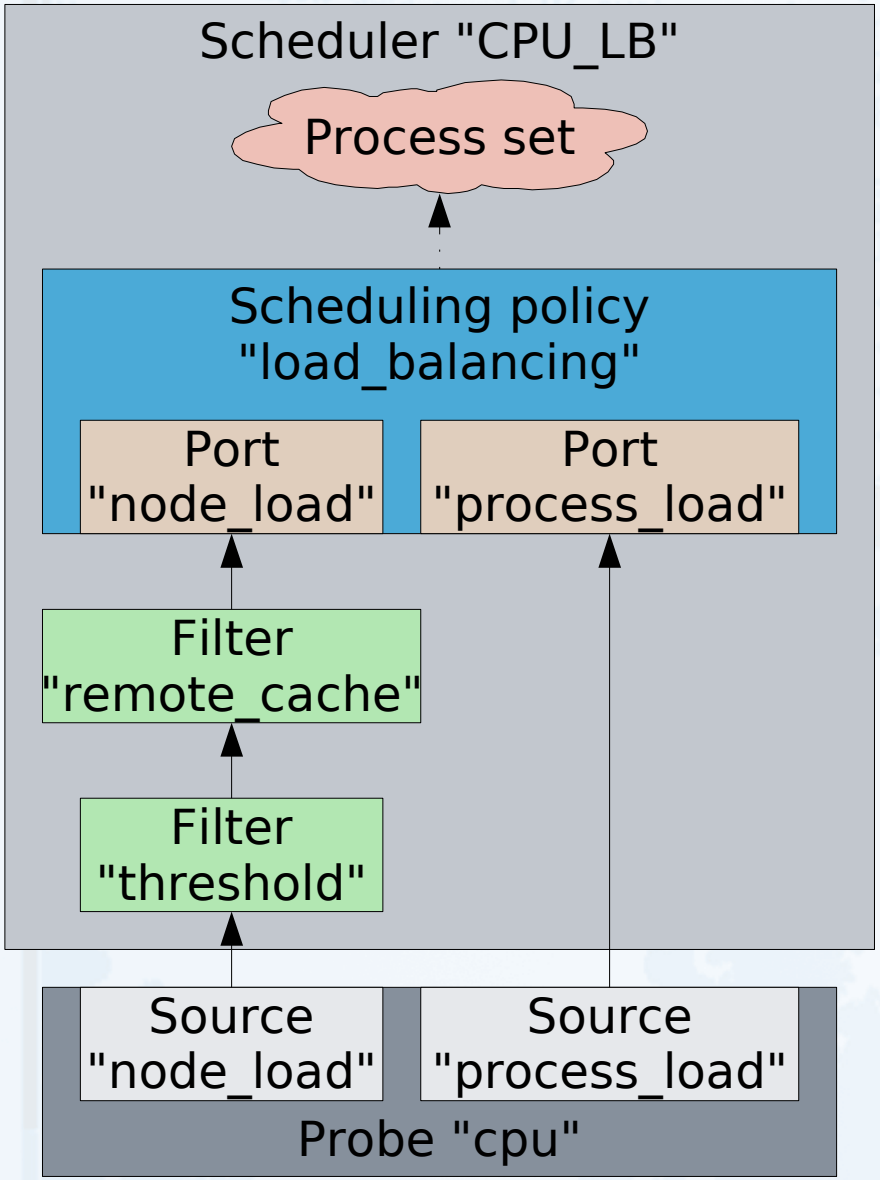
Dynamic Configuration

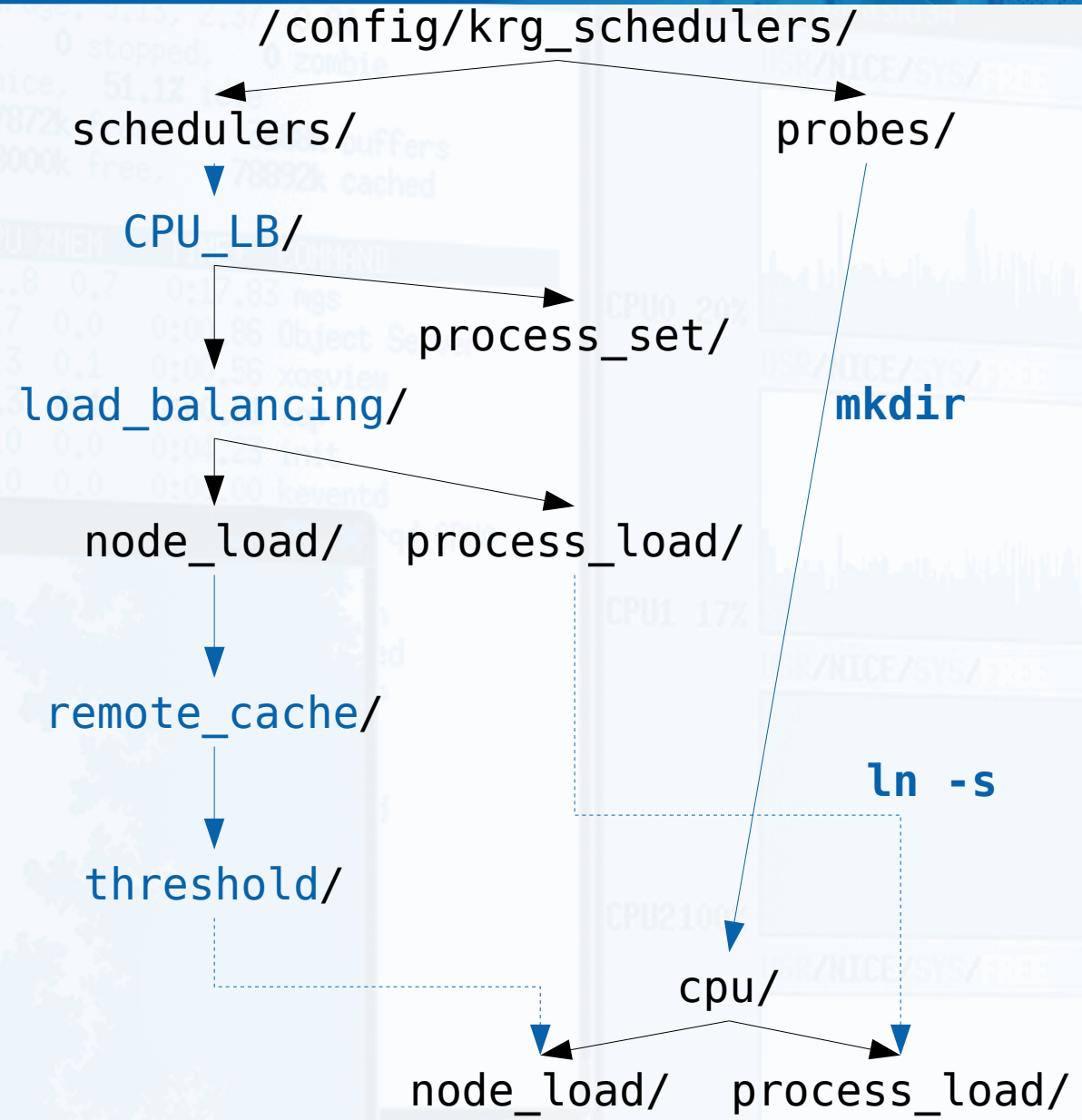
- Configfs
  - Quoting Linux documentation:

    "configfs is a ram-based filesystem that provides the converse of sysfs's functionality. Where sysfs is a filesystem-based view of kernel objects, configfs is a filesystem-based manager of kernel objects, or `config_items`. "

  - **`mkdir`** -> create a `config_item`
  - **`read`**/**`write`** -> see/set `config_item` attributes
  - **`symlink`** -> aggregate `config_items` from different subtrees
- Map scheduler component creations and connections to configfs operations

Configfs hierarchy

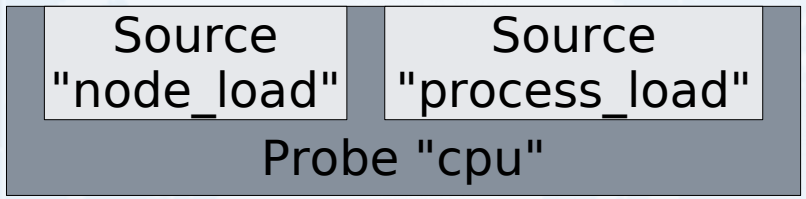




Setting up a scheduler

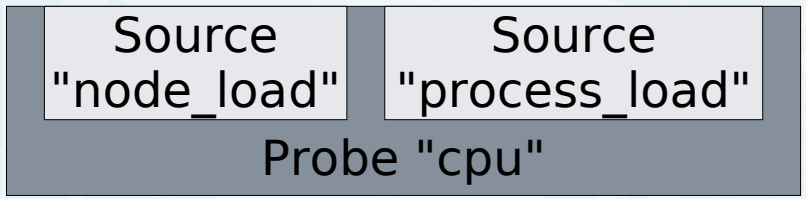
Setting up a scheduler

```
# SROOT=/config/krg_schedulers
# PROBE=$SROOT/probes/cpu
# mkdir $PROBE
     [loads cpu.ko and] activates the probe
```

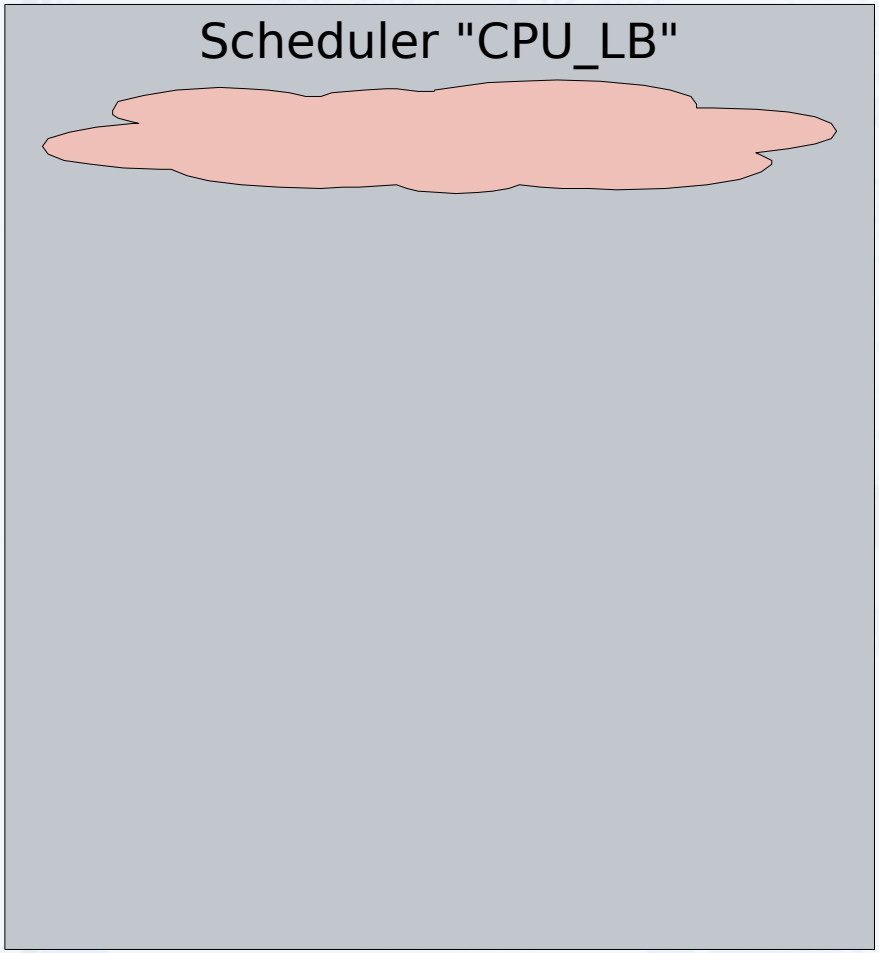


Setting up a scheduler

```
# SROOT=/config/krg_schedulers
# PROBE=$SROOT/probes/cpu
# mkdir $PROBE
     [loads cpu.ko and] activates the probe
# echo 1000 > $PROBE/probe_period
                make the probe notify updates
                                every 1000 jiffies
```

