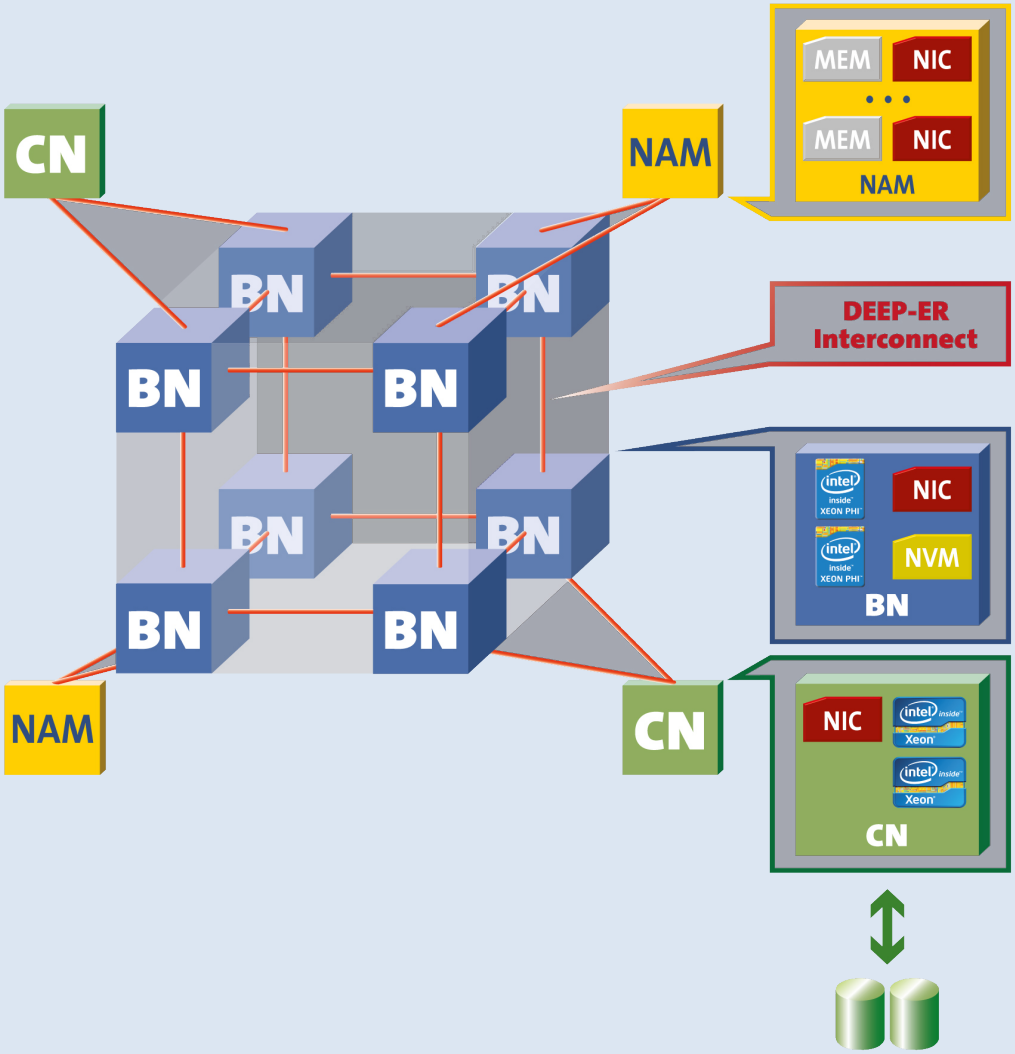
# DEEP-ER Resiliency API

# SCR

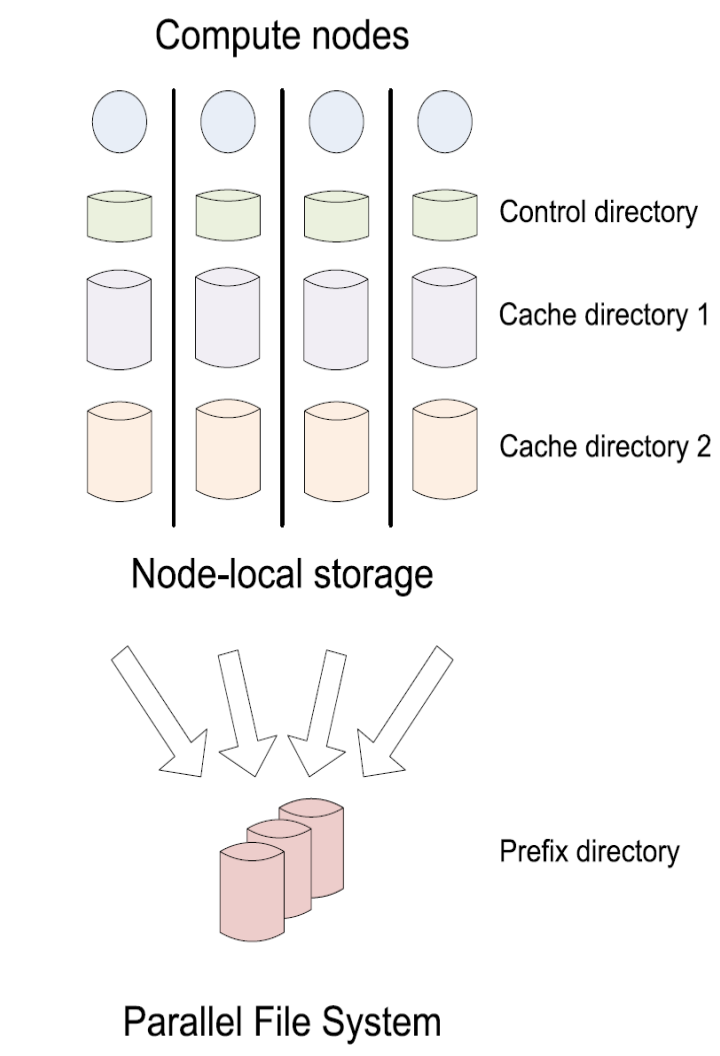
# DEEP-ER Architecture



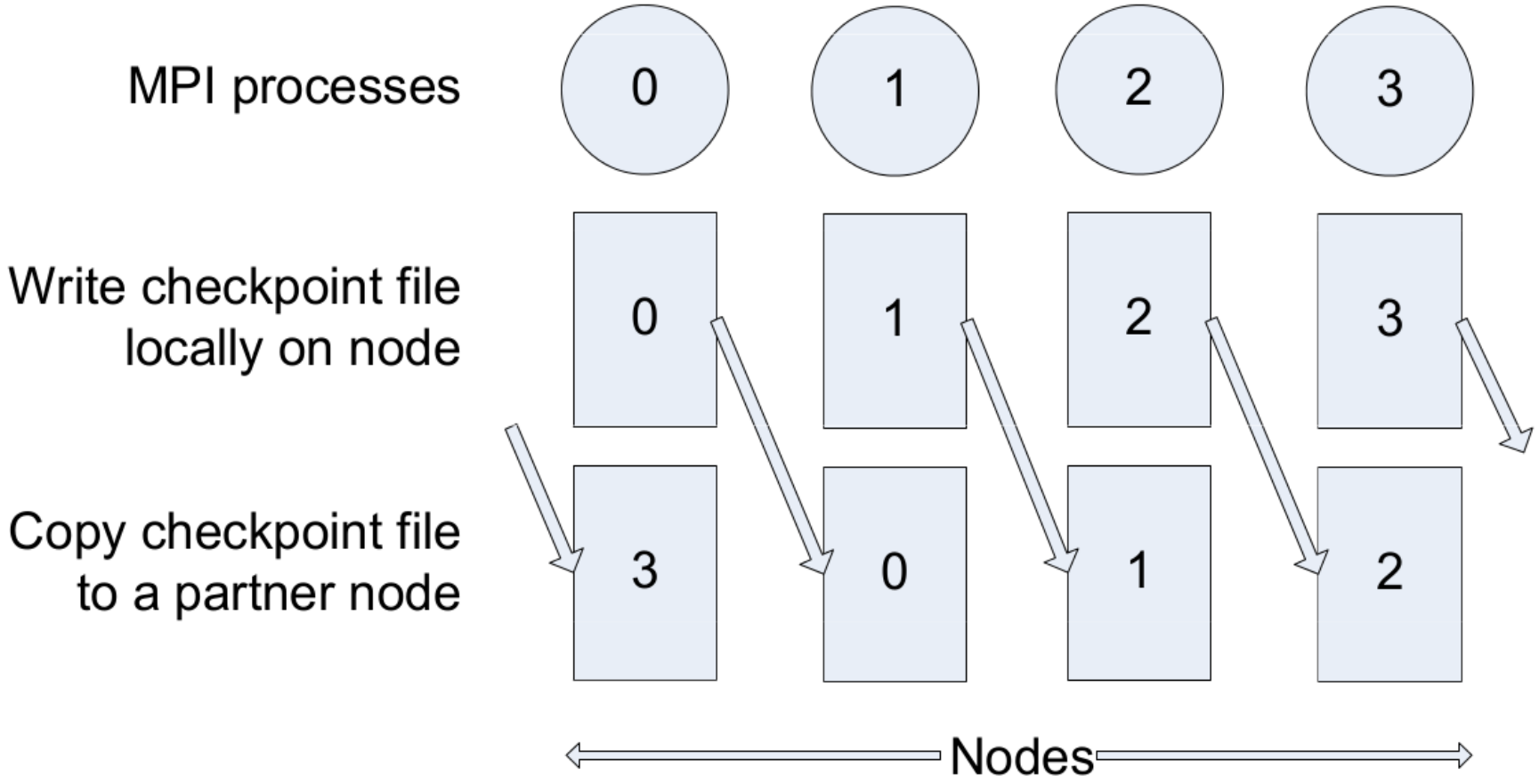
# SCR

- approach based on Scalable Checkpoint Restart Library (SCR)
- Lawrence Livermore National Laboratory (LLNR)
- multi-level checkpointing and redundancy
  - node-local storage
  - RAM-Disk
  - buddy-checkpointing
  - XOR-Files (similar to RAID5)
  - PFS
- synchronous/asynchronous flushes (via daemon process)
- creates + provides file-path to application for writing and reading

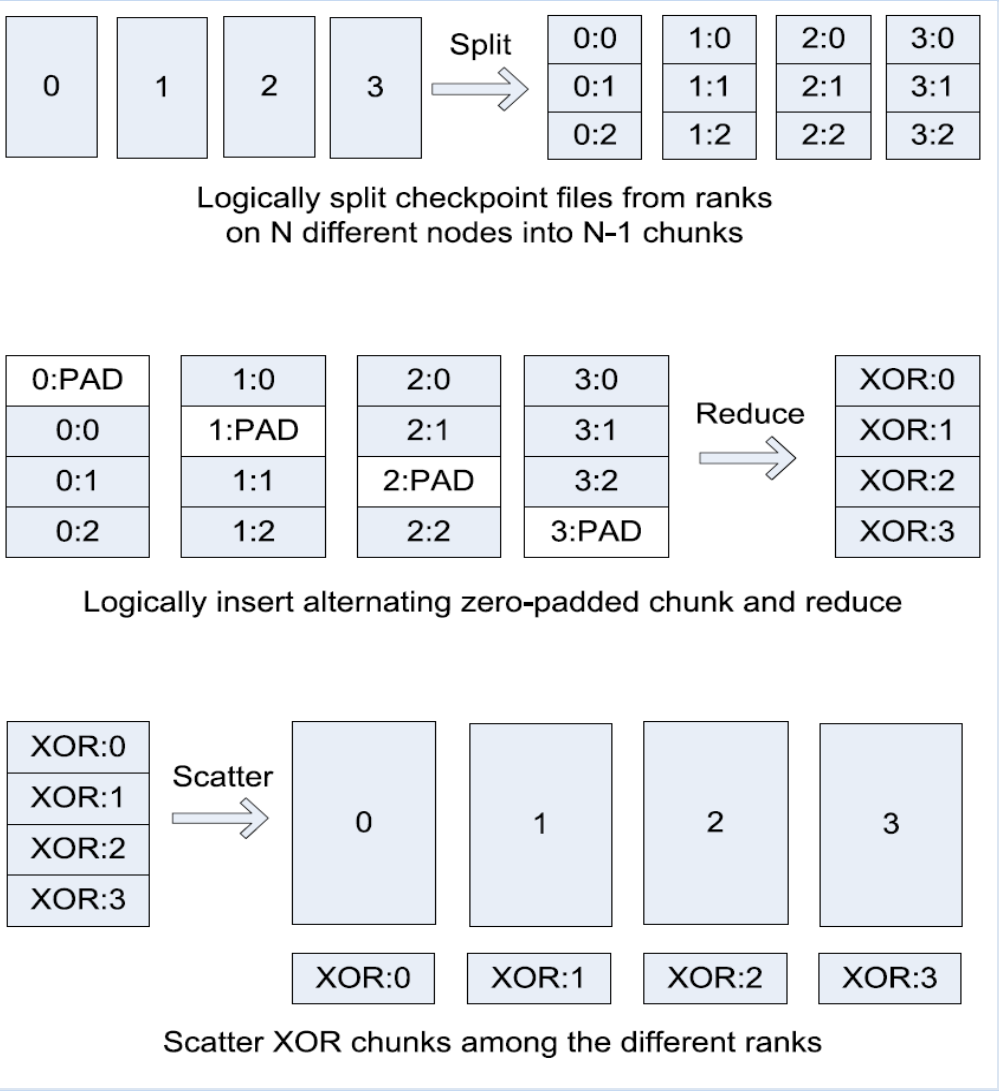
# SCR scheme


Compute nodes
Control directory
Cache directory 1
Cache directory 2
Node-local storage
Prefix directory
Parallel File System

# Buddy-Checkpointing



# XOR Redundancy



# Interface

**`SCR_Initialize()`**

- collective
- called after MPI_Init()
- init global vars, and CP database
- init MPI environment (global-, local-, level-communicators)
- sync with other processes
- search and/or create meta data
- search for checkpoint to restart from
- create cache and PFS directories
- save start time

# Interface

```
SCR_Need_Checkpoint(int *flag)
```

- condition for checkpoint met?
- depends on policy
  - global policy
  - user policy

# Interface

**`SCR_Start_Checkpoint()`**

- collective
- called immediately before writing a checkpoint
- saves start time of creation
- initializes a new CP in database (+ create location in cache)
- flushes CPs to other stages
  - synchronous transfer to other storage levels
  - background (asynchronous) transfer to other storage levels
- deletes older CP(s)
- checks for sufficient space

# Interface

**`SCR_Route_File(char *name, char *file)`**

- builds the *file* path (out) to a checkpoint file named *name* (in)
- called after `SCR_Start_Checkpoint()`
  - path to new checkpoint
- called after `SCR_Initialize()` and before `SCR_Start_Checkpoint()`
  - path to last valid checkpoint accessible by all processes

# Interface

**`SCR_Complete_Checkpoint(int valid)`**

- collective
- called immediately after writing a checkpoint
- save information that all checkpoints were written, whether successful or not
- if valid, sets last checkpoint to current checkpoint
- write meta data files
- save end time of creation
  - calculates time for writing the checkpoint

# Interface

**`SCR_Finalize()`**

- called before `MPI_Finalize()`
- flush checkpoints
- de-allocate resources
- save information that run was successful

# Configuration

1) Environment variables
2) User configuration files
3) System configuration file
4) Compile-time constants

