EuroMPI 2019
ETH Zurich

www.legato-project.eu

Multilevel Checkpointing for MPI Applications

Konstantinos Parasyris
konstantinos.parasyris@bsc.es

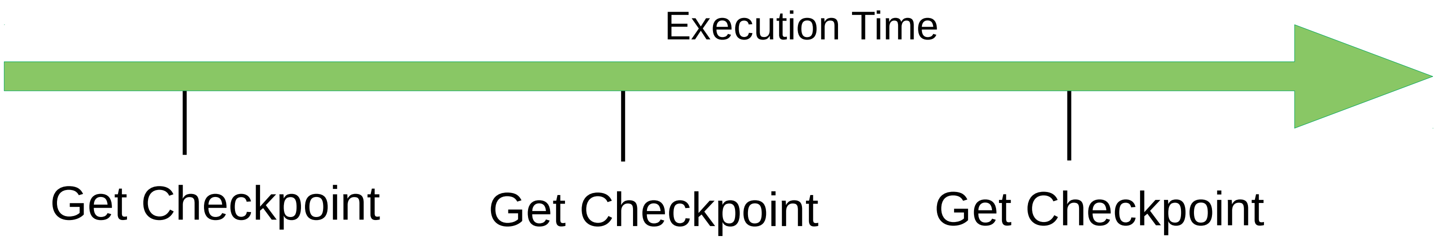
Material

```
git clone -b tutorial https://github.com/leobago/fti TUTORIAL
```

Motivation

- Fault tolerance is critical at extreme scale.
- More components, more failures.
- Multiple different types of failures (hard, soft, ect).
- Power limits might impact reliability.
- Current techniques will not scale.