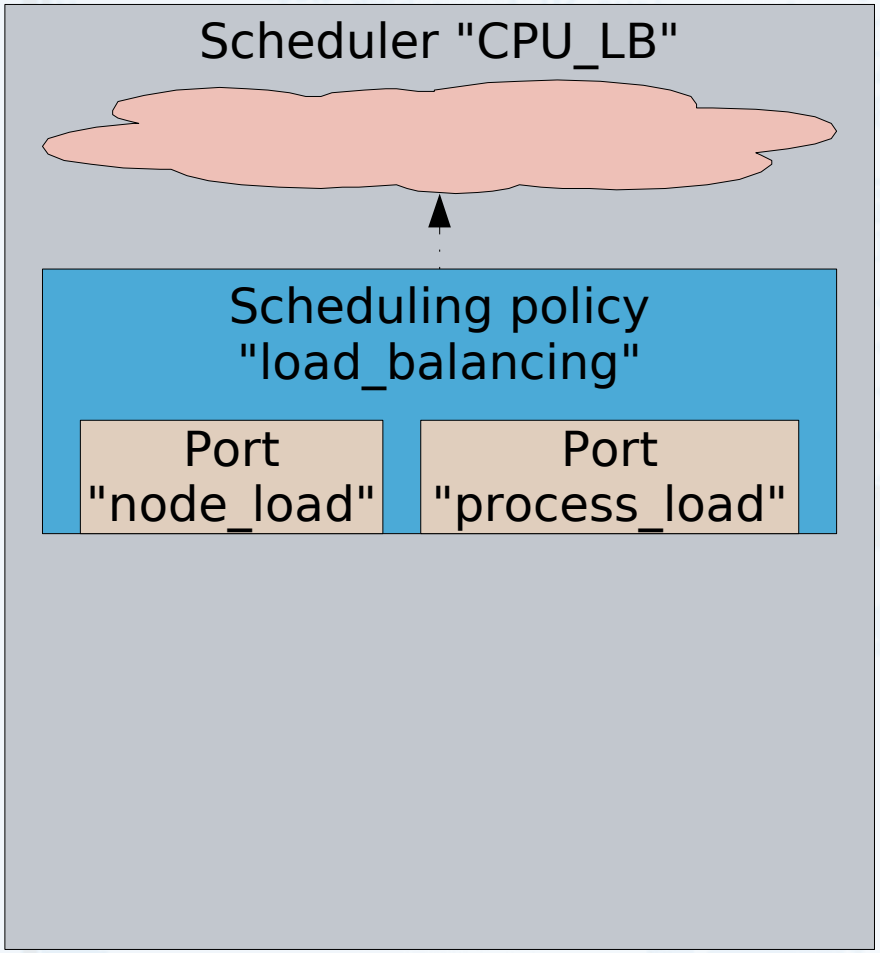


Setting up a scheduler



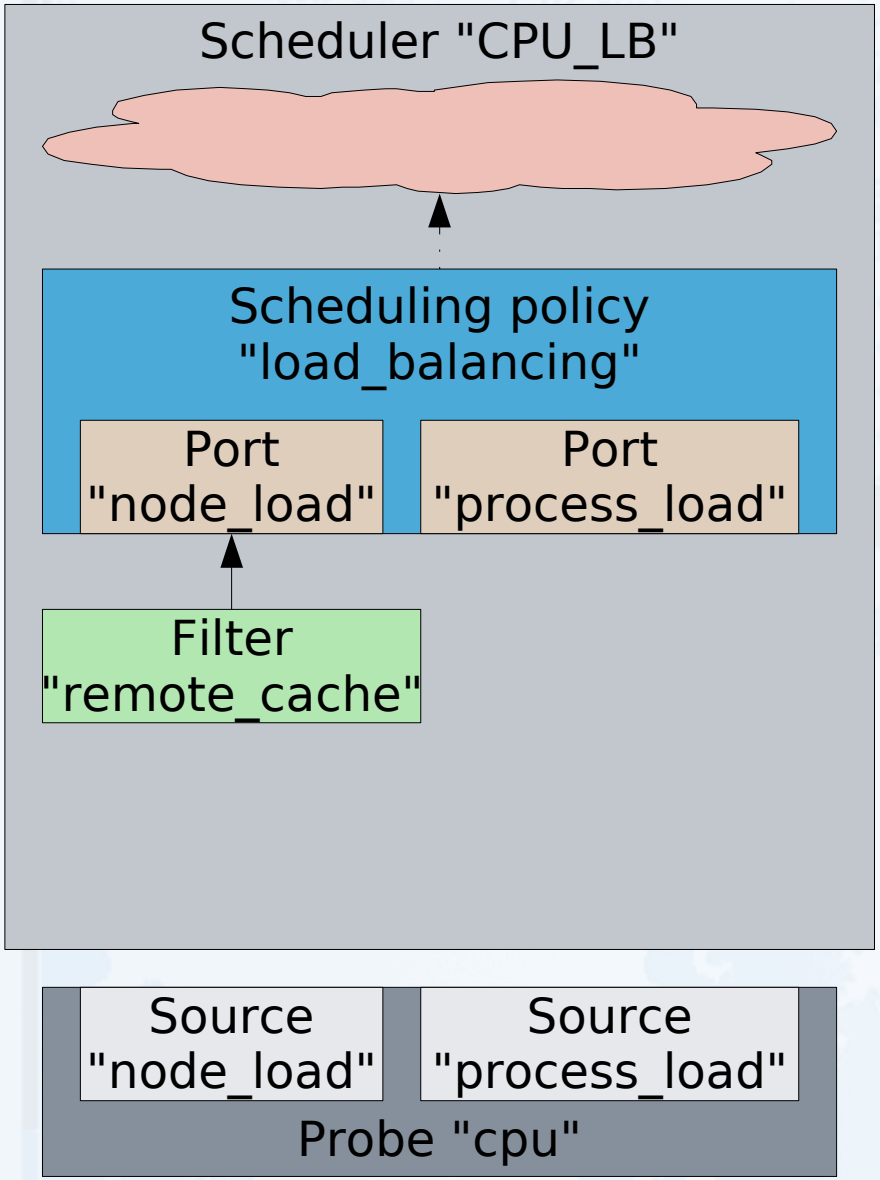

```
# SROOT=/config/krg_schedulers
# PROBE=$SROOT/probes/cpu
# mkdir $PROBE
     [loads cpu.ko and] activates the probe
# echo 1000 > $PROBE/probe_period
              make the probe notify updates
                            every 1000 jiffies
# SCHEDULER=$SROOT/schedulers/CPU_LB
# mkdir $SCHEDULER
      creates a new scheduler called CPU_LB
```

Setting up a scheduler



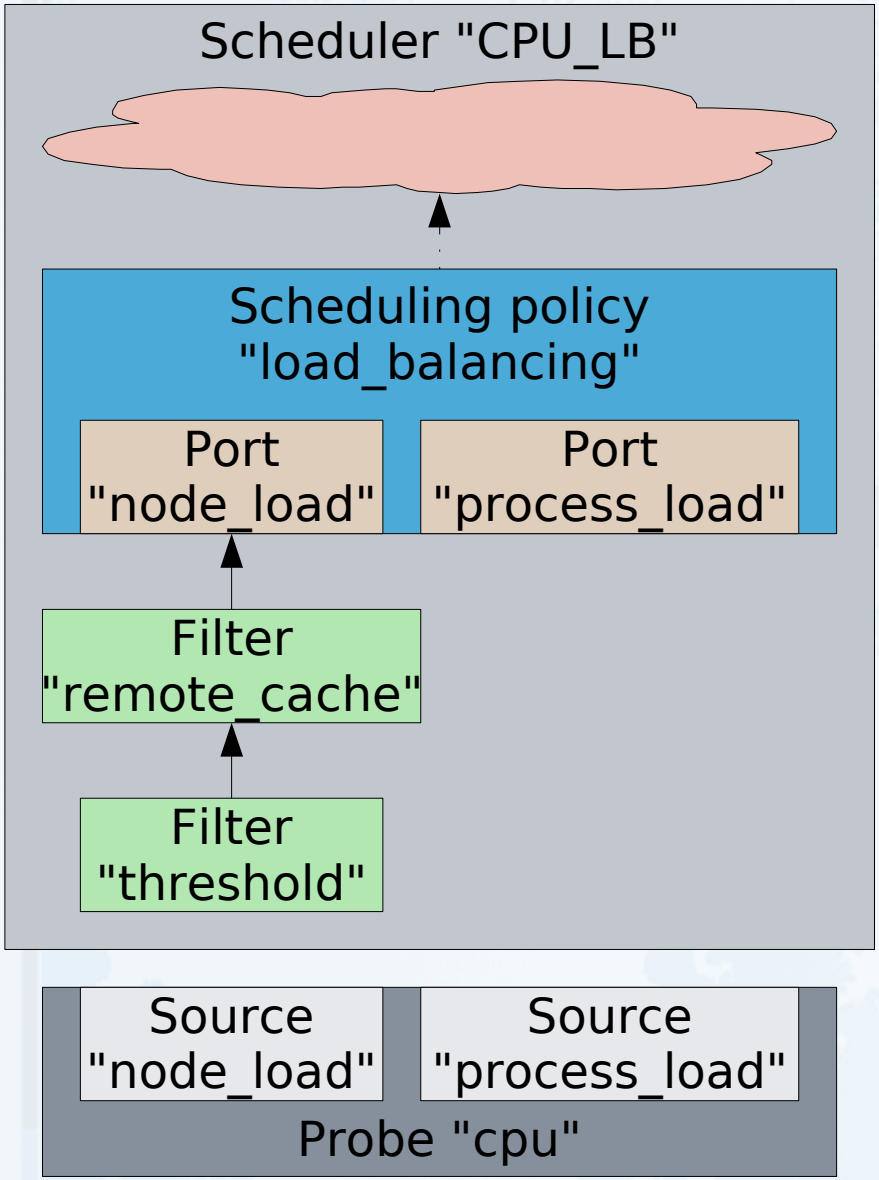

```
# SROOT=/config/krg_schedulers
# PROBE=$SROOT/probes/cpu
# mkdir $PROBE
     [loads cpu.ko and] activates the probe
# echo 1000 > $PROBE/probe_period
               make the probe notify updates
                              every 1000 jiffies
# SCHEDULER=$SROOT/schedulers/CPU_LB
# mkdir $SCHEDULER
      creates a new scheduler called CPU_LB
# POLICY=$SCHEDULER/load_balancing
# mkdir $POLICY
               [loads load_balancing.ko and]
        instantiates a load balancing policy
```

Setting up a scheduler



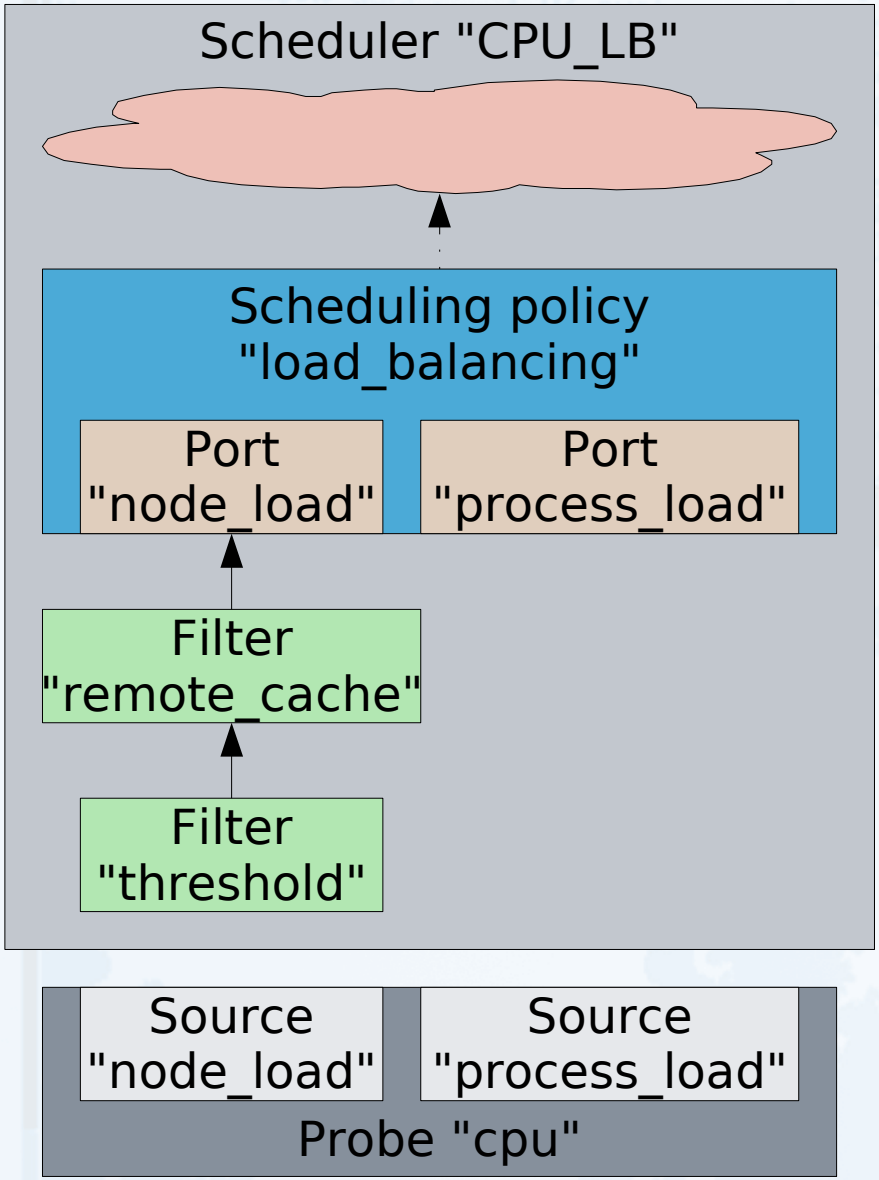

```
# SROOT=/config/krg_schedulers
# PROBE=$SROOT/probes/cpu
# mkdir $PROBE
     [loads cpu.ko and] activates the probe
# echo 1000 > $PROBE/probe_period
               make the probe notify updates
                              every 1000 jiffies
# SCHEDULER=$SROOT/schedulers/CPU_LB
# mkdir $SCHEDULER
      creates a new scheduler called CPU_LB
# POLICY=$SCHEDULER/load_balancing
# mkdir $POLICY
               [loads load_balancing.ko and]
       instantiates a load balancing policy
# cd $POLICY/node_load
# mkdir remote_cache
                     [loads remote_cache.ko,]
     instantiates a remote cache filter and
       make it the source of port node_load
```

Setting up a scheduler



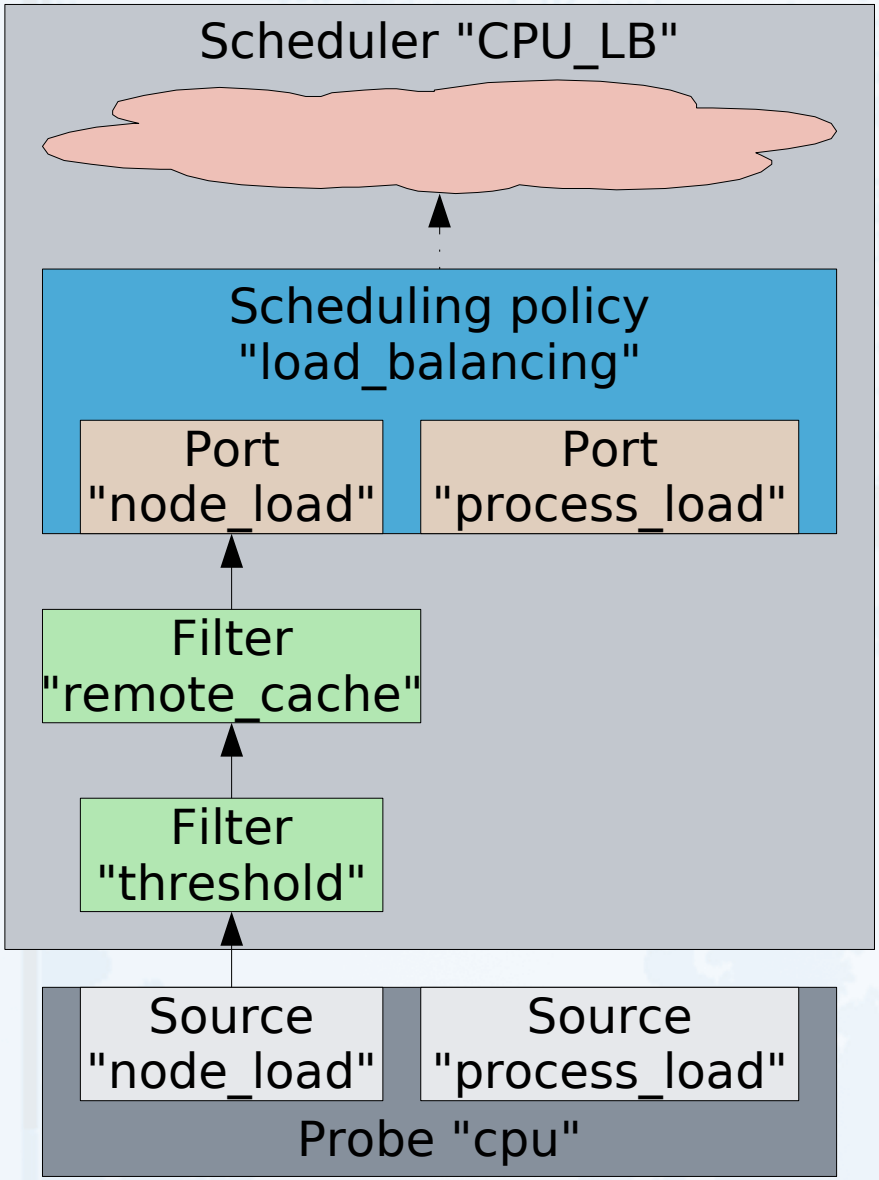

```
# SROOT=/config/krg_schedulers
# PROBE=$SROOT/probes/cpu
# mkdir $PROBE
     [loads cpu.ko and] activates the probe
# echo 1000 > $PROBE/probe_period
               make the probe notify updates
                              every 1000 jiffies
# SCHEDULER=$SROOT/schedulers/CPU_LB
# mkdir $SCHEDULER
      creates a new scheduler called CPU_LB
# POLICY=$SCHEDULER/load_balancing
# mkdir $POLICY
               [loads load_balancing.ko and]
       instantiates a load balancing policy
# cd $POLICY/node_load
# mkdir remote_cache
                        [loads remote_cache.ko,]
     instantiates a remote cache filter and
       make it the source of port node_load
# mkdir remote_cache/threshold
                           [loads threshold.ko,]
        instantiates a threshold filter and
  make it the source of filter remote_cache
```

Setting up a scheduler



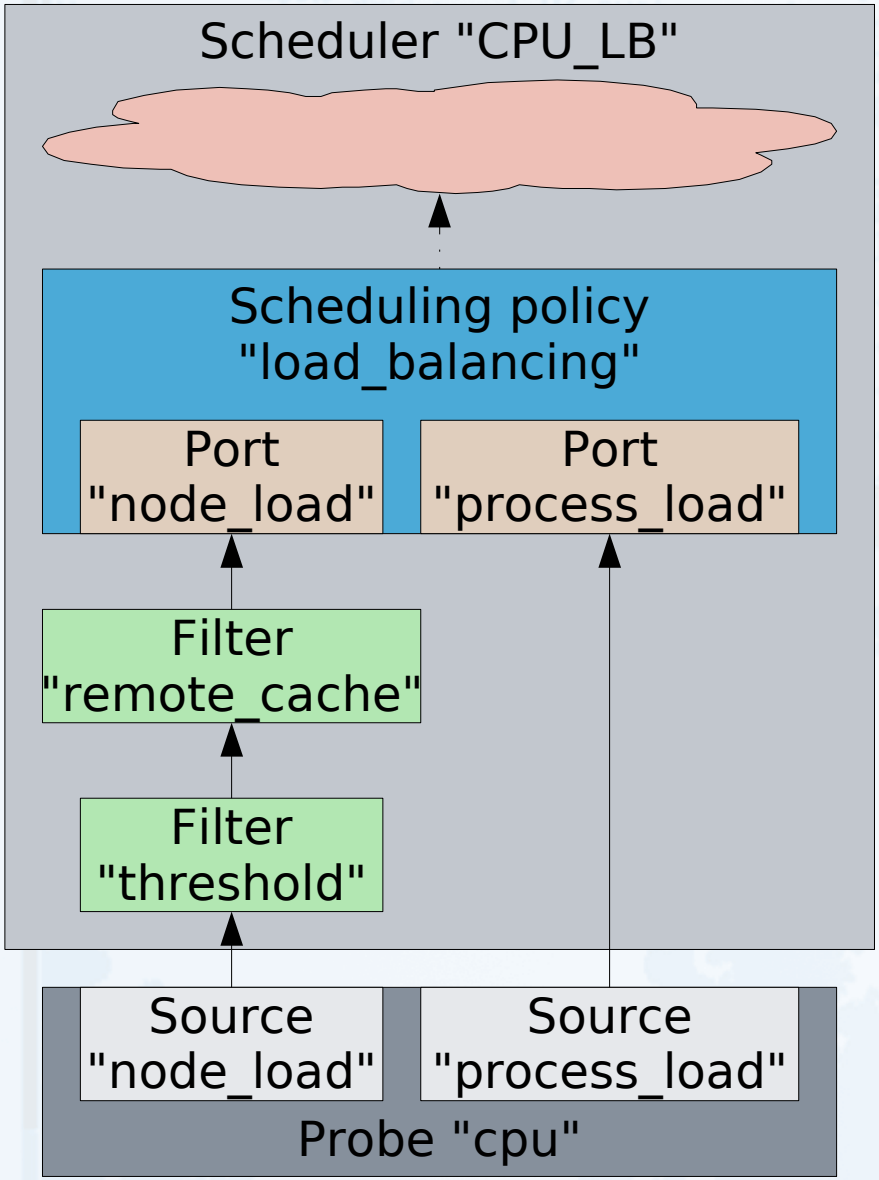

```
                              ...
# echo 1000 > $PROBE/probe_period
                  make the probe notify updates
                                  every 1000 jiffies
# SCHEDULER=$SROOT/schedulers/CPU_LB
# mkdir $SCHEDULER
      creates a new scheduler called CPU_LB
# POLICY=$SCHEDULER/load_balancing
# mkdir $POLICY
                  [loads load_balancing.ko and]
       instantiates a load balancing policy
# cd $POLICY/node_load
# mkdir remote_cache
                        [loads remote_cache.ko,]
     instantiates a remote cache filter and
       make it the source of port node_load
# mkdir remote_cache/threshold
                             [loads threshold.ko,]
        instantiates a threshold filter and
  make it the source of filter remote_cache
# cd remote_cache/threshold
# echo 100 > threshold
                       set the filter's threshold
```

Setting up a scheduler



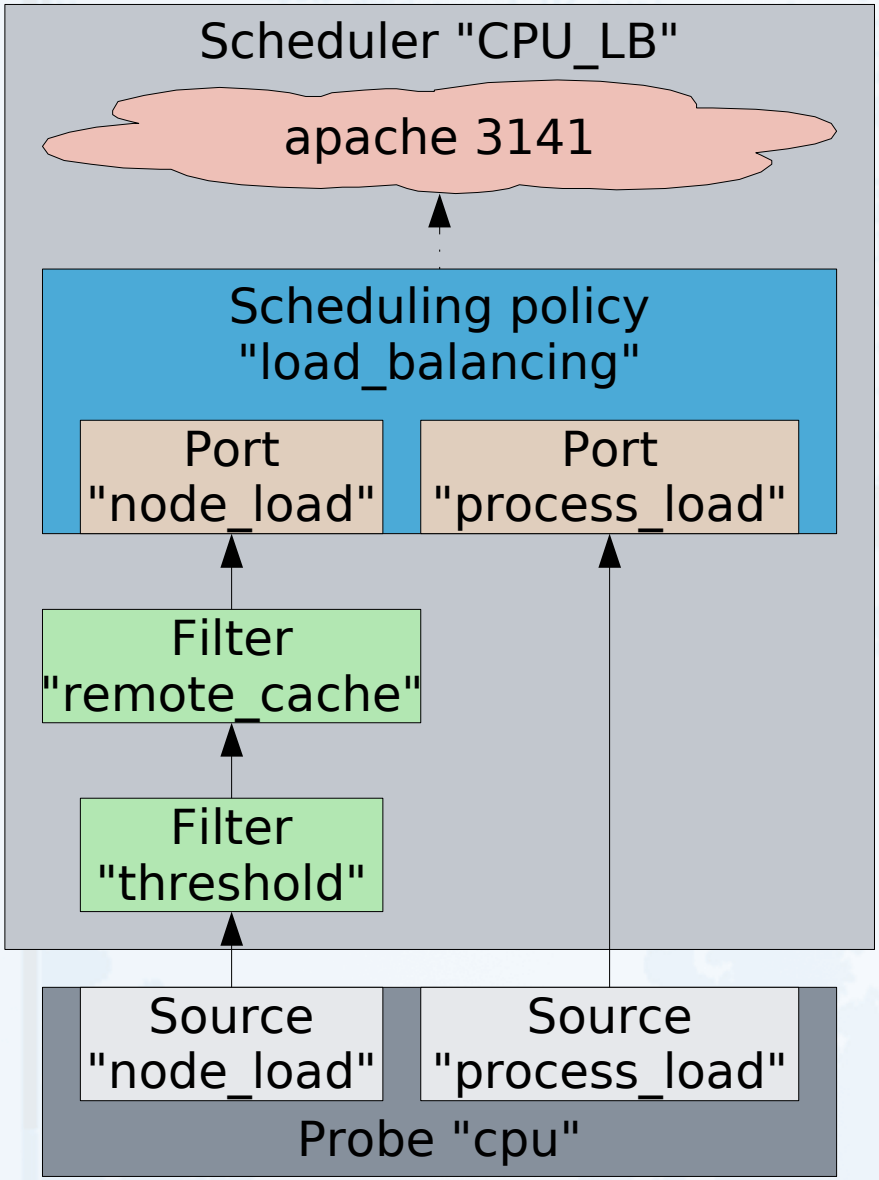

```
                                ...
# SCHEDULER=$SROOT/schedulers/CPU_LB
# mkdir $SCHEDULER
      creates a new scheduler called CPU_LB
# POLICY=$SCHEDULER/load_balancing
# mkdir $POLICY
               [loads load_balancing.ko and]
       instantiates a load balancing policy
# cd $POLICY/node_load
# mkdir remote_cache
                     [loads remote_cache.ko,]
     instantiates a remote cache filter and
       make it the source of port node_load
# mkdir remote_cache/threshold
                          [loads threshold.ko,]
        instantiates a threshold filter and
  make it the source of filter remote_cache
# cd remote_cache/threshold
# echo 100 > threshold
                   set the filter's threshold
# ln -s $PROBE/node_load cpu_load
                 make probe source node_load
              the source of filter threshold
```

Setting up a scheduler



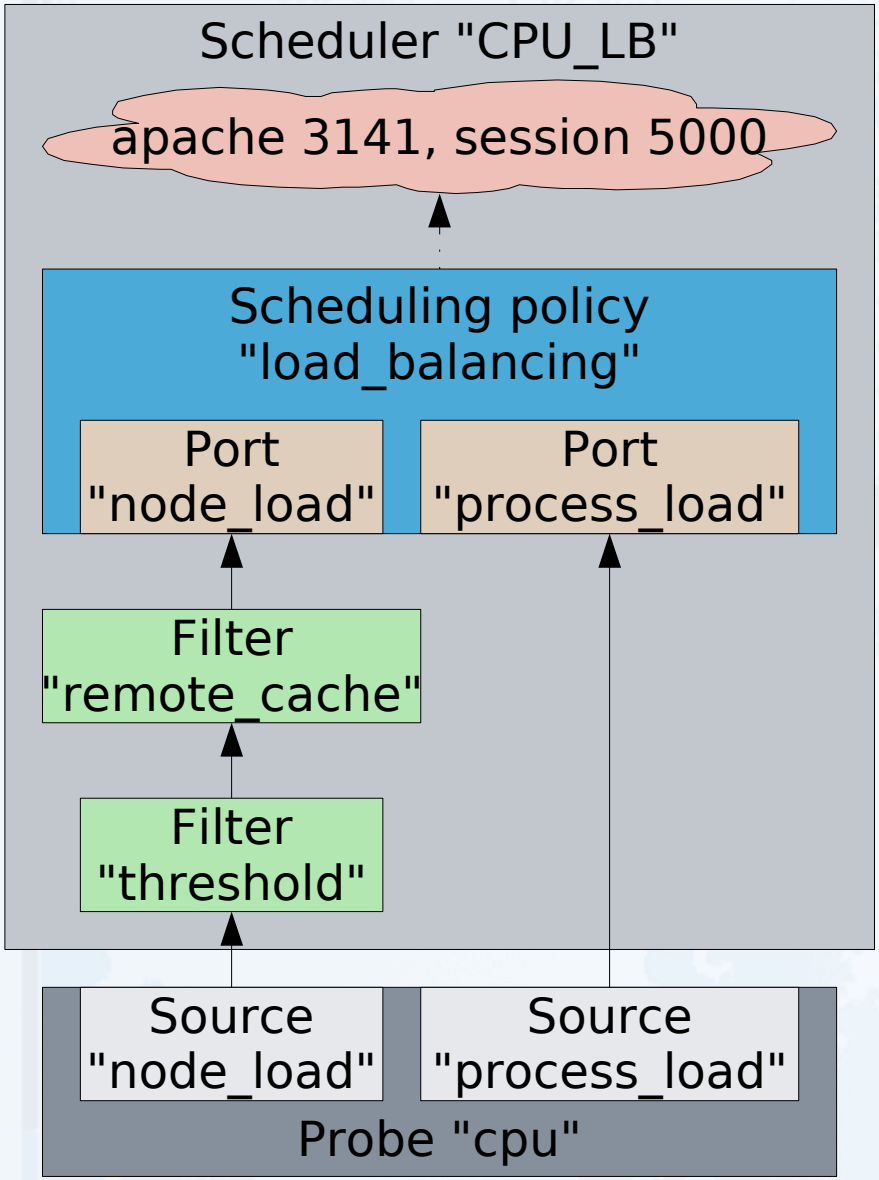

```
                                ...
# mkdir $POLICY
                    [loads load_balancing.ko and]
          instantiates a load balancing policy
# cd $POLICY/node_load
# mkdir remote_cache
                          [loads remote_cache.ko,]
       instantiates a remote cache filter and
         make it the source of port node_load
# mkdir remote_cache/threshold
                           [loads threshold.ko,]
          instantiates a threshold filter and
  make it the source of filter remote_cache
# cd remote_cache/threshold
# echo 100 > threshold
                       set the filter's threshold
# ln -s $PROBE/node_load cpu_load
                    make probe source node_load
                  the source of filter threshold
# cd $POLICY/process_load
# ln -s $PROBE/process_load cpu_load
                  make probe source process_load
                the source of port process_load
```

Setting up a scheduler



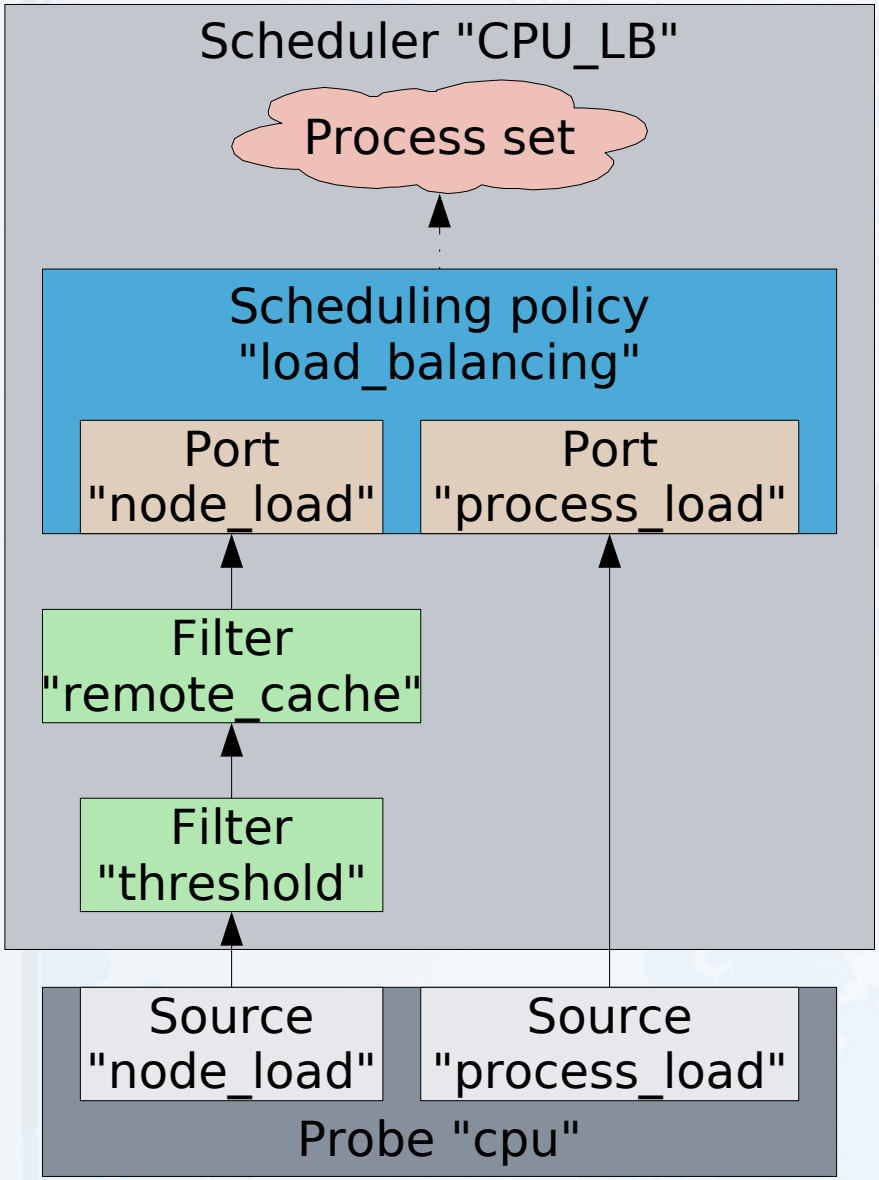

```
                              ...
# cd $POLICY/node_load
# mkdir remote_cache
                        [loads remote_cache.ko,]
     instantiates a remote cache filter and
       make it the source of port node_load
# mkdir remote_cache/threshold
                           [loads threshold.ko,]
         instantiates a threshold filter and
  make it the source of filter remote_cache
# cd remote_cache/threshold
# echo 100 > threshold
                     set the filter's threshold
# ln -s $PROBE/node_load cpu_load
                   make probe source node_load
                the source of filter threshold
# cd $POLICY/process_load
# ln -s $PROBE/process_load cpu_load
                make probe source process_load
              the source of port process_load
# cd $SCHEDULER/process_set
# mkdir single_processes/`pidof apache`
     attach running Apache to the scheduler
```

Setting up a scheduler

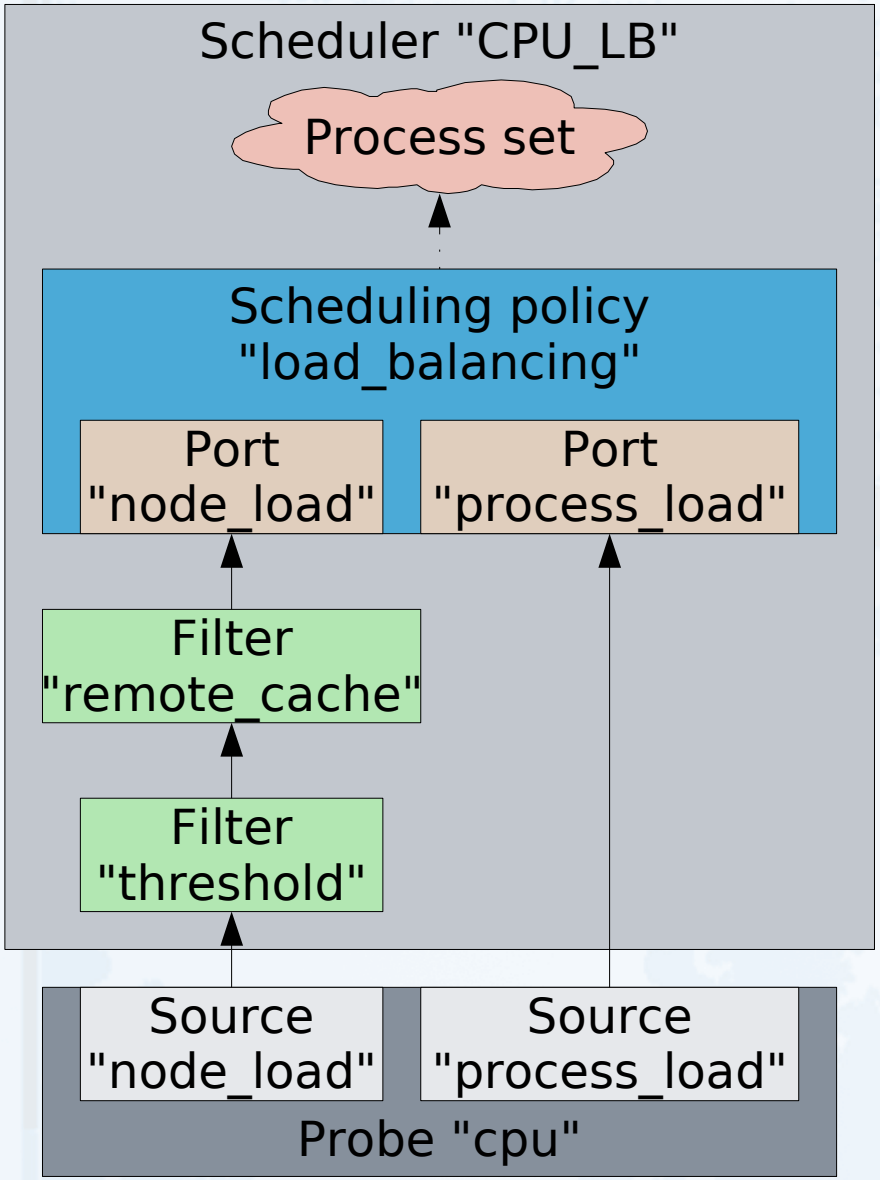



```
                              ...
     instantiates a remote cache filter and
       make it the source of port node_load
# mkdir remote_cache/threshold
                            [loads threshold.ko,]
        instantiates a threshold filter and
  make it the source of filter remote_cache
# cd remote_cache/threshold
# echo 100 > threshold
                     set the filter's threshold
# ln -s $PROBE/node_load cpu_load
                  make probe source node_load
                the source of filter threshold
# cd $POLICY/process_load
# ln -s $PROBE/process_load cpu_load
                make probe source process_load
              the source of port process_load
# cd $SCHEDULER/process_set
# mkdir single_processes/`pidof apache`
     attach running Apache to the scheduler
# mkdir process_sessions/`ps h -osid $$`
  attach all (running and future) processes
    of the current session to the scheduler
```

Modifying a policy
Ex: removing the threshold filter

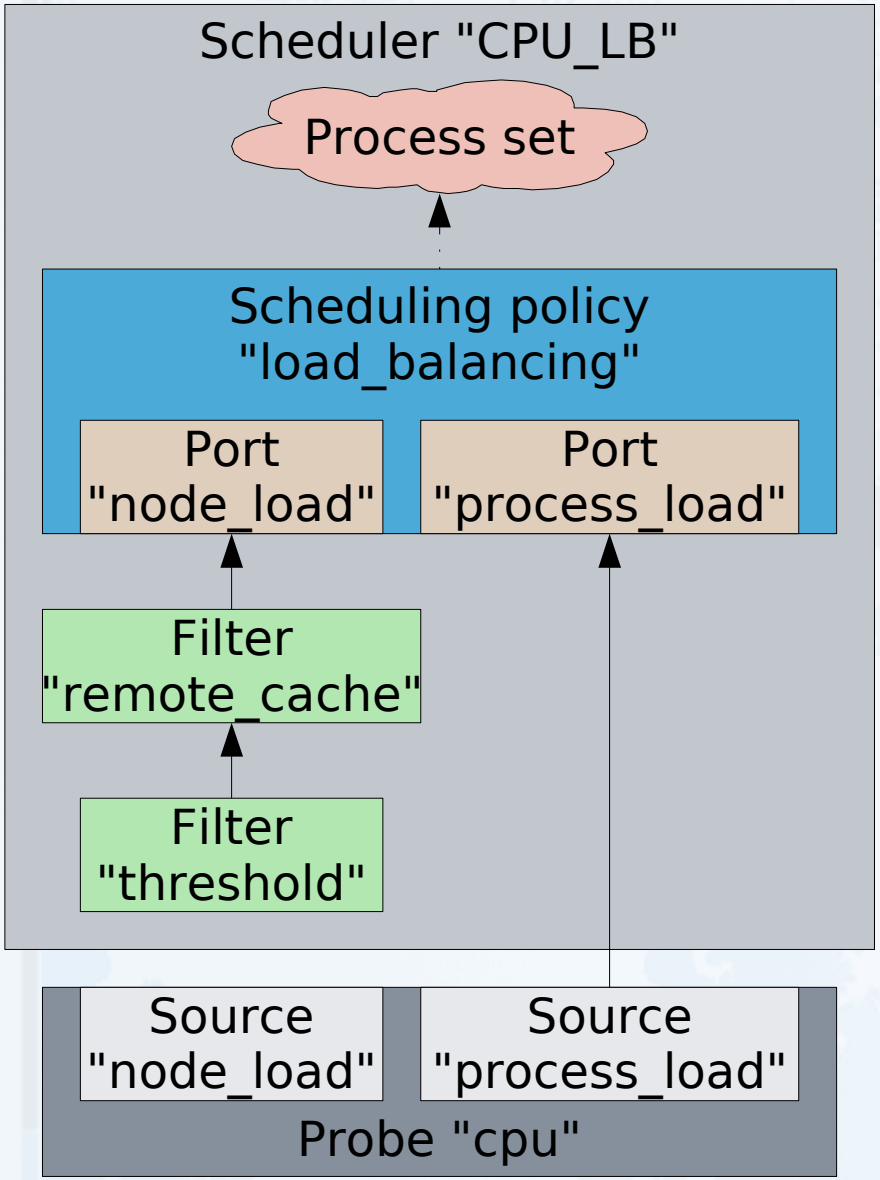


Modifying a policy
Ex: removing the threshold filter



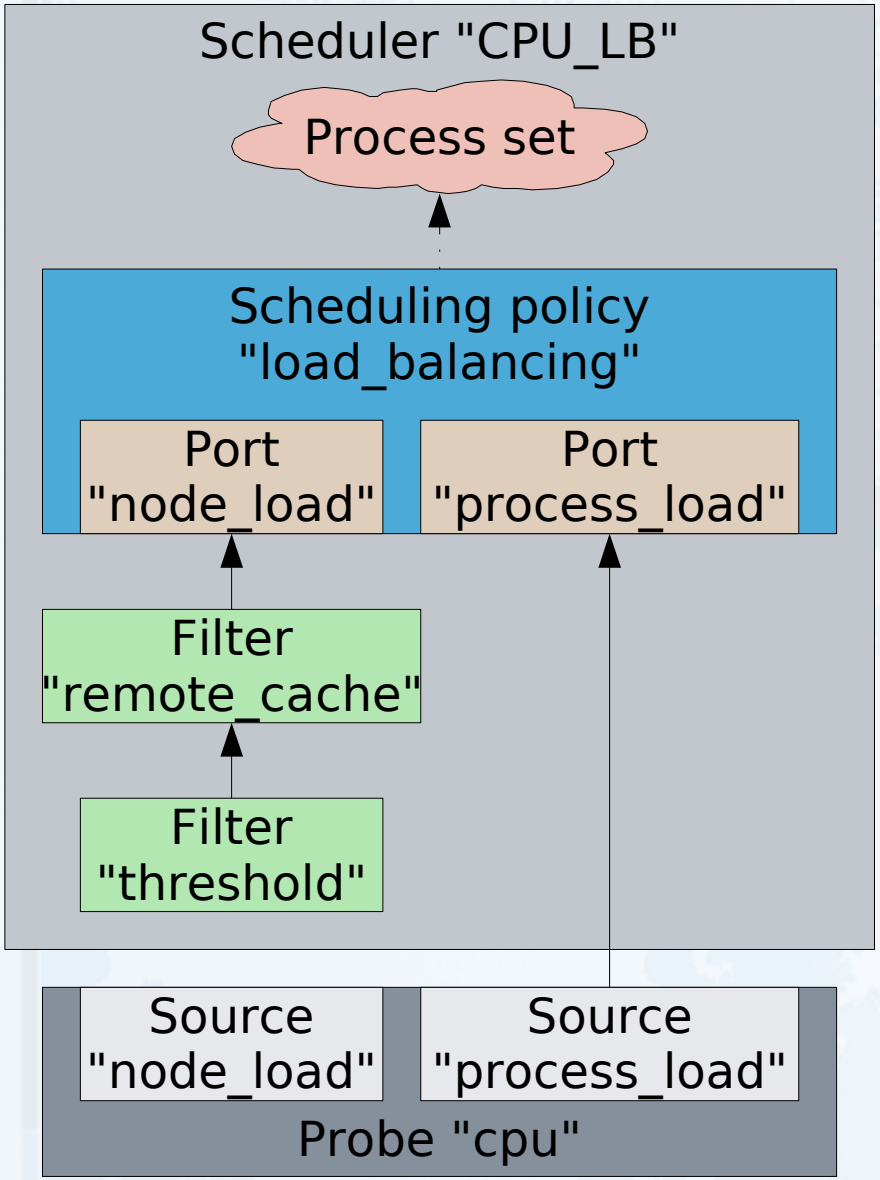

```
# cd $POLICY/remote_cache
# rm threshold/cpu_load
```

Modifying a policy
Ex: removing the threshold filter



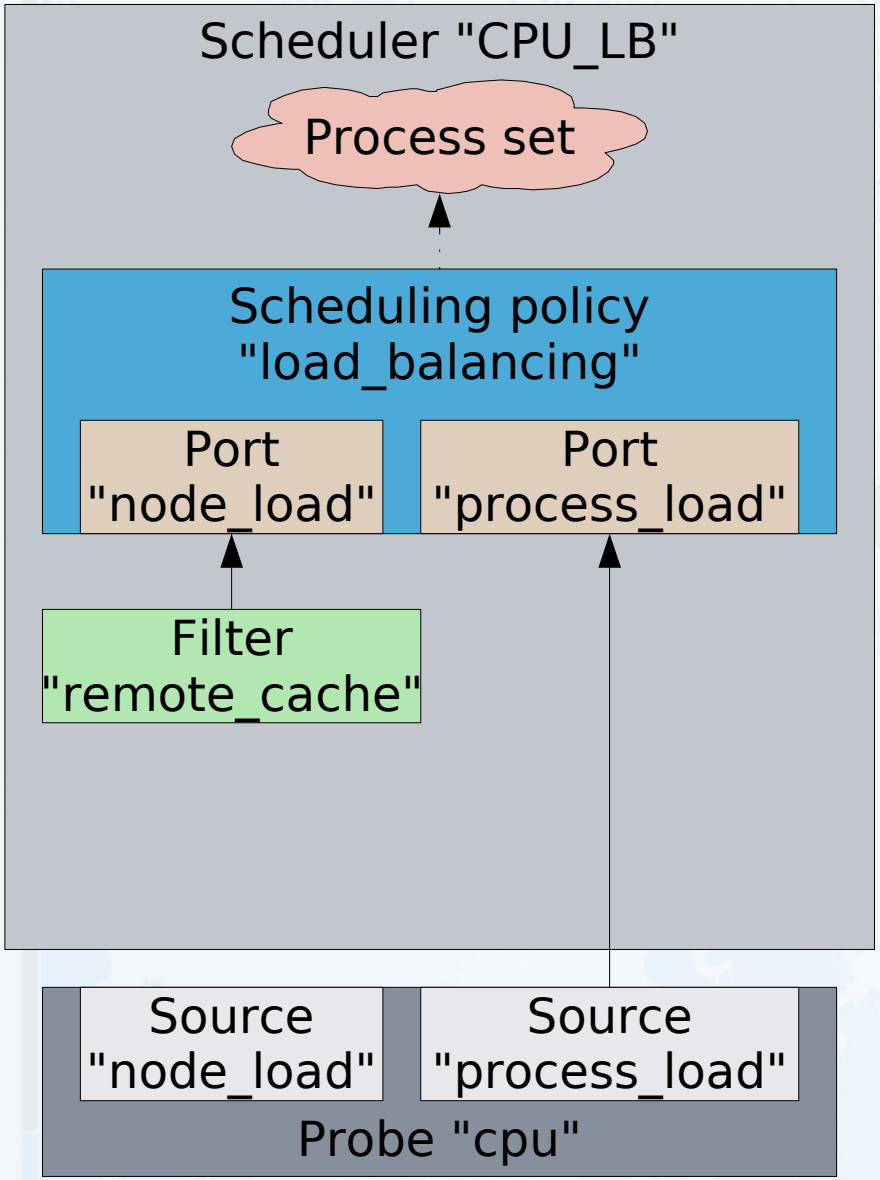

```
# cd $POLICY/remote_cache
# rm threshold/cpu_load
```

Modifying a policy
Ex: removing the threshold filter



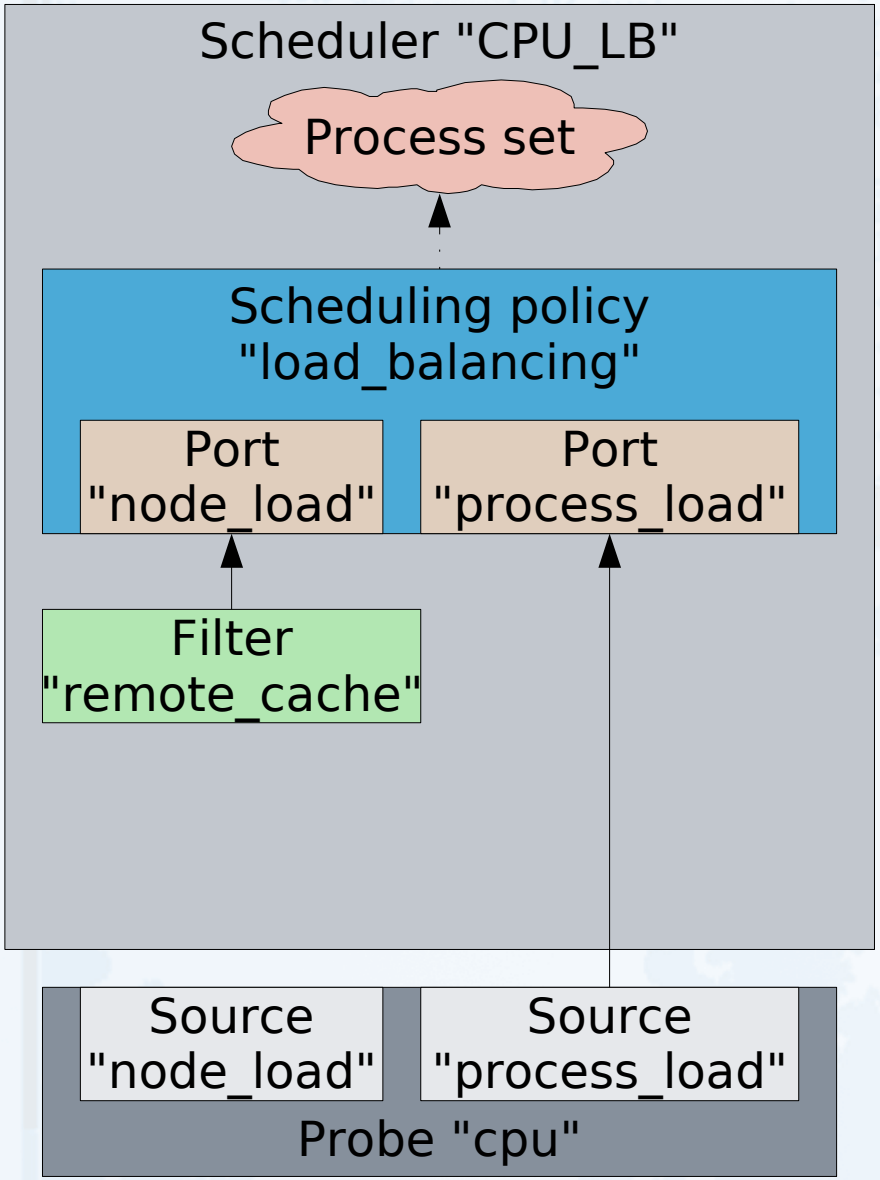

```
# cd $POLICY/remote_cache
# rm threshold/cpu_load
# rmdir threshold
```

Modifying a policy
Ex: removing the threshold filter



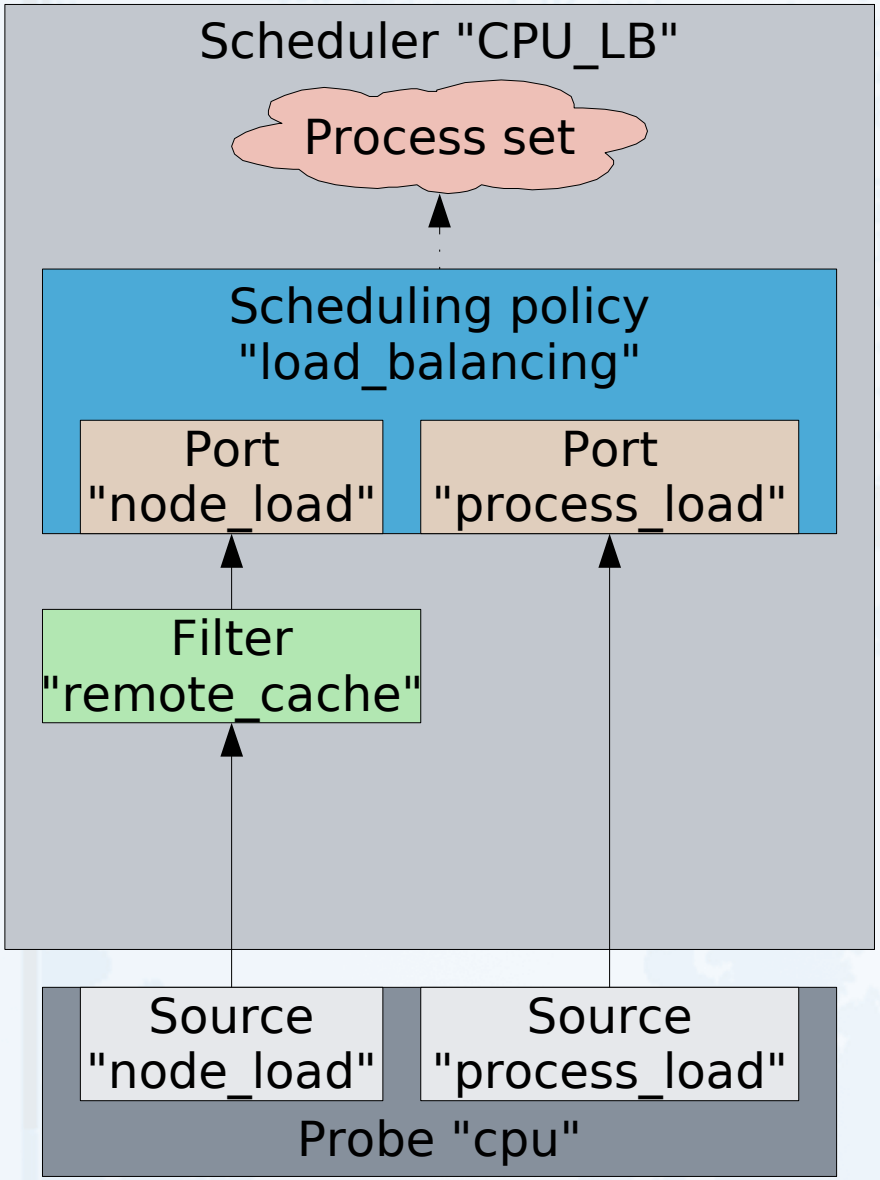

```
# cd $POLICY/remote_cache
# rm threshold/cpu_load
# rmdir threshold
```

Modifying a policy
Ex: removing the threshold filter



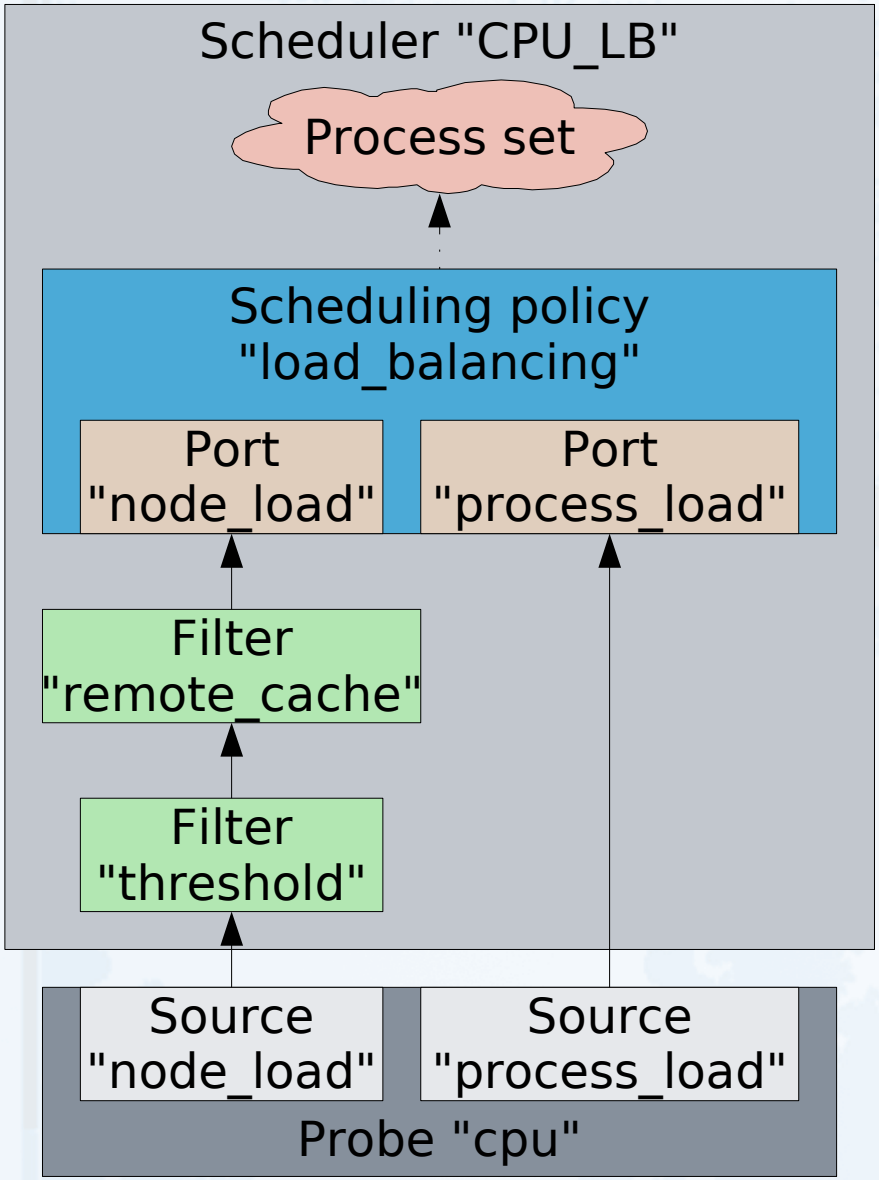

```
# cd $POLICY/remote_cache
# rm threshold/cpu_load
# rmdir threshold
# ln -s $PROBE/node_load cpu_load
```

Modifying a policy
Ex: removing the threshold filter

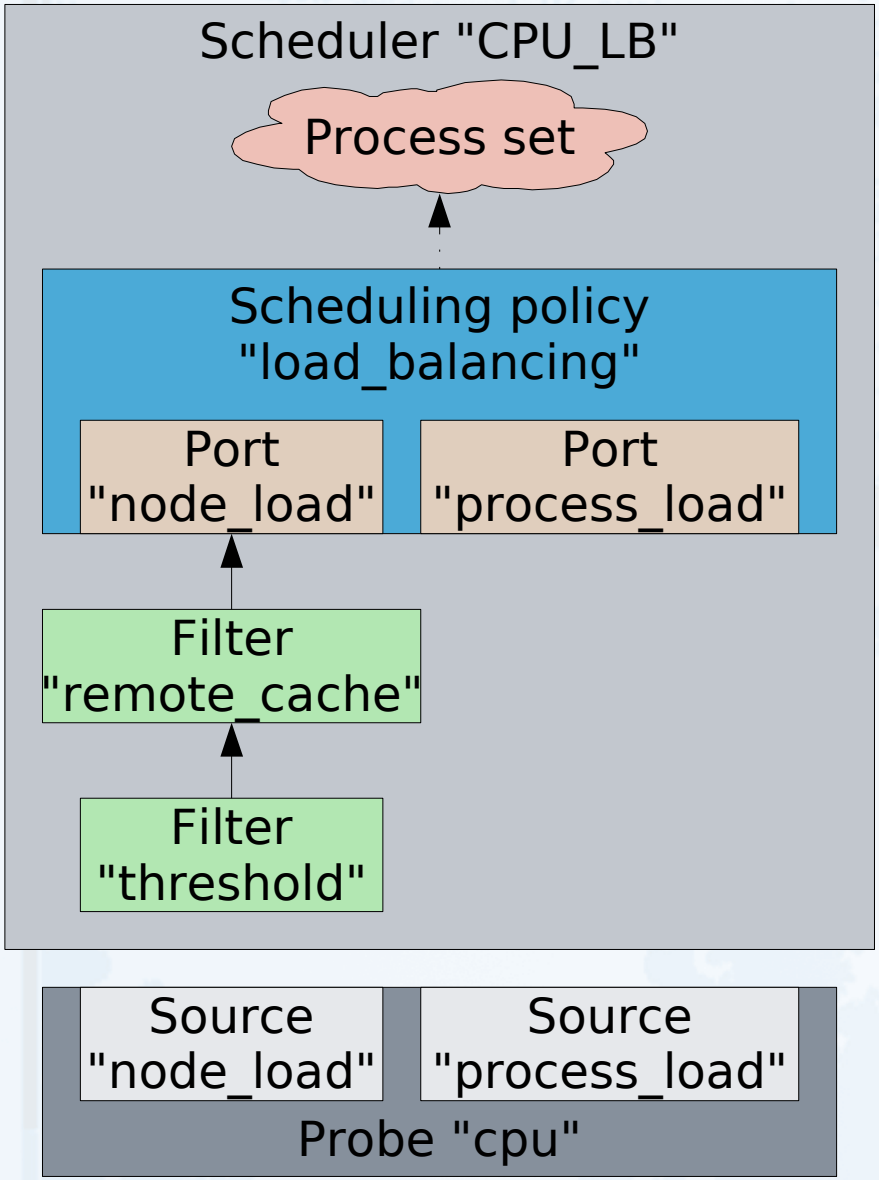



```
# cd $POLICY/remote_cache
# rm threshold/cpu_load
# rmdir threshold
# ln -s $PROBE/node_load cpu_load
```

Replacing a policy

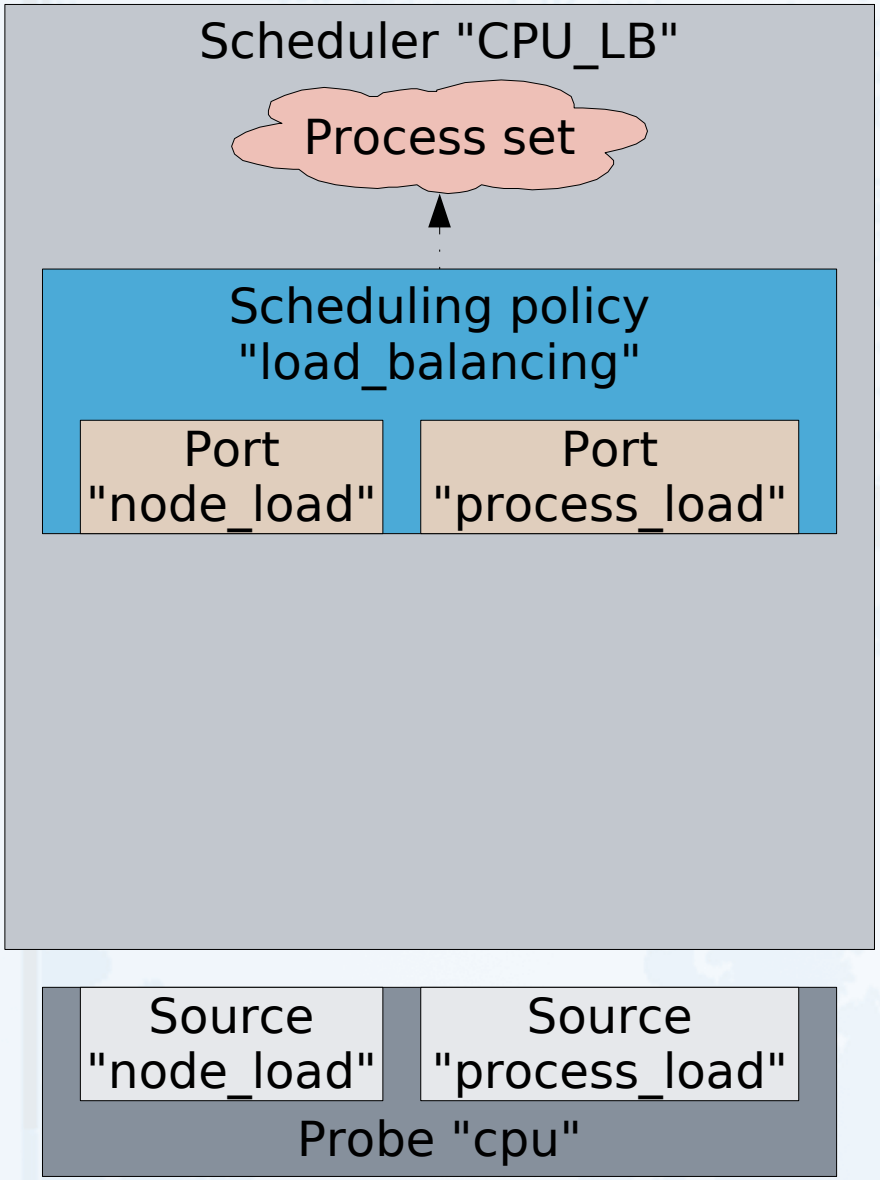


Replacing a policy



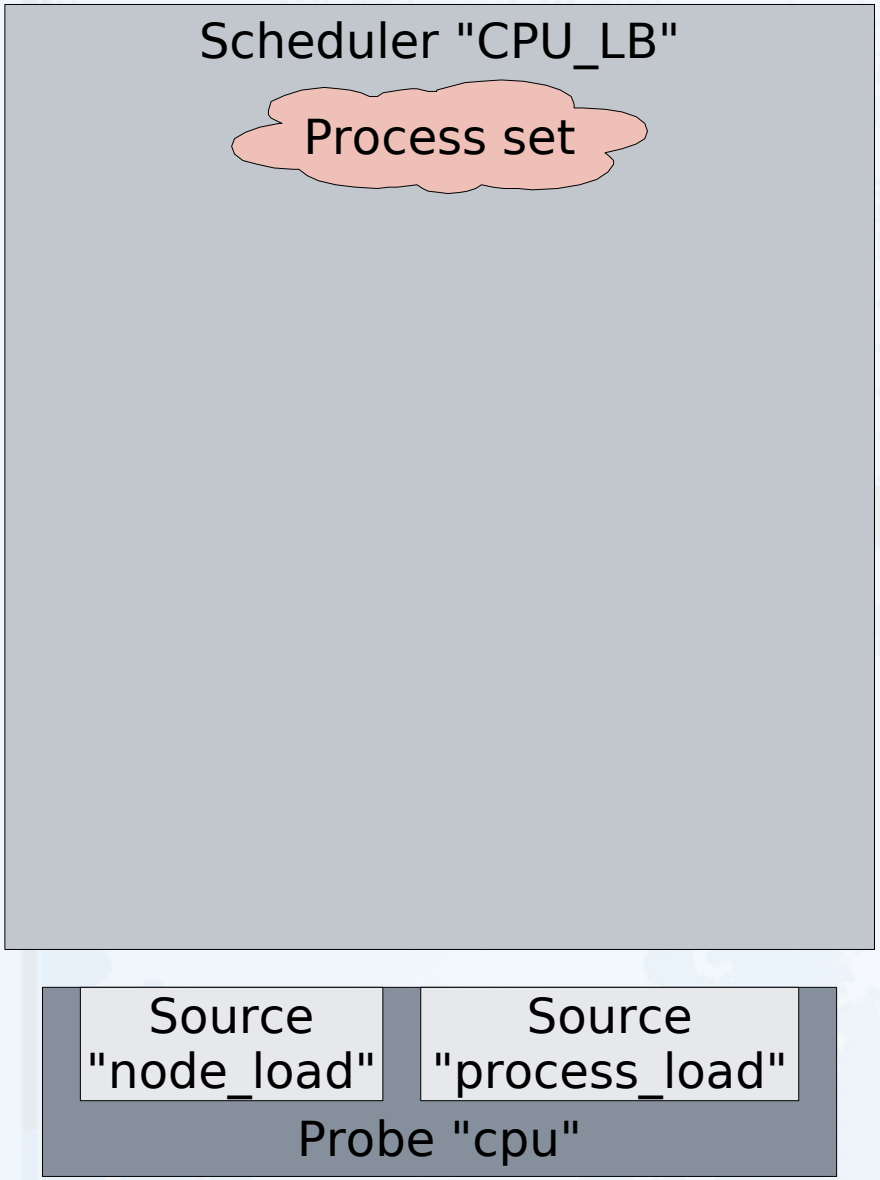

```
# find $POLICY -type l -exec rm '{}' ';'
 remove all links between policy and probes
```

Replacing a policy



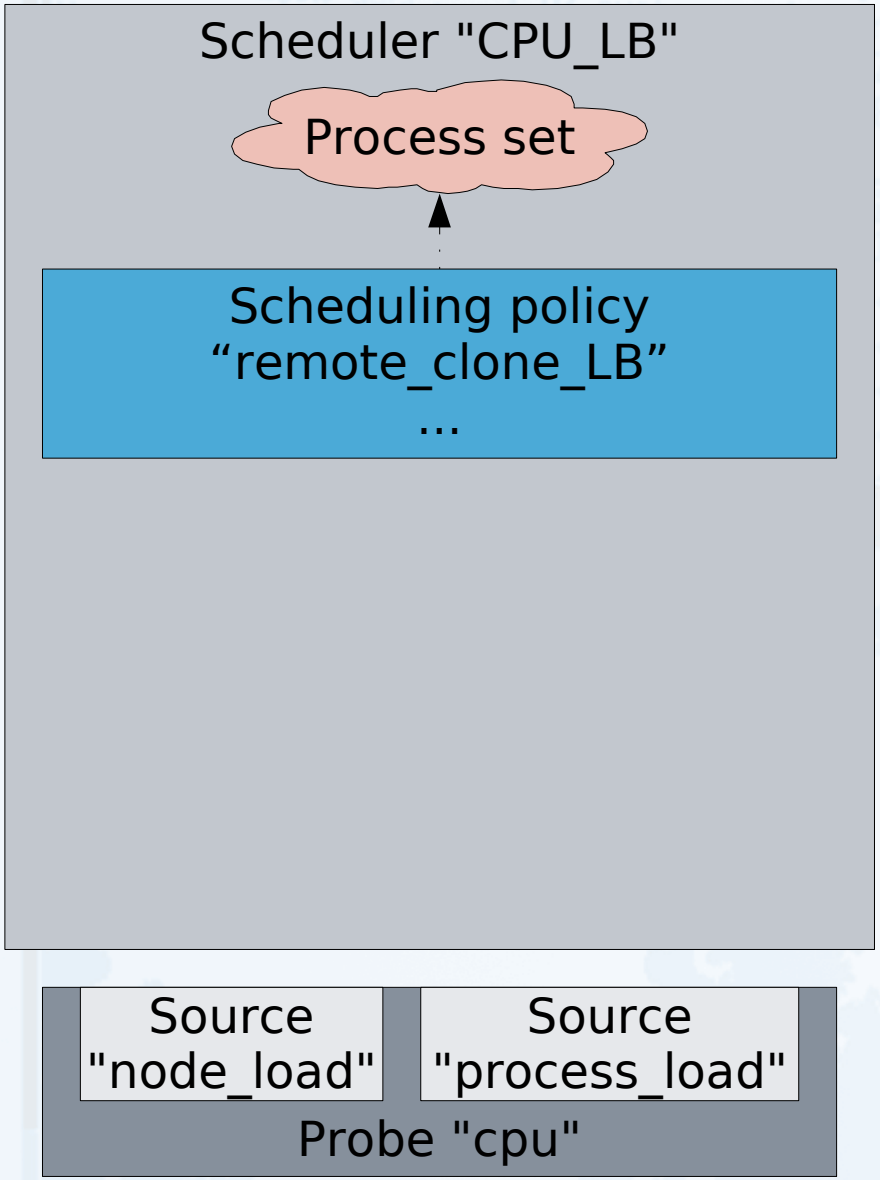

```
# find $POLICY -type l -exec rm '{}' ';'
 remove all links between policy and probes
# find $POLICY | tac | xargs -n 1 rmdir
   remove policy and all configured filters
```

Replacing a policy



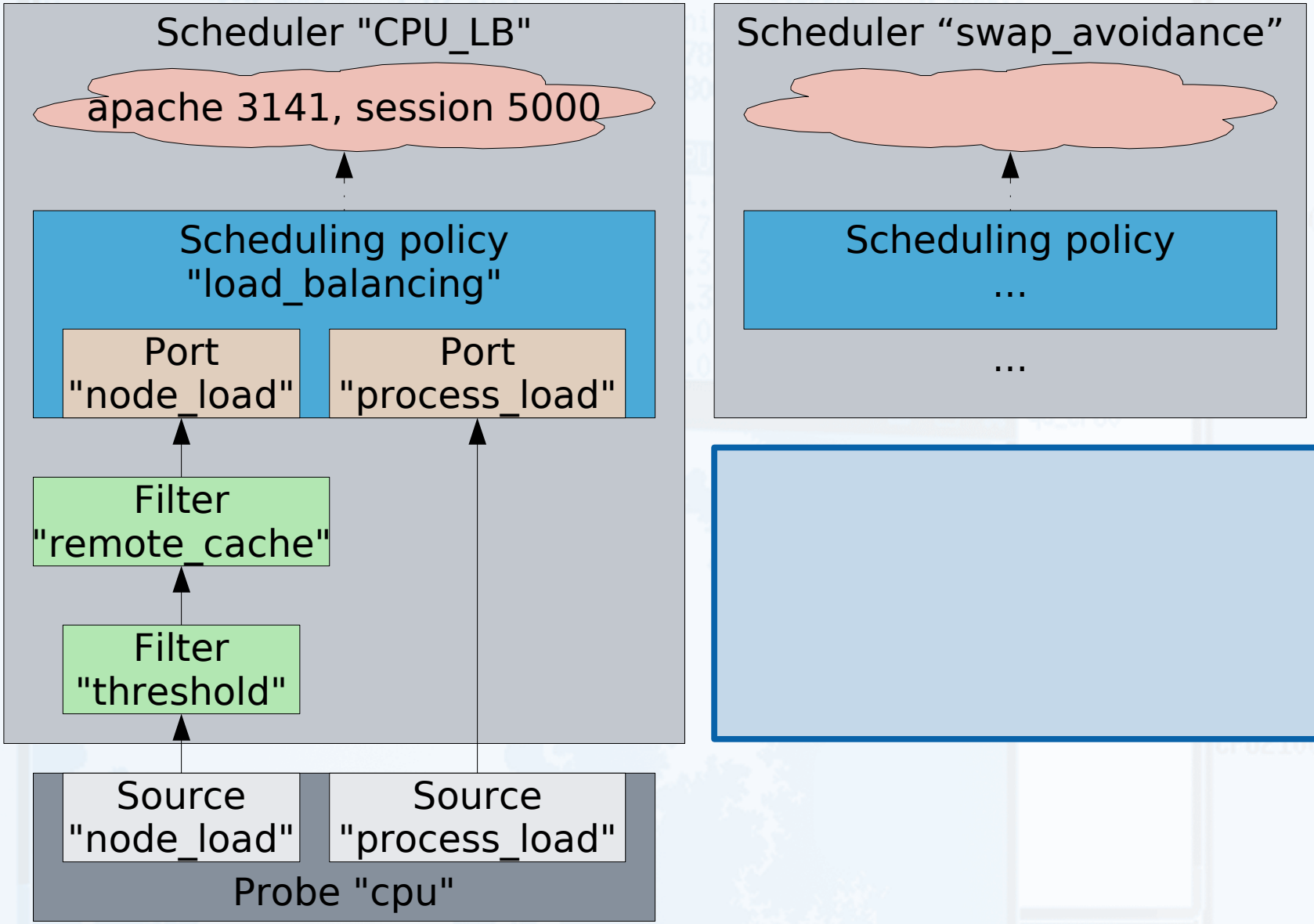

```
# find $POLICY -type l -exec rm '{}' ';'
 remove all links between policy and probes
# find $POLICY | tac | xargs -n 1 rmdir
   remove policy and all configured filters
```

Replacing a policy

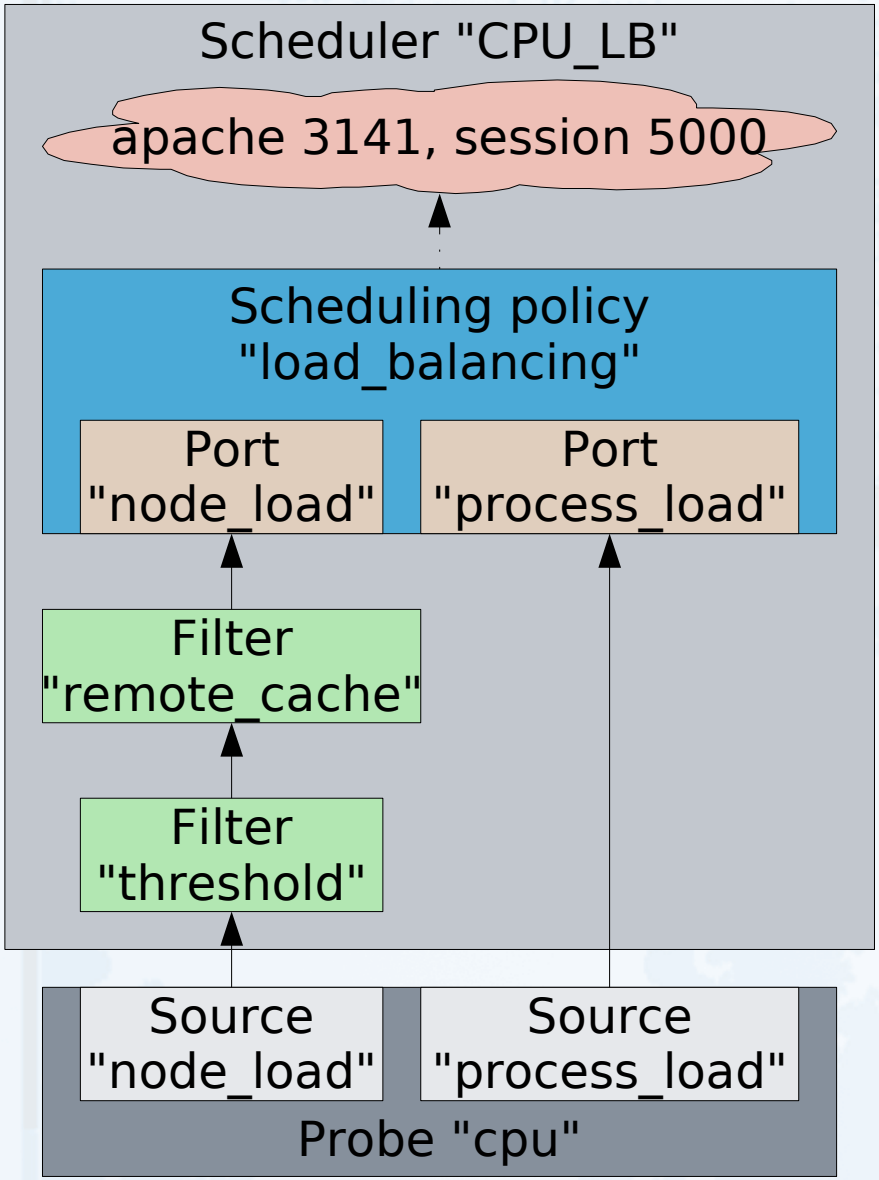


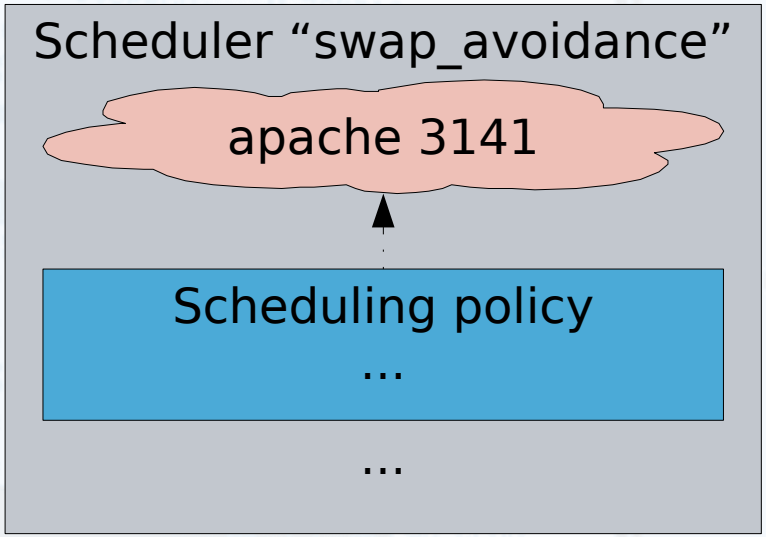

```
# find $POLICY -type l -exec rm '{}' ';'
 remove all links between policy and probes
# find $POLICY | tac | xargs -n 1 rmdir
   remove policy and all configured filters
# POLICY=$SCHEDULER/remote_clone_LB
# mkdir $POLICY
...
```

Stacking schedulers



Stacking schedulers



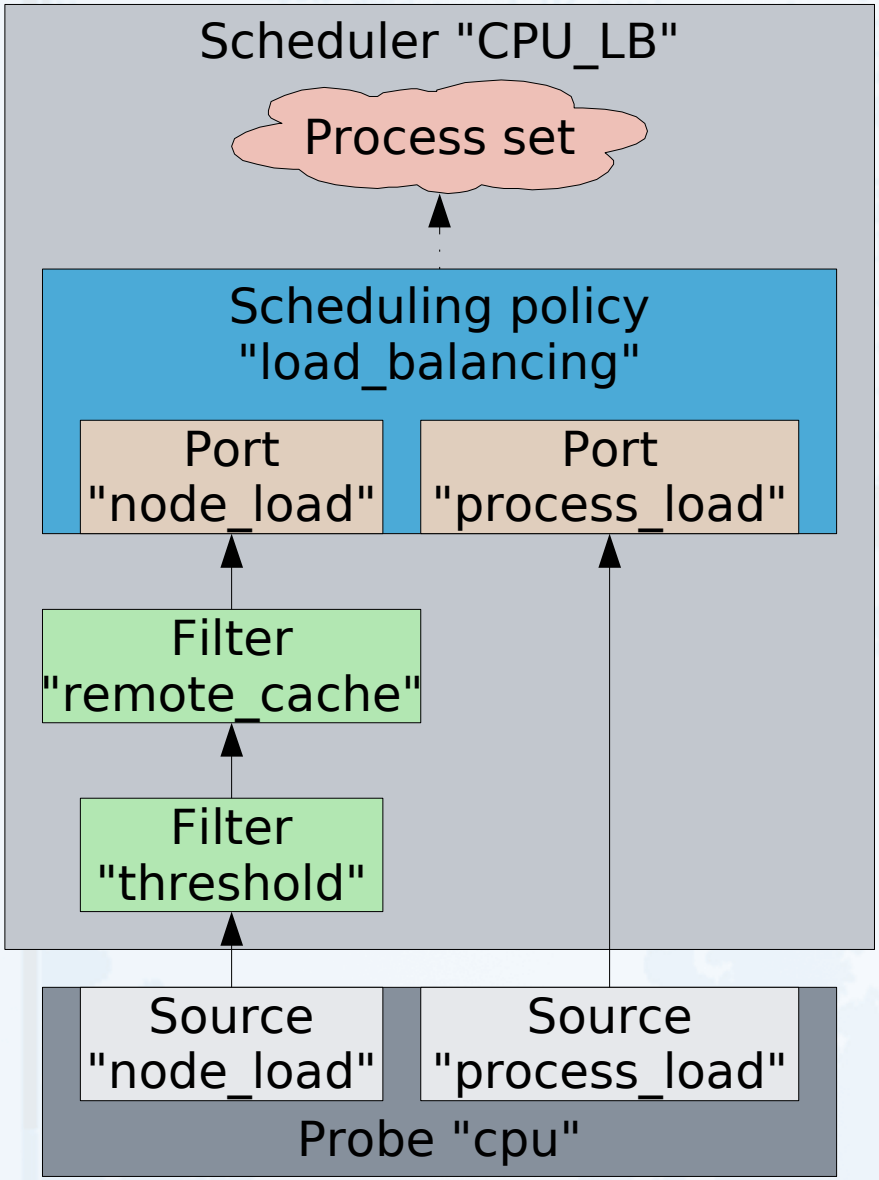



```
# cd $SROOT/schedulers
# cd swap_avoidance/process_set
# mkdir single_processes/`pidof apache`
```

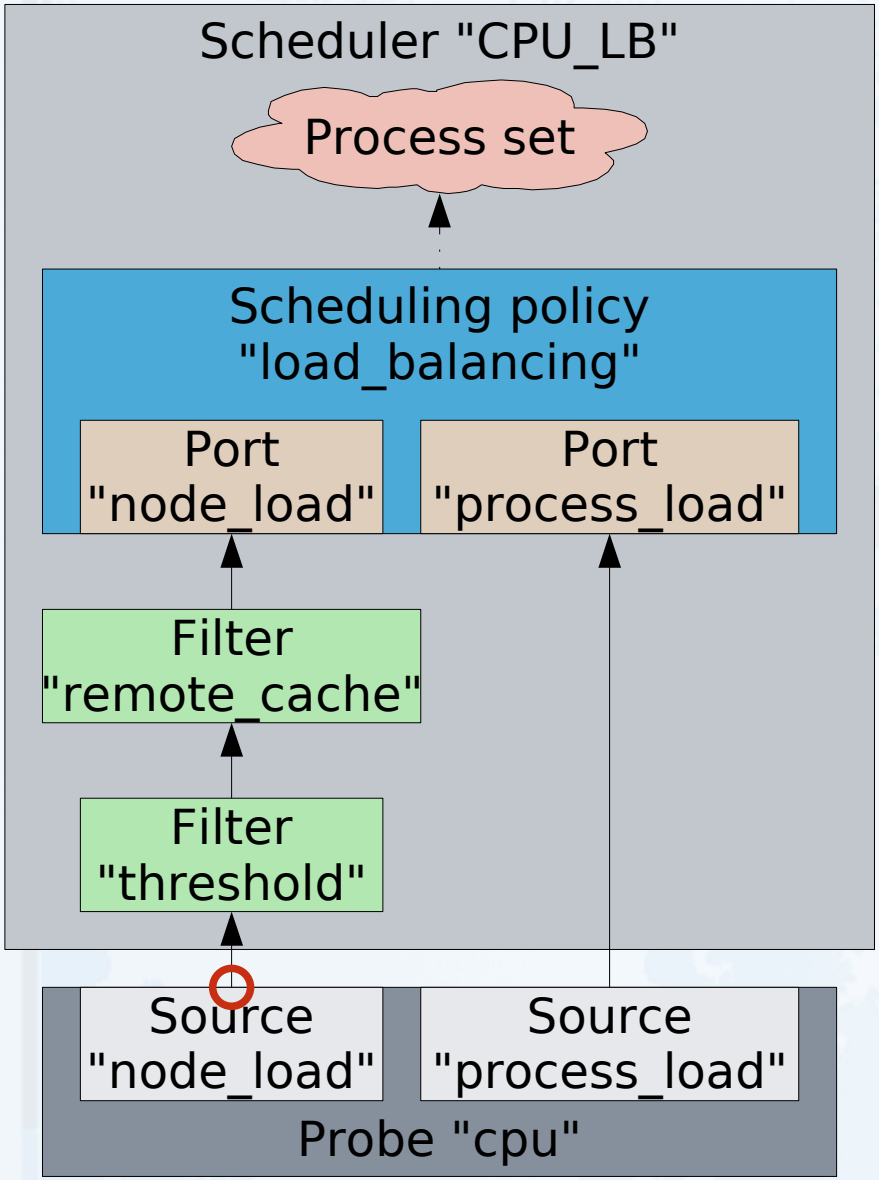
running apache is controlled by both **CPU_LB** and **swap_avoidance**
swap_avoidance takes precedence for Apache forks placement

Order matters!

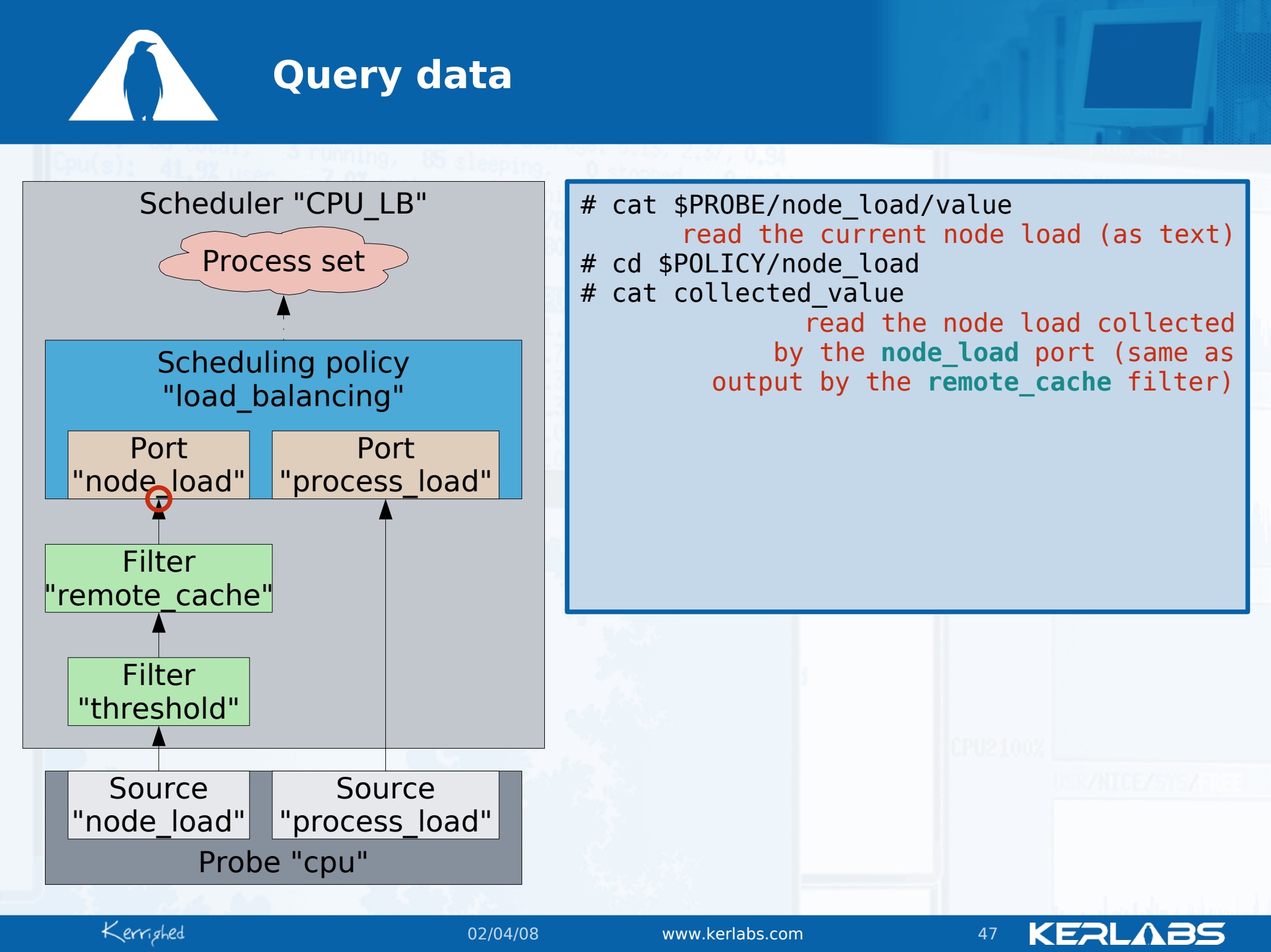
Query data


Scheduler "CPU_LB"
Process set
Scheduling policy
"load_balancing"
Port
"node_load"
Port
"process_load"
Filter
"remote_cache"
Filter
"threshold"
Source
"node_load"
Source
"process_load"
Probe "cpu"

Query data

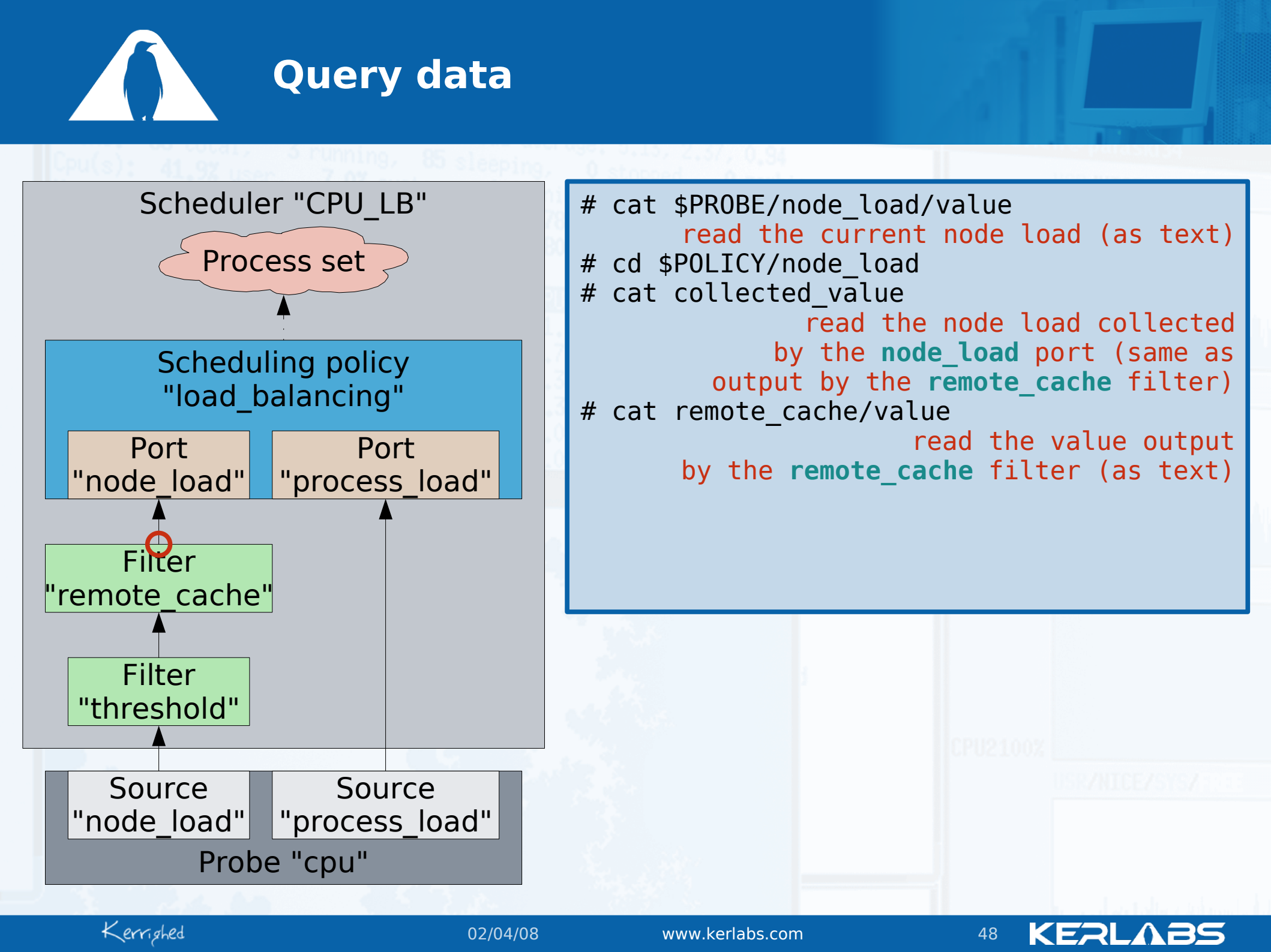



```
# cat $PROBE/node_load/value
        read the current node load (as text)
```



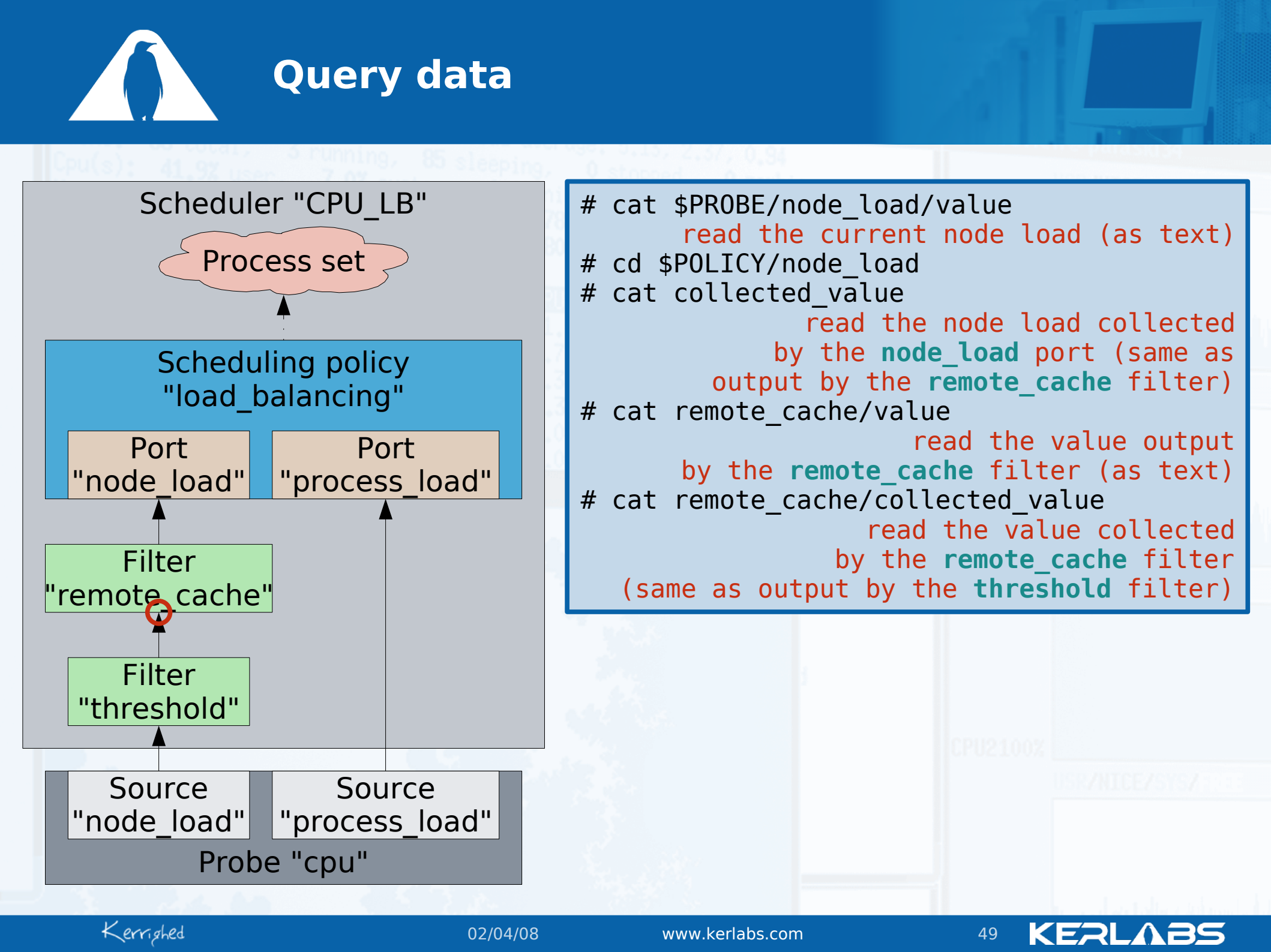
Query data


```
# cat $PROBE/node_load/value
        read the current node load (as text)
# cd $POLICY/node_load
# cat collected_value
                read the node load collected
             by the node_load port (same as
          output by the remote_cache filter)
```



Query data

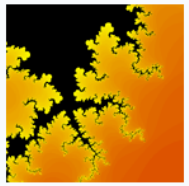

```
# cat $PROBE/node_load/value
        read the current node load (as text)
# cd $POLICY/node_load
# cat collected_value
                read the node load collected
              by the node_load port (same as
          output by the remote_cache filter)
# cat remote_cache/value
                          read the value output
        by the remote_cache filter (as text)
```



Query data


```
# cat $PROBE/node_load/value
        read the current node load (as text)
# cd $POLICY/node_load
# cat collected_value
                read the node load collected
              by the node_load port (same as
          output by the remote_cache filter)
# cat remote_cache/value
                           read the value output
        by the remote_cache filter (as text)
# cat remote_cache/collected_value
                        read the value collected
                     by the remote_cache filter
   (same as output by the threshold filter)
```

Outline



- Rationale
- Big picture
- Example
- User-level interface
- Kernel-level API (samples)
- Status

Main object types

- `struct scheduler` (not managed by programmers)
- `struct process_set` (not managed by programmers)
- `struct sched_policy`, `struct sched_policy_type`
- `struct scheduler_port`, `struct scheduler_port_type`
- `struct scheduler_filter`,
  `struct scheduler_filter_type`
- `struct probe`, `struct probe_type`
  `struct probe_source`, `struct probe_source_type`

Features

- Non-blocking, `kmalloc`-free data/event flows
  - Data/event handling functions **must** be **non-blocking**
  - Memory allocations are done when creating/configuring components
- Object-oriented handling of multiple instances (scheduling policies, ports, filters)
- Typed data
- Remote data access
- Process set local iterators

Scheduling policies
Types

```
#include <scheduler/configfs-sched-policy.h>

struct sched_policy {
    ... /* framework internal */
    spinlock_t lock; /* convenience lock for the programmer */
    ... /* framework internal */
};

struct sched_policy_operations {
    /* sched policy constructor */
    struct sched_policy * (*new)(const char *name);
    /* sched policy destructor */
    void (*destroy)(struct sched_policy *policy);
    /* process placement function called when
     * a task attached to this policy creates a new task */
    kerrighed_node_t (*new_task_node)(struct sched_policy *policy,
                                      struct task_struct *parent);
};

struct sched_policy_type {
    /* framework internal */
    ...
};
```

Scheduling policies Example

- Example in modules/scheduler/modules/echo_policy.c

```
#include <scheduler/configfs-sched-policy.h>

struct echo_policy {
    struct sched_policy policy;
    /* Other attributes */
};

static struct sched_policy * echo_policy_new(const char *name);
static void echo_policy_destroy(struct sched_policy *policy);

static struct sched_policy_operations echo_policy_ops = {
    .new = echo_policy_new,
    .destroy = echo_policy_destroy,
    /* .new_task_node = NULL */
};

static SCHED_POLICY_TYPE(echo_policy_type, "echo_policy",
                         &echo_policy_ops, NULL);
```

Scheduling policies Example

```
static struct sched_policy * echo_policy_new(const char *name)
{
    struct echo_policy *p;
    int err;

    p = kmalloc(sizeof(*p), GFP_KERNEL);
    if (!p) goto err_echo_policy;
    err = krg_sched_policy_init(&p->policy, name, &echo_policy_type,
                                NULL);
    if (err) goto err_policy;
    return &p->policy;

err_policy:
    kfree(p);
err_echo_policy:
    return NULL;
}

static void echo_policy_destroy(struct sched_policy *policy)
{
    struct echo_policy *p;
    p = container_of(policy, struct echo_policy, policy);
    krg_sched_policy_cleanup(policy);
    kfree(p);
}
```

Scheduling policies Example

```
int init_module(void)
{
    return krg_sched_policy_type_register(&echo_policy_type);
}

void cleanup_module(void)
{
    krg_sched_policy_type_unregister(&echo_policy_type);
}
```

Parsing the local processes of a process set

```
#include <scheduler/configfs-schedulers.h>
#include <scheduler/configfs-process-set.h>

void some_function(struct sched_policy *policy) {
    struct scheduler *s;
    struct process_set *pset;
    struct task_struct *p;

    s = sched_policy_get_scheduler(policy);
    if (s) {
        pset = scheduler_get_process_set(s);
        if (pset) {
            process_set_lock(pset);
            process_set_prepare_do_each_process(pset);
            process_set_do_each_process(p, pset) {
                ...
            } process_set_while_each_process(p, pset);
            ...
            process_set_cleanup_do_each_process(pset);
            process_set_unlock(pset);
            process_set_put(pset);
        }
        scheduler_put(s);
    }
}
```

Adding a port

```
#include <scheduler/configfs-port.h>

struct echo_policy {
    struct sched_policy policy;
    struct scheduler_port my_port;
    /* Other attributes */
};

DEFINE_SCHEDULER_PORT_UPDATE_VALUE(my_port)
{
    unsigned long value;

    if (scheduler_port_get_value(my_port, &value, 1, NULL, 0) > 0) {
                printk(KERN_INFO "echo_policy: value=%lu\n", value);
    }
}

static BEGIN_SCHEDULER_PORT_TYPE(my_port),
    .SCHEDULER_PORT_UPDATE_VALUE(my_port),
    .SCHEDULER_PORT_VALUE_TYPE(my_port, unsigned long),
END_SCHEDULER_PORT_TYPE(my_port);
```

Initializing a port

```
static struct sched_policy * echo_policy_new(const char *name)
{
    struct echo_policy *p;
    struct config_group *def_groups[2];
    int err;

    ...
    err = scheduler_port_init(&p->my_port, “my_port”, &my_port_type,
                              NULL, NULL);
    if (err) goto err_port;
    def_groups[0] = scheduler_port_config_group(&p->my_port);
    def_groups[1] = NULL;
    err = krg_sched_policy_init(&p->policy, name, &echo_policy_type,
                                def_groups);
    ...
}

int init_module(void)
{
    ...
    err = scheduler_port_type_init(&my_port_type, NULL);
    ...
}
```

Probes

```
#include <scheduler/configfs-probe.h>

unsigned long values[NR_CPUS];

DEFINE_PROBE_SOURCE_GET(my_value,          /* Probe source name*/
                        unsigned long,     /* Type of value output */
                        value_p,           /* Array of values to fill */
                        nr)                /* max #values to return */
{
    int i;
    for (i = 0; i < max(nr, NR_CPUS); i++) value_p[i] = values[i];
    return i;                              /* #values returned */
}

DEFINE_PROBE_SOURCE_SHOW(my_value,         /* Probe source name */
                         buf)              /* 4KB buffer */
{
    ssize_t count = 0, ssize_t tmp_count = 0;
    int i;
    for (i = 0; i < NR_CPUS; count += tmp_count, i++) {
        tmp_count = snprintf(buf, 4096 – count, “%lu\n”, values[i]);
        if (tmp_count < 0) return tmp_count;
    }
    return min(count + 1, 4096);
}
```

Probes (2)

```
DEFINE_PROBE_SOURCE_HAS_CHANGED(my_value)
{
    return 1;  /* Periodically notify updates, even if no changes */
}

static BEGIN_PROBE_SOURCE_TYPE(my_value),
    .PROBE_SOURCE_GET(my_value),
    .PROBE_SOURCE_SHOW(my_value),
    .PROBE_SOURCE_VALUE_TYPE(my_value, unsigned long),
    .PROBE_SOURCE_HAS_CHANGED(my_value),
END_PROBE_SOURCE_TYPE(my_value);

static struct probe_source *my_probe_sources[2];

static void my_probe_refresh(void)
{
    /* Update my_values[] */
}

static PROBE_TYPE(my_probe, NULL, my_probe_refresh);
```

Probes initialization

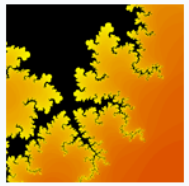
```
static struct probe *my_probe;

int init_module(void)
{
    int err = -ENOMEM;

    my_probe_sources[0] = krg_probe_source_create(&my_value_type,
                                                  “my_value”);
    if (!my_probe_sources[0]) goto err_my_value;
    my_probe_sources[1] = NULL;
    my_probe = krg_probe_create(&my_probe_type, “my_probe”,
                                my_probe_sources);
    if (!my_probe) goto err_my_probe;
    err = krg_probe_register(my_probe);
    if (err) goto err_register;
    return 0;

err_register:
    krg_probe_free(my_probe);
err_my_probe:
    krg_probe_source_free(my_probe_sources[0]);
err_my_value:
    return err;
}
```

Outline



- Rationale
- Big picture
- Example
- User-level interface
- Kernel-level API (samples)
- Status

Status

- Beta version in `devel-sched` branch
- Sample schedulers
  - Toy components
    - `echo_policy`, `cpu_probe`, `mem_probe`
  - Modular MOSIX-like migration-based load balancer
    - split into 5 components
    - still a few patches to commit
  - Round robin new task placement

  ➔Strictly more features than the legacy hard-coded scheduler
- Benchmarking in progress
- Need **reviews** and **beta testers**

Future work

- Few API simplifications
- Improve userland API
- Node reconfiguration support
  - Node addition
- Automatic scheduler creation upon **execve**()
- What do *you* want?

This is the end...

Thank you for your attention!

Security

- Only root can change the configuration
- Components having incompatible types cannot be connected
  - Limits buffer overflows to kernel programming errors