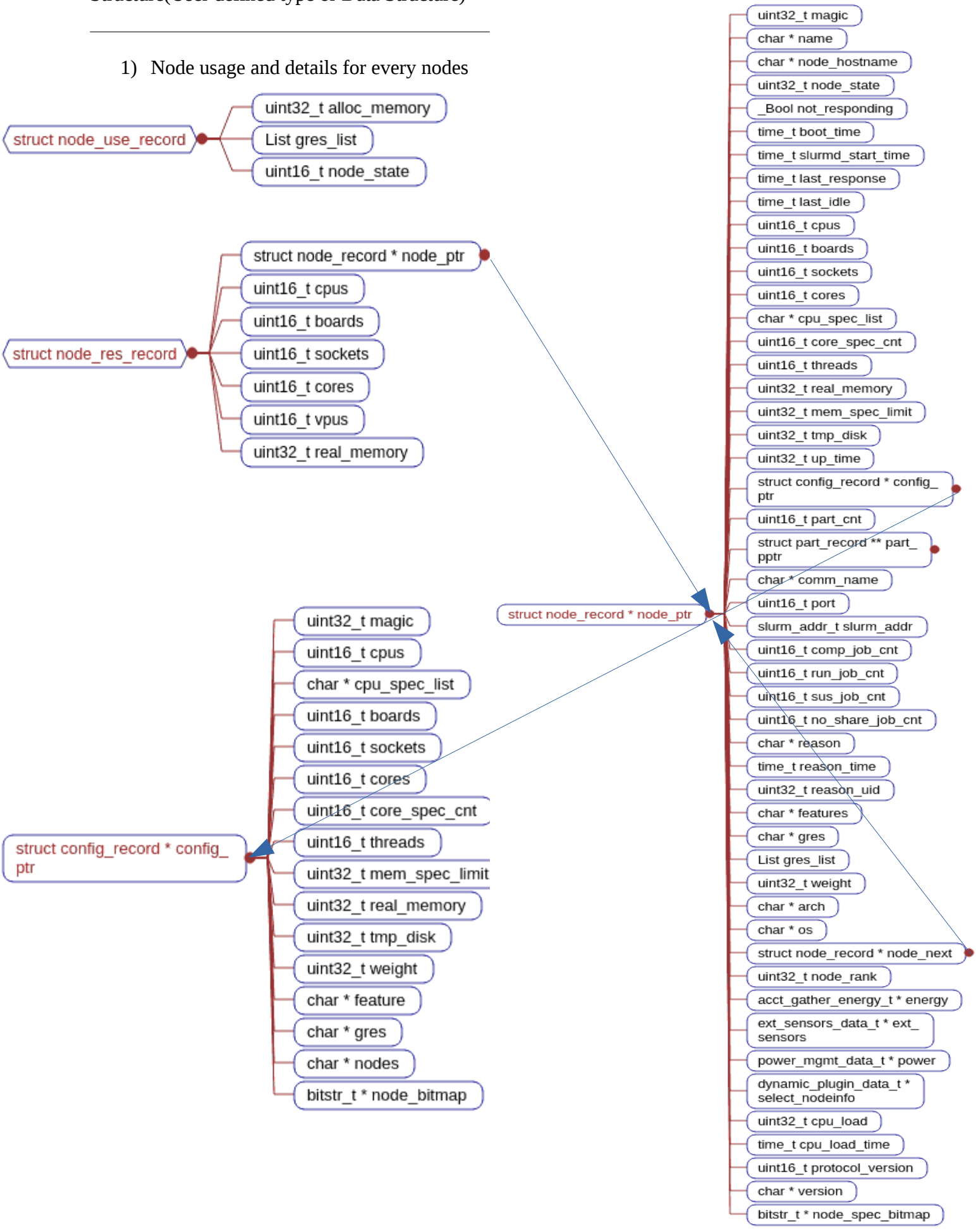
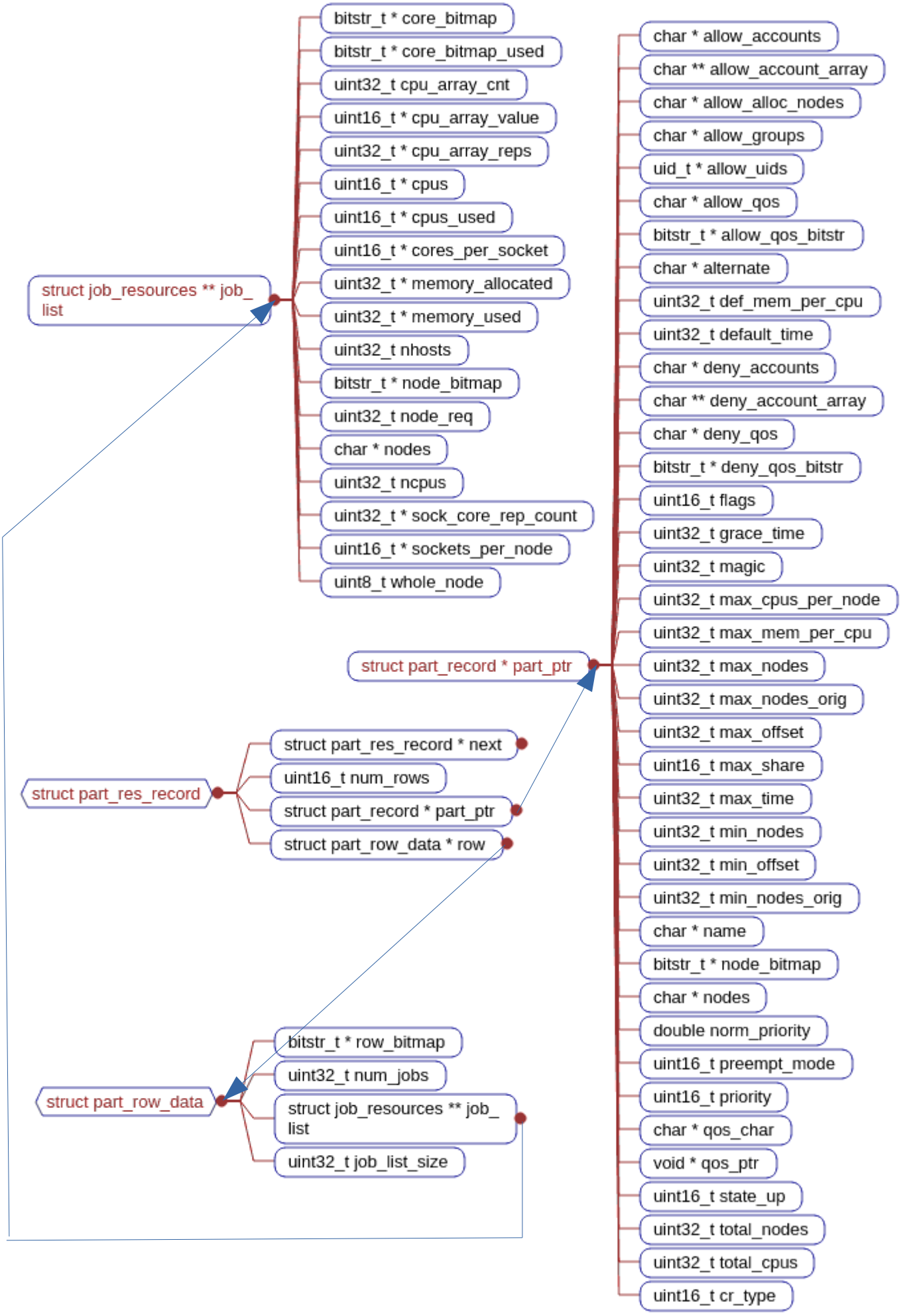
Structure(User defined type or Data Structure)

---

1) Node usage and details for every nodes



2) Partition and Job_resources details



```
/* struct job_resources defines exactly which resources are allocated
 *        to a job, step, partition, etc.
 *
 * core_bitmap                - Bitmap of allocated cores for all nodes and sockets
 * core_bitmap_used  - Bitmap of cores allocated to job steps
 * cores_per_socket   - Count of cores per socket on this node, build by
 *                      build_job_resources() and insures consistent
 *                      interpretation of core_bitmap
 * cpus               - Count of desired/allocated CPUs per node for job/step
 * cpus_used          - For a job, count of CPUs per node used by job steps
 * cpu_array_cnt      - Count of elements in cpu_array_* below
 * cpu_array_value    - Count of allocated CPUs per node for job
 * cpu_array_reps     - Number of consecutive nodes on which cpu_array_value
 *                      is duplicated. See NOTES below.
 * memory_allocated  - MB per node reserved for the job or step
 * memory_used                - MB per node of memory consumed by job steps
 * nhosts             - Number of nodes in the allocation.  On a
 *                     bluegene machine this represents the number
 *                     of midplanes used.  This should always be
 *                     the number of bits set in node_bitmap.
 * node_bitmap        - Bitmap of nodes allocated to the job. Unlike the
 *                      node_bitmap in slurmctld's job record, the bits
 *                      here do NOT get cleared as the job completes on a
 *                      node
 * node_req           - NODE_CR_RESERVED|NODE_CR_ONE_ROW|
NODE_CR_AVAILABLE
 * nodes              - Names of nodes in original job allocation
 * ncpus              - Number of processors in the allocation
 * sock_core_rep_count        - How many consecutive nodes that sockets_per_node
 *                      and cores_per_socket apply to, build by
 *                      build_job_resources() and insures consistent
 *                      interpretation of core_bitmap
 * sockets_per_node   - Count of sockets on this node, build by
 *                      build_job_resources() and insures consistent
 *                      interpretation of core_bitmap
 * whole_node         - Job allocated full node (used only by select/cons_res)
 *
 * NOTES:
 * cpu_array_* contains the same information as "cpus", but in a more compact
 * format. For example if cpus = {4, 4, 2, 2, 2, 2, 2, 2} then cpu_array_cnt=2
 * cpu_array_value = {4, 2} and cpu_array_reps = {2, 6}. We do not need to
 * save/restore these values, but generate them by calling
 * build_job_resources_cpu_array()
 *
 * Sample layout of core_bitmap:
 *   |            Node_0                    |            Node_1                    |
 *   |     Sock_0        |     Sock_1        |     Sock_0        |     Sock_1       |
 *   | Core_0 | Core_1 | Core_0 | Core_1 | Core_0 | Core_1  | Core_0 | Core_1     |
 *   | Bit_0   | Bit_1   | Bit_2   | Bit_3   | Bit_4   | Bit_5    | Bit_6    | Bit_7       |
 *
 * If a job changes size (reliquishes nodes), the node_bitmap will remain
 * unchanged, but cpus, cpus_used, cpus_array_*, and memory_used will be
```

```
 * updated (e.g. cpus and mem_used on that node cleared).
 */
```
struct job_resources

Node-0				Node-1			
Sock-0		Sock-1		Sock-2		Sock-3	
Core-0	Core-1	Core-2	Core-3	Core-4	Core-5	Core-6	Core-7

```
/*
 * node_res_record.node_state assists with the unique state of each node.
 * When a job is allocated, these flags provide protection for nodes in a
 * Shared=NO or Shared=EXCLUSIVE partition from other jobs.
 *
 * NOTES:
 * - If node is in use by Shared=NO part, some CPUs/memory may be available
 * - Caution with NODE_CR_AVAILABLE: a Sharing partition could be full.
 *
 * - these values are staggered so that they can be incremented as multiple
 *   jobs are allocated to each node. This is needed to be able to support
 *   preemption, which can override these protections.
 */
enum node_cr_state {
        NODE_CR_AVAILABLE = 0,    /* The node may be IDLE or IN USE (shared) */
        NODE_CR_ONE_ROW = 1,      /* node is in use by Shared=NO part */
        NODE_CR_RESERVED = 64000  /* node is in use by Shared=EXCLUSIVE part */
};

/* job_details - specification of a job's constraints,
 * can be purged after initiation */
struct job_details {
        char *acctg_freq;               /* accounting polling interval */
        uint32_t argc;                  /* count of argv elements */
        char **argv;                    /* arguments for a batch job script */
        time_t begin_time;              /* start at this time (srun --begin),
                                         * resets to time first eligible
                                         * (all dependencies satisfied) */
        char *ckpt_dir;                         /* directory to store checkpoint
                                         * images */
        uint16_t contiguous;            /* set if requires contiguous nodes */
        uint16_t core_spec;             /* specialized core count */
        char *cpu_bind;                         /* binding map for map/mask_cpu - This
                                         * currently does not matter to the
                                         * job allocation, setting this does
                                         * not do anything for steps. */
        uint16_t cpu_bind_type;                 /* see cpu_bind_type_t - This
                                         * currently does not matter to the
                                         * job allocation, setting this does
```

```
                         * not do anything for steps. */
uint32_t cpu_freq_min;   /* Minimum cpu frequency  */
uint32_t cpu_freq_max;   /* Maximum cpu frequency  */
uint32_t cpu_freq_gov;   /* cpu frequency governor */
uint16_t cpus_per_task;          /* number of processors required for
                         * each task */
List depend_list;        /* list of job_ptr:state pairs */
char *dependency;        /* wait for other jobs */
char *orig_dependency;           /* original value (for archiving) */
uint16_t env_cnt;        /* size of env_sup (see below) */
char **env_sup;                  /* supplemental environment variables
                         * as set by Moab */
bitstr_t *exc_node_bitmap; /* bitmap of excluded nodes */
char *exc_nodes;         /* excluded nodes */
uint32_t expanding_jobid; /* ID of job to be expanded */