C/R Idea



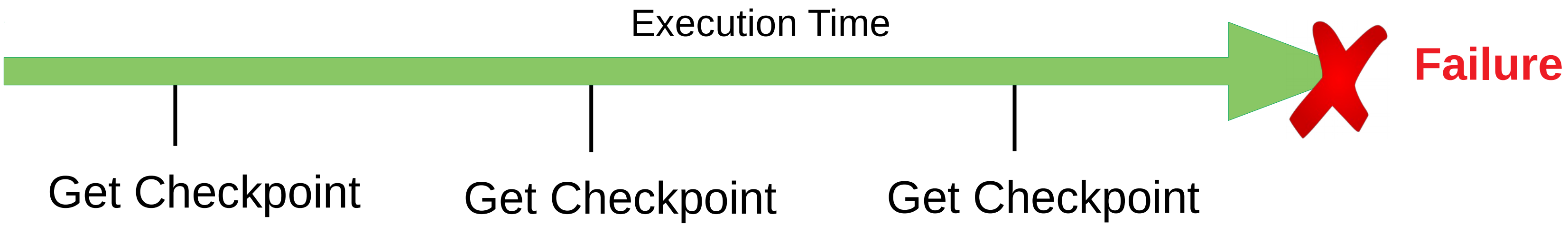
Execution Time

Get Checkpoint

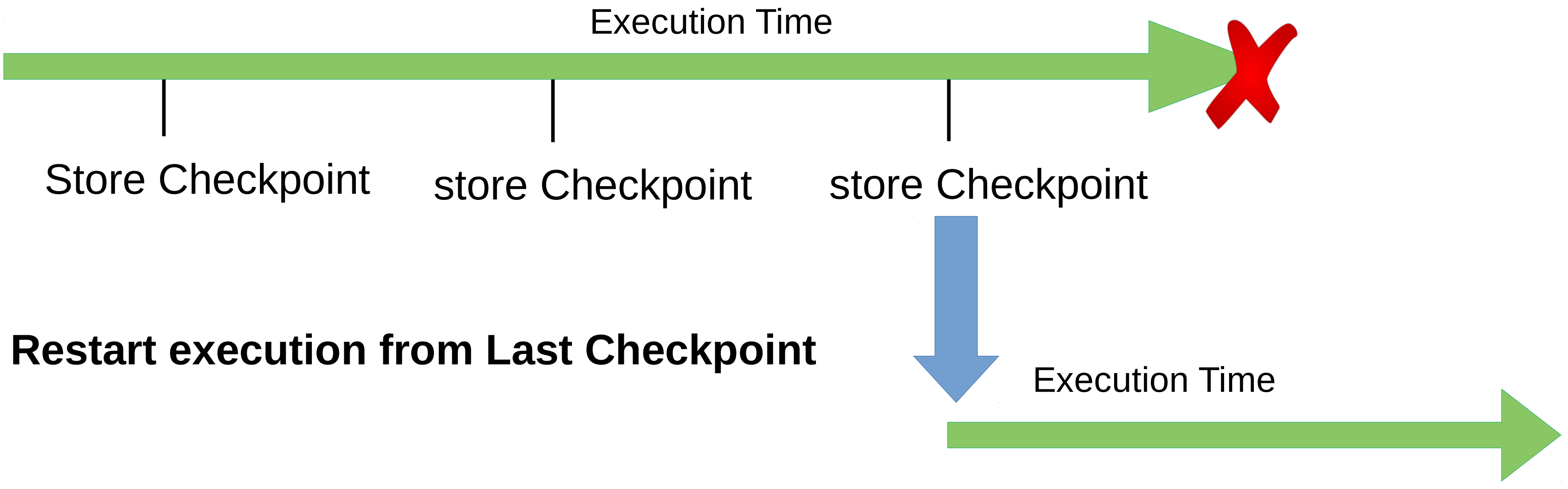
Get Checkpoint

Get Checkpoint

C/R Idea

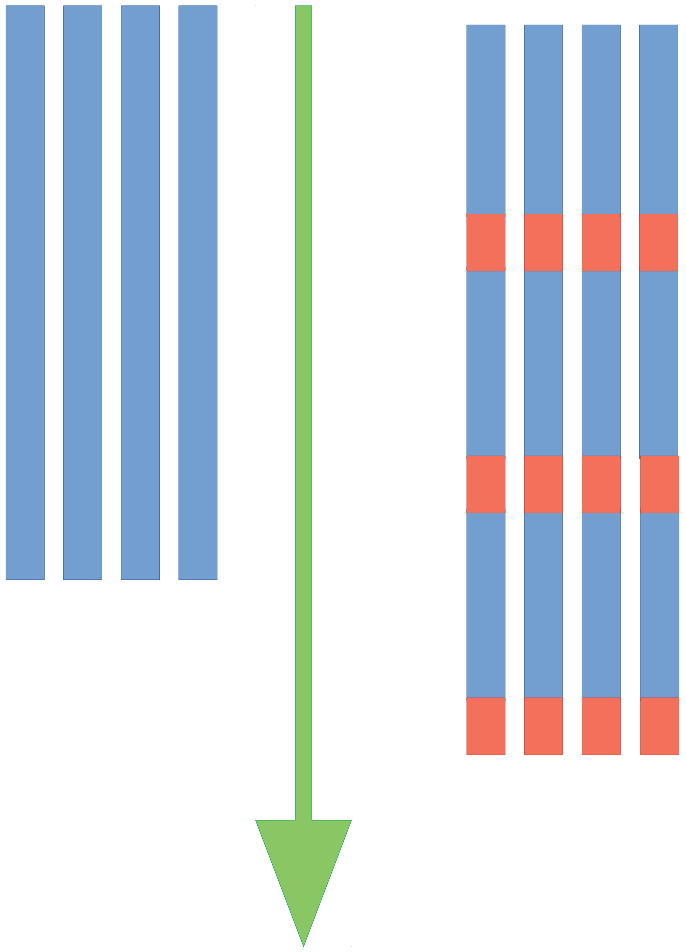

Execution Time
Failure
Get Checkpoint
Get Checkpoint
Get Checkpoint

C/R Idea



Restart execution from Last Checkpoint

Checkpoint efficiency aspects



Application Execution (No Checkpoint)

Time

Application Execution (With Checkpoint)

🟥 Checkpoint overhead depends on:

- Size of data to be stored.
- Frequency of checkpoints.
  (Non Frequent checkpoints increase execution time on recovery).
- File system:
  Storage close to the node (SSD) are faster but less reliable.

Checkpoint Scope

1) Application Level

- The developer indicates:
  - The data to be stored
  - When will the checkpoint take place

2) User Level

- Checkpoint entire user-application.

3) Kernel Level

- Checkpoint Entire System

Basic information about FTI

- Download at: http://www.github.com/leobago/fti
- Documentation at https://github.com/leobago/fti/wiki/Introduction (Library in C/C++ with Fortran bindings)
- More than 10000 lines of code
- Applications ported:
  - HACC
  - Nek5K
  - CESM (ice module)
  - LAMMPS
  - GYSELA 5D
  - SPECFEM3D
  - HYDRO
  - