Logging

"VProc sleeping"

We enrich vproc sleeping by extending the wake event with a cause argument. The cause argument indicates the reason that the vproc woke from sleeping. There are four possibilities:

- remote - Either another vproc or the "ping" thread signaled the host vproc.
- time - The time limit expired.
- interrupt - The OS sent an interrupt to pthread corresponding to the host vproc.
- error - An error occurred that is related to the OS locks and synchronization variables.

{ "name" : "VProcWakeup",
  "args" : [ ]
}

"Work stealing"

Each work group receives a unique ID.

{ "name" : "WSInit",
  "args" : [ ]
}

The per-vproc worker-initialization event is dependent on its enclosing work group.

{ "name" : "WSWorkerInit",
  "args" : [ ]
}

The following two events indicate a worker executing work items and a busy worker being preempted, respectively.

{ "name" : "WSExecute",
  "args" : [ ]
}

{ "name" : "WSPreempted",
  "args" : [ ]
}

Below we introduce five events to mark the progress of the thief. The first occurs when the thief worker sends a its thief liaison process to a randomly-chosen victim. The second and third events mark the interval when the thief process executes on the victim vproc. The last two events indicate when the thief receives notification on the result of the steal.

{ "name" : "WSThiefSend",
  "args" : [ ]
}
When a thief has exceeded its maximum duration of consecutive steal attempts, the thief goes to sleep for a while. We do this to avoid flooding workers with steal attempts.

The following group defines some state transitions and dependencies specific to the thief.
Finally, we define the state transitions that workers may make. Each worker is always in one of the four states:

- Inactive - Either uninitialized, terminated, or waiting to be scheduled.
- Busy - Working productively on part of the computation.
- Thief - Attempting to steal work items.
- Sleeping - Doing nothing for a while to avoid flooding busy workers with steal attempts.

The following group defines the allowable state transitions.

```
Unknown macro: { "desc" }
```

**"Ropes"**

The primary event of interest related to ropes is that of rebalancing, since excessive rebalancing might be the cause of a rather subtle performance bug. We therefore introduce two rebalances events to denote the start and end point of the rebalancing. Both events carry the length of the rope being rebalanced.

```
{ "name" : "RopeRebalanceBegin",
  "args" : [ ]
}
```

```
{ "name" : "RopeRebalanceEnd",
  "args" : [ ]
}
```