Each CPU has a "base" scheduling domain (struct sched_domain). These are accessed via cpu_sched_domain(i) and this_sched_domain() macros. The domain hierarchy is built from these base domains via the ->parent pointer. ->parent MUST be NULL terminated, and domain structures should be per-CPU as they are locklessly updated.

Each scheduling domain spans a number of CPUs (stored in the ->span field). A domain's span MUST be a superset of it child's span (this restriction could be relaxed if the need arises), and a base domain for CPU i MUST span at least i. The top domain for each CPU will generally span all CPUs in the system although strictly it doesn't have to, but this could lead to a case where some CPUs will never be given tasks to run unless the CPUs allowed mask is explicitly set. A sched domain's span means "balance process load among these CPUs".

Each scheduling domain must have one or more CPU groups (struct sched_group) which are organised as a circular one way linked list from the ->groups pointer. The union of cpumasks of these groups MUST be the same as the domain's span. The intersection of cpumasks from any two of these groups MUST be the empty set. The group pointed to by the ->groups pointer MUST contain the CPU to which the domain belongs. Groups may be shared among CPUs as they contain read only data after they have been set up.

Balancing within a sched domain occurs between groups. That is, each group is treated as one entity. The load of a group is defined as the sum of the load of each of its member CPUs, and only when the load of a group becomes out of balance are tasks moved between groups.

In kernel/sched.c, rebalance_tick is run periodically on each CPU. This function takes its CPU's base sched domain and checks to see if has reached its rebalance interval. If so, then it will run load_balance on that domain. rebalance_tick then checks the parent sched_domain (if it exists), and the parent of the parent and so forth.

*** Implementing sched domains ***
The "base" domain will "span" the first level of the hierarchy. In the case of SMT, you'll span all siblings of the physical CPU, with each group being a single virtual CPU.

In SMP, the parent of the base domain will span all physical CPUs in the node. Each group being a single physical CPU. Then with NUMA, the parent of the SMP domain will span the entire machine, with each group having the cpumask of a node. Or, you could do multi-level NUMA or Opteron, for example, might have just one domain covering its one NUMA level.

The implementor should read comments in include/linux/sched.h: struct sched_domain fields, SD_FLAG_*, SD_*_INIT to get an idea of the specifics and what to tune.

For SMT, the architecture must define CONFIG_SCHED_SMT and provide a cpumask_t cpu_sibling_map[NR_CPUS], where cpu_sibling_map[i] is the mask of all "i"'s siblings as well as "i" itself.

Architectures may retain the regular override the default SD_*_INIT flags while using the generic domain builder in kernel/sched.c if they wish to retain the traditional SMT->SMP->NUMA topology (or some subset of that). This can be done by #define'ing ARCH_HASH_SCHED_TUNE.

Alternatively, the architecture may completely override the generic domain builder by #define'ing ARCH_HASH_SCHED_DOMAIN, and exporting your arch_init_sched_domains function. This function will attach domains to all CPUs using cpu_attach_domain.

The sched-domains debugging infrastructure can be enabled by enabling CONFIG_SCHED_DEBUG. This enables an error checking parse of the sched domains which should catch most possible errors (described above). It also prints out the domain structure in a visual format.