List feature_list;       /* required features with
                         * node counts */
char *features;                  /* required features */
uint32_t magic;                  /* magic cookie for data integrity */
uint32_t max_cpus;       /* maximum number of cpus */
uint32_t max_nodes;      /* maximum number of nodes */
multi_core_data_t *mc_ptr; /* multi-core specific data */
char *mem_bind;                  /* binding map for map/mask_cpu */
uint16_t mem_bind_type;          /* see mem_bind_type_t */
uint32_t min_cpus;       /* minimum number of cpus */
uint32_t min_nodes;      /* minimum number of nodes */
uint16_t nice;           /* requested priority change,
                         * NICE_OFFSET == no change */
uint16_t ntasks_per_node; /* number of tasks on each node */
uint32_t num_tasks;      /* number of tasks to start */
uint8_t open_mode;       /* stdout/err append or trunctate */
uint8_t overcommit;      /* processors being over subscribed */
uint16_t plane_size;     /* plane size when task_dist =
                         * SLURM_DIST_PLANE */
/* job constraints: */
uint32_t pn_min_cpus;            /* minimum processors per node */
uint32_t pn_min_memory;          /* minimum memory per node (MB) OR
                         * memory per allocated
                         * CPU | MEM_PER_CPU */
uint32_t pn_min_tmp_disk; /* minimum tempdisk per node, MB */
uint8_t prolog_running;          /* set while prolog_slurmctld is
                         * running */
uint32_t reserved_resources; /* CPU minutes of resources reserved
                         * for this job while it was pending */
bitstr_t *req_node_bitmap; /* bitmap of required nodes */
uint16_t *req_node_layout; /* task layout for required nodes */
time_t preempt_start_time; /* time that preeption began to start
                         * this job */
char *req_nodes;         /* required nodes */
uint16_t requeue;        /* controls ability requeue job */
char *restart_dir;       /* restart execution from ckpt images
                         * in this dir */
```

```
        uint8_t share_res;          /* set if job can share resources with
                                     * other jobs */
        char *std_err;              /* pathname of job's stderr file */
        char *std_in;               /* pathname of job's stdin file */
        char *std_out;              /* pathname of job's stdout file */
        time_t submit_time;         /* time of submission */
        uint16_t task_dist;         /* task layout for this job. Only
                                     * useful when Consumable Resources
                                     * is enabled */
        uint32_t usable_nodes;              /* node count needed by preemption */
        uint8_t whole_node;         /* job requested exclusive node use */
        char *work_dir;                     /* pathname of working directory */
};

typedef struct job_array_struct {
        uint32_t task_cnt;          /* count of remaining task IDs */
        bitstr_t *task_id_bitmap;   /* bitmap of remaining task IDs */
        char *task_id_str;          /* string describing remaining task IDs,
                                     * needs to be recalcuated if NULL */
        uint32_t array_flags;       /* Flags to control behavior (FUTURE) */
        uint32_t max_run_tasks;             /* Maximum number of running tasks */
        uint32_t tot_run_tasks;             /* Current running task count */
        uint32_t min_exit_code;             /* Minimum exit code from any task */
        uint32_t max_exit_code;             /* Maximum exit code from any task */
        uint32_t tot_comp_tasks;    /* Completed task count */
} job_array_struct_t;

/*
 * NOTE: When adding fields to the job_record, or any underlying structures,
 * be sure to sync with _rec_job_copy.
 */
struct job_record {
        char    *account;           /* account number to charge */
        char     *alias_list;       /* node name to address aliases */
        char    *alloc_node;        /* local node making resource alloc */
        uint16_t alloc_resp_port;   /* RESPONSE_RESOURCE_ALLOCATION port */
        uint32_t alloc_sid;         /* local sid making resource alloc */
        uint32_t array_job_id;              /* job_id of a job array or 0 if N/A */
        uint32_t array_task_id;             /* task_id of a job array */
        job_array_struct_t *array_recs;     /* job array details,
                                     * only in meta-job record */
        uint32_t assoc_id;          /* used for accounting plugins */
        void    *assoc_ptr;         /* job's assoc record ptr, it is
                                     * void* because of interdependencies
                                     * in the header files, confirm the
                                     * value before use */
        uint16_t batch_flag;        /* 1 or 2 if batch job (with script),
                                     * 2 indicates retry mode (one retry) */
        char *batch_host;           /* host executing batch script */
        char *burst_buffer;         /* burst buffer specification */
        check_jobinfo_t check_job;  /* checkpoint context, opaque */