# Environment vars

- `SCR_CACHE_BASE` (cache directory)
- `SCR_CACHE_SIZE` (checkpoints to keep)
- `SCR_FLUSH` (flush every N checkpoints)
- `SCR_FLUSH_ASYNC` (use asynchronous flushing)
- `SCR_PREFIX` (background storage directory)
- `SCR_CHECKPOINT_INTERVAL` (checkpoint every nth call)
- `SCR_CHECKPOINT_SECONDS` (number of seconds between checkpoints)
- `SCR_CHECKPOINT_OVERHEAD` (overhead allowed for checkpointing)
- `SCR_COPY_TYPE (PARTNER, XOR, SINGLE, FILE)`

# Configuration file

- provides more fine granular configuration
- store descriptors to specify multiple directories
  - `STORE=0 BASE=/tmp`
  - `STORE=1 BASE=/tmp/buddy COUNT=10`

- checkpoint descriptors for multiple checkpoint types
  - `CKPT=0 BASE=/tmp INTERVAL=1 TYPE=SINGLE`
  - `CKPT=1 BASE=/tmp/buddy INTERVAL=4 TYPE=PARTNER`

# Remarks

- When defining CPs using a config file (next slide)
  - `SCR_COPY_TYPE="FILE"`
  - `TYPE` can then be set for each checkpoint descriptor
  - one store descriptor must be defined for `/tmp` (control directory)
- Checkpoint conditions are checked the following order:
  1) `SCR_CHECKPOINT_INTERVAL`
  2) `SCR_CHECKPOINT_SECONDS`
  3) `SCR_CHECKPOINT_OVERHEAD`
- `SCR_FLUSH_ASYNC` requires demon process
  - `mpiexec -x -np 1 $HOME/scr/bin/scr_transfer /tmp/$USER/scr.$SCR_JOB_ID/transfer.scrinfo &`

# Checkpoint Example

```
SCR_Need_checkpoint(&perform_checkpoint);
if (perform_checkpoint == 1) {
    dbg0("Starting checkpoint\n");
    SCR_Start_checkpoint();
    // Get backup file path
    sprintf(name, "solve_nbody-%dx%d_r%d.ckpt", nb,
    BLOCK_SIZE,rank);
    if(SCR_Route_file(name, path)==SCR_SUCCESS){
        FILE* fd = fopen(path, "wb");

        // Open, write and close file
        assert(fd != NULL);
        saved_data = fwrite(&current_tstep,sizeof(int),1,fd);
        saved_data += fwrite(particles, sizeof(particles_block_t),
                                    nb,fd);
        assert(fclose(fd)==0);
    }
    int is_valid = saved_data == (nb + 1);
    SCR_Complete_checkpoint(is_valid);

}
```

# Restore Example

```
if (SCR_Route_file(name, path) == SCR_SUCCESS) {
    dbg0("File found! Restoring data.\n");

    FILE* fd = fopen(path, "rb");

    // Open, read and close file
    assert(fd != NULL);
    num_read  = fread(&temp_tstep,sizeof(int),1,fd);
    num_read += fread(local, sizeof(particles_block_t), nb,fd);
    assert(fclose(fd)==0);

    if(num_read == (nb+1)){
        found_cp = 1;
    }else{
        status = -1;
    }
}
```

# Batch-file DEEP

```
export SCR_FLUSH=1
export SCR_COPY_TYPE="SINGLE"
export SCR_CLUSTER_NAME=deepm
export SCR_PREFIX=/work/$USER/checkpoints
export SCR_USER_NAME=$USER
export SCR_JOB_NAME=$PBS_JOBNAME
export SCR_JOB_ID=`echo  $PBS_JOBID | awk
-v FS="." '{print $1}'`
mpiexec -x -np 2 ./binary args
```

# Compilation + Usage

- `./configure CC=mpicc --prefix=$PREFIX --without-yogrt --without-mysql`
- `make`
- `make install`
- link with your application using
  - `CFLAGS= -I$PREFIX/include`
  - `LDFLAGS= -L$PREFIX/lib -lscr`

# DEEP-ER Extensions

- Support for node-local parallel I/O (SIONlib, MPI-IO, PHDF5)
- Support for BeeGFS Cache API
- Support for SIONlib Buddy-Checkpointing

# Thank you!
# Questions?