There may be some different mapping possibilities.

Mapping directive always takes at least one argument: the object to map. Following arguments are keyword arguments (i.e. they are named) that may be specified in any order. Their meaning is tied to what kind of object you are mapping, and how you are mapping it.

Some mapping arguments may be optional.

Let's explore mappable objects, and how they may be deployed.

Tasks

as Software Task

- Mandatory arguments:
    - `run`: A Cpu to run task on
    - `stack`: A memory Segment to put stack in. Stack does not grow dynamically. This should be a segment of cacheable memory for performance reasons.
    - `desc`: A memory Segment to put the task's description in. This description won't be written to. Segment may be read-only memory.
    - `status`: A memory Segment to put dynamic status on. This must be uncached in order to be able to use inter-cpu communications.
- Optional arguments:
    - `code`: A memory segment to copy task code in. This won't be written to. If not specified, code will be placed in global code segment, specified when mapping OS objects.
- Example:

```
mapper.map( "cons0",
            run = mapper.hard.processor,
            stack = mapper.hard.cram0,
            desc = mapper.hard.cram0,
            status = mapper.hard.uram0,
            code = mapper.hard.cram0)
```

as Coprocessor

An `HwTask()` implementation must exist for the task in order to be able to map it as a coprocessor.

- Mandatory arguments:
    - `vci`: A VCI interconnect to connect MwMr controller to.
- Example:

```
mapper.map( "cons0",
            vci = mapper.hard.vgmn)
```

MwMr channels

- Mandatory arguments:
  - `lock`: A Locks dedicated hardware component, like SocLib's Locks component. Uses one lock.
  - `desc`: A memory Segment to put the channel's description in. This description won't be written to. Segment may be read-only memory.
  - `status`: A memory Segment to put dynamic status and buffer in. This must be cacheable. Software Mwmr protocol implémentations do explicit cachelines flushing.
- Example:

```
mapper.map( "fifo",
            lock    = mapper.hard.locks,
            status  = mapper.hard.cram1,
            desc    = mapper.hard.cram1)
```

Processor

- Mandatory arguments:
  - `desc`: A memory Segment to put the processor's description in. This description won't be written to. Segment may be read-only memory.
  - `status`: A memory Segment to put dynamic status in. This must be uncached as it is used to implement inter-cpu communication.
  - `priv`: A memory Segment to put the processor's private data in. This should be cacheable.
- Example:

```
mapper.map("processor",
            desc    = mapper.hard.cram0,
            priv    = mapper.hard.cram0,
            status  = mapper.hard.uram0)
```

OS objects

- Mandatory arguments:
  - `desc`: A memory Segment to put the os' description in. This description won't be written to. Segment may be read-only memory.
  - `shared`: A memory Segment to put os' global data in. This must be uncached as it is used to implement inter-process communication.
  - `code`: A memory Segment to put all unplaced executable code in. This should be cacheable and may be read-only. If you specify a list of segments here, code will be replicated on each segment and closest copy will be used by CPUs.
- Example:

```
mapper.map(mapper.tcg,
            desc    = mapper.hard.cram1,
            shared  = mapper.hard.uram1,
            code    = mapper.hard.cram1)
```