        uint16_t ckpt_interval;             /* checkpoint interval in minutes */
```

```
time_t ckpt_time;              /* last time job was periodically
                                * checkpointed */
char *comment;                        /* arbitrary comment */
uint32_t cpu_cnt;              /* current count of CPUs held
                                * by the job, decremented while job is
                                * completing (N/A for bluegene
                                * systems) */
uint16_t cr_enabled;           /* specify if if Consumable Resources
                                * is enabled. Needed since CR deals
                                * with a finer granularity in its
                                * node/cpu scheduling (available cpus
                                * instead of available nodes) than the
                                * bluegene and the linear plugins
                                * 0 if cr is NOT enabled,
                                * 1 if cr is enabled */
uint32_t db_index;             /* used only for database
                                * plugins */
uint32_t derived_ec;           /* highest exit code of all job steps */
struct job_details *details;   /* job details */
uint16_t direct_set_prio;      /* Priority set directly if
                                * set the system will not
                                * change the priority any further. */
time_t end_time;               /* time execution ended, actual or
                                * expected. if terminated from suspend
                                * state, this is time suspend began */
bool epilog_running;           /* true of EpilogSlurmctld is running */
uint32_t exit_code;            /* exit code for job (status from
                                * wait call) */
front_end_record_t *front_end_ptr; /* Pointer to front-end node running
                                * this job */
char *gres;                    /* generic resources requested by job */
List gres_list;                /* generic resource allocation detail */
char *gres_alloc;              /* Allocated GRES added over all nodes
                                * to be passed to slurmdbd */
char *gres_req;                       /* Requested GRES added over all nodes
                                * to be passed to slurmdbd */
char *gres_used;               /* Actual GRES use added over all nodes
                                * to be passed to slurmdbd */
uint32_t group_id;             /* group submitted under */
uint32_t job_id;               /* job ID */
struct job_record *job_next;  /* next entry with same hash index */
struct job_record *job_array_next_j; /* job array linked list by job_id */
struct job_record *job_array_next_t; /* job array linked list by task_id */
job_resources_t *job_resrcs; /* details of allocated cores */
uint16_t job_state;            /* state of the job */
uint16_t kill_on_node_fail;   /* 1 if job should be killed on
                                * node failure */
char *licenses;                       /* licenses required by the job */
List license_list;             /* structure with license info */
uint16_t limit_set_max_cpus;          /* if max_cpus was set from
                                * a limit false if user set */
uint16_t limit_set_max_nodes;         /* if max_nodes was set from
```

```
                                * a limit false if user set */
uint16_t limit_set_min_cpus;/* if max_cpus was set from
                                * a limit false if user set */
uint16_t limit_set_min_nodes;       /* if max_nodes was set from
                                * a limit false if user set */
uint16_t limit_set_pn_min_memory; /* if pn_min_memory was set from
                                * a limit false if user set */
uint16_t limit_set_time;      /* if time_limit was set from
                                * a limit false if user set */
uint16_t limit_set_qos;             /* if qos_limit was set from
                                * a limit false if user set */
uint16_t mail_type;           /* see MAIL_JOB_* in slurm.h */
char *mail_user;              /* user to get e-mail notification */
uint32_t magic;                     /* magic cookie for data integrity */
char *name;                   /* name of the job */
char *network;                      /* network/switch requirement spec */
uint32_t next_step_id;              /* next step id to be used */
char *nodes;                  /* list of nodes allocated to job */
slurm_addr_t *node_addr;    /* addresses of the nodes allocated to
                                * job */
bitstr_t *node_bitmap;              /* bitmap of nodes allocated to job */
bitstr_t *node_bitmap_cg;   /* bitmap of nodes completing job */
uint32_t node_cnt;          /* count of nodes currently
                                * allocated to job */
uint32_t node_cnt_wag;              /* count of nodes Slurm thinks
                                * will be allocated when the
                                * job is pending and node_cnt
                                * wasn't given by the user.
                                * This is packed in total_nodes
                                * when dumping state.  When
                                * state is read in check for
                                * pending state and set this
                                * instead of total_nodes */
char *nodes_completing;             /* nodes still in completing state
                                * for this job, used to insure
                                * epilog is not re-run for job */
uint16_t other_port;          /* port for client communications */
char *partition;              /* name of job partition(s) */
List part_ptr_list;           /* list of pointers to partition recs */
bool part_nodes_missing;    /* set if job's nodes removed from this
                                * partition */
struct part_record *part_ptr;  /* pointer to the partition record */
uint8_t power_flags;          /* power management flags,
                                * see SLURM_POWER_FLAGS_ */
time_t pre_sus_time;          /* time job ran prior to last suspend */
time_t preempt_time;          /* job preemption signal time */
bool preempt_in_progress;   /* Premption of other jobs in progress
                                * in order to start this job,
                                * (Internal use only, don't save) */
uint32_t priority;            /* relative priority of the job,
                                * zero == held (don't initiate) */
uint32_t *priority_array;     /* partition based priority */
```

```
priority_factors_object_t *prio_factors; /* cached value used
                                          * by sprio command */
uint32_t profile;               /* Acct_gather_profile option */
uint32_t qos_id;                /* quality of service id */
void *qos_ptr;                  /* pointer to the quality of
                                 * service record used for
                                 * this job, it is
                                 * void* because of interdependencies
                                 * in the header files, confirm the
                                 * value before use */
uint8_t reboot;                         /* node reboot requested before start */
uint16_t restart_cnt;           /* count of restarts */
time_t resize_time;             /* time of latest size change */
uint32_t resv_id;               /* reservation ID */
char *resv_name;                /* reservation name */
struct slurmctld_resv *resv_ptr;/* reservation structure pointer */
uint32_t requid;                /* requester user ID */
char *resp_host;                /* host for srun communications */
char *sched_nodes;              /* list of nodes scheduled for job */
dynamic_plugin_data_t *select_jobinfo;/* opaque data, BlueGene */
uint8_t sicp_mode;              /* set for inter-cluster jobs */
char **spank_job_env;                   /* environment variables for job prolog
                                 * and epilog scripts as set by SPANK
                                 * plugins */
uint32_t spank_job_env_size;            /* element count in spank_env */
uint16_t start_protocol_ver;  /* Slurm version step was
                                 * started with */
time_t start_time;              /* time execution begins,
                                 * actual or expected */
char *state_desc;               /* optional details for state_reason */
uint32_t state_reason;          /* reason job still pending or failed
                                 * see slurm.h:enum job_wait_reason */
List step_list;                 /* list of job's steps */
time_t suspend_time;            /* time job last suspended or resumed */
time_t time_last_active;        /* time of last job activity */
uint32_t time_limit;            /* time_limit minutes or INFINITE,
                                 * NO_VAL implies partition max_time */
uint32_t time_min;              /* minimum time_limit minutes or
                                 * INFINITE,
                                 * zero implies same as time_limit */
time_t tot_sus_time;            /* total time in suspend state */
uint32_t total_cpus;            /* number of allocated cpus,
                                 * for accounting */
uint32_t total_nodes;           /* number of allocated nodes
                                 * for accounting */
uint32_t user_id;               /* user the job runs as */
uint16_t wait_all_nodes;        /* if set, wait for all nodes to boot
                                 * before starting the job */
uint16_t warn_flags;            /* flags for signal to send */
uint16_t warn_signal;           /* signal to send before end_time */
uint16_t warn_time;             /* when to send signal before
                                 * end_time (secs) */
```

```
	char *wckey;			/* optional wckey */

	/* Request number of switches support */
	uint32_t req_switch;  /* Minimum number of switches             */
	uint32_t wait4switch; /* Maximum time to wait for minimum switches */
	bool     best_switch; /* true=min number of switches met        */
	time_t wait4switch_start; /* Time started waiting for switch    */
};

/*****************************************************************************\
 *  PARTITION parameters and data structures
\*****************************************************************************/
#define PART_MAGIC 0xaefe8495

struct part_record {
	char *allow_accounts;		/* comma delimited list of accounts,
					 * NULL indicates all */
	char **allow_account_array; /* NULL terminated list of allowed
					 * accounts */
	char *allow_alloc_nodes;/* comma delimited list of allowed
					 * allocating nodes
					 * NULL indicates all */
	char *allow_groups;	/* comma delimited list of groups,
					 * NULL indicates all */
	uid_t *allow_uids;	/* zero terminated list of allowed user IDs */
	char *allow_qos;	/* comma delimited list of qos,
					 * NULL indicates all */
	bitstr_t *allow_qos_bitstr; /* (DON'T PACK) assocaited with
					 * char *allow_qos but used internally */
	char *alternate;	/* name of alternate partition */
	uint32_t def_mem_per_cpu; /* default MB memory per allocated CPU */
	uint32_t default_time;		/* minutes, NO_VAL or INFINITE */
	char *deny_accounts; /* comma delimited list of denied accounts */
	char **deny_account_array; /* NULL terminated list of denied accounts */
	char *deny_qos;			/* comma delimited list of denied qos */
	bitstr_t *deny_qos_bitstr; /* (DON'T PACK) associated with
					 * char *deny_qos but used internallly */
	uint16_t flags;		/* see PART_FLAG_* in slurm.h */
	uint32_t grace_time;  /* default preempt grace time in seconds */
	uint32_t magic;			/* magic cookie to test data integrity */
	uint32_t max_cpus_per_node; /* maximum allocated CPUs per node */
	uint32_t max_mem_per_cpu; /* maximum MB memory per allocated CPU */
	uint32_t max_nodes;  /* per job or INFINITE */
	uint32_t max_nodes_orig;/* unscaled value (c-nodes on BlueGene) */
	uint32_t max_offset;  /* select plugin max offset */
	uint16_t max_share;   /* number of jobs to gang schedule */
	uint32_t max_time;    /* minutes or INFINITE */
	uint32_t min_nodes;   /* per job */
	uint32_t min_offset;   /* select plugin min offset */
	uint32_t min_nodes_orig;/* unscaled value (c-nodes on BlueGene) */
	char *name;		/* name of the partition */
	bitstr_t *node_bitmap;		/* bitmap of nodes in partition */
```

```
        char *nodes;            /* comma delimited list names of nodes */
        double   norm_priority;         /* normalized scheduling priority for
                                 * jobs (DON'T PACK) */
        uint16_t preempt_mode;      /* See PREEMPT_MODE_* in slurm/slurm.h */
        uint16_t priority;      /* scheduling priority for jobs */
        char *qos_char;        /* requested QOS from slurm.conf */
        void *qos_ptr;         /* pointer to the quality of
                                 * service record attached to this
                                 * partition, it is void* because of
                                 * interdependencies in the header
                                 * files, confirm the value before use */
        uint16_t state_up;      /* See PARTITION_* states in slurm.h */
        uint32_t total_nodes;  /* total number of nodes in the partition */
        uint32_t total_cpus;   /* total number of cpus in the partition */
        uint32_t max_cpu_cnt;       /* max # of cpus on a node in the partition */
        uint32_t max_core_cnt;      /* max # of cores on a node in the partition */
        uint16_t cr_type;       /* Custom CR values for partition (if supported by select plugin) */
};

extern List part_list;                  /* list of part_record entries */
extern time_t last_part_update;             /* time of last part_list update */
extern struct part_record default_part;     /* default configuration values */
extern char *default_part_name;             /* name of default partition */
extern struct part_record *default_part_loc; /* default partition ptr */
extern uint16_t part_max_priority;      /* max priority in all partitions */
```

struct config_record {

```
        uint32_t magic;             /* magic cookie to test data integrity */
        uint16_t cpus;          /* count of processors running on the node */
        char *cpu_spec_list;   /* arbitrary list of specialized cpus */
        uint16_t boards;        /* count of boards configured */
        uint16_t sockets;       /* number of sockets per node */
        uint16_t cores;             /* number of cores per CPU */
        uint16_t core_spec_cnt;     /* number of specialized cores */
        uint16_t threads;       /* number of threads per core */
        uint32_t mem_spec_limit; /* MB real memory for memory specialization */
        uint32_t real_memory;       /* MB real memory on the node */
        uint32_t tmp_disk;     /* MB total storage in TMP_FS file system */
        uint32_t weight;        /* arbitrary priority of node for
                                 * scheduling work on */
        char *feature;          /* arbitrary list of node's features */
        char *gres;             /* arbitrary list of node's generic resources */
        char *nodes;            /* name of nodes with this configuration */
        bitstr_t *node_bitmap;      /* bitmap of nodes with this configuration */
};
extern List config_list;        /* list of config_record entries */

extern List front_end_list;     /* list of slurm_conf_frontend_t entries */
```

```
struct node_record {
        uint32_t magic;                 /* magic cookie for data integrity */
        char *name;                     /* name of the node. NULL==defunct */
        char *node_hostname;            /* hostname of the node */
        uint32_t node_state;            /* enum node_states, ORed with
                                         * NODE_STATE_NO_RESPOND if not
                                         * responding */
        bool not_responding;            /* set if fails to respond,
                                         * clear after logging this */
        time_t boot_time;               /* Time of node boot,
                                         * computed from up_time */
        time_t slurmd_start_time;       /* Time of slurmd startup */
        time_t last_response;           /* last response from the node */
        time_t last_idle;               /* time node last become idle */
        uint16_t cpus;                  /* count of processors on the node */
        uint16_t boards;                /* count of boards configured */
        uint16_t sockets;               /* number of sockets per node */
        uint16_t cores;                 /* number of cores per CPU */
        char *cpu_spec_list;            /* node's specialized cpus */
        uint16_t core_spec_cnt;         /* number of specialized cores on node*/
        uint16_t threads;               /* number of threads per core */
        uint32_t real_memory;           /* MB real memory on the node */
        uint32_t mem_spec_limit;        /* MB memory limit for specialization */
        uint32_t tmp_disk;              /* MB total disk in TMP_FS */
        uint32_t up_time;               /* seconds since node boot */
        struct config_record *config_ptr;  /* configuration spec ptr */
        uint16_t part_cnt;              /* number of associated partitions */
        struct part_record **part_pptr;         /* array of pointers to partitions
                                         * associated with this node*/
        char *comm_name;                /* communications path name to node */
        uint16_t port;                  /* TCP port number of the slurmd */
        slurm_addr_t slurm_addr;        /* network address */
        uint16_t comp_job_cnt;          /* count of jobs completing on node */
        uint16_t run_job_cnt;           /* count of jobs running on node */
        uint16_t sus_job_cnt;           /* count of jobs suspended on node */
        uint16_t no_share_job_cnt;      /* count of jobs running that will
                                         * not share nodes */
        char *reason;                   /* why a node is DOWN or DRAINING */
        time_t reason_time;             /* Time stamp when reason was
                                         * set, ignore if no reason is set. */
        uint32_t reason_uid;            /* User that set the reason, ignore if
                                         * no reason is set. */
        char *features;                 /* node's features, used only
                                         * for state save/restore, DO NOT
                                         * use for scheduling purposes */
        char *gres;                     /* node's generic resources, used only
                                         * for state save/restore, DO NOT
                                         * use for scheduling purposes */
        List gres_list;                 /* list of gres state info managed by
                                         * plugins */
        uint32_t weight;                /* orignal weight, used only for state
                                         * save/restore, DO NOT use for
```

```
                                        * scheduling purposes. */
        char *arch;                     /* computer architecture */
        char *os;                       /* operating system now running */
        struct node_record *node_next;      /* next entry with same hash index */
        uint32_t node_rank;             /* Hilbert number based on node name,
                                        * or other sequence number used to
                                        * order nodes by location,
                                        * no need to save/restore */
#ifdef HAVE_ALPS_CRAY
        uint32_t basil_node_id;             /* Cray-XT BASIL node ID,
                                        * no need to save/restore */
        time_t down_time;               /* When first set to DOWN state */
#endif  /* HAVE_ALPS_CRAY */
        acct_gather_energy_t *energy;       /* power consumption data */
        ext_sensors_data_t *ext_sensors; /* external sensor data */
        power_mgmt_data_t *power;/* power management data */
        dynamic_plugin_data_t *select_nodeinfo; /* opaque data structure,
                                                 * use select_g_get_nodeinfo()
                                                 * to access contents */
        uint32_t cpu_load;              /* CPU load * 100 */
        time_t cpu_load_time;               /* Time when cpu_load last set */
        uint16_t protocol_version;      /* Slurm version number */
        char *version;                  /* Slurm version */
        bitstr_t *node_spec_bitmap;  /* node cpu specialization bitmap */
};
extern struct node_record *node_record_table_ptr;  /* ptr to node records */
extern int node_record_count;               /* count in node_record_table_ptr */
extern xhash_t* node_hash_table;    /* hash table for node records */
extern time_t last_node_update;             /* time of last node record update */

extern uint16_t *cr_node_num_cores;
extern uint32_t *cr_node_cores_offset;

/*
 * bitmap2node_name_sortable - given a bitmap, build a list of comma
 *      separated node names. names may include regular expressions
 *      (e.g. "lx[01-10]")
 * IN bitmap - bitmap pointer
 * IN sort   - returned ordered list or not
 * RET pointer to node list or NULL on error
 * globals: node_record_table_ptr - pointer to node table
 * NOTE: the caller must xfree the memory at node_list when no longer required
 */
char * bitmap2node_name_sortable (bitstr_t *bitmap, bool sort);

/*
 * bitmap2node_name - given a bitmap, build a list of comma separated node
 *      names. names may include regular expressions (e.g. "lx[01-10]")
 * IN bitmap - bitmap pointer
 * RET pointer to node list or NULL on error
 * globals: node_record_table_ptr - pointer to node table
 * NOTE: the caller must xfree the memory at node_list when no longer required
```

```
 */
char * bitmap2node_name (bitstr_t *bitmap);

/*****************************************************************************\
 *  Consumable Resources parameters and data structures
\*****************************************************************************/

/*
 * Define the type of update and of data retrieval that can happen
 * from the "select/cons_res" plugin. This information needed to
 * support processors as consumable resources.  This structure will be
 * useful when updating other types of consumable resources as well
*/
enum select_plugindata_info {
        SELECT_CR_PLUGIN,    /* data-> uint32 1 if CR plugin */
        SELECT_BITMAP,       /* Unused since version 2.0 */
        SELECT_ALLOC_CPUS,   /* data-> uint16 alloc cpus (CR support) */
        SELECT_ALLOC_LPS,    /* data-> uint32 alloc lps  (CR support) */
        SELECT_AVAIL_MEMORY, /* data-> uint32 avail mem  (CR support) */
        SELECT_STATIC_PART,  /* data-> uint16, 1 if static partitioning
                              * BlueGene support */
        SELECT_CONFIG_INFO   /* data-> List get .conf info from select
                              * plugin */
} ;
```

job_resource.h,node_conf.h,slurmctld.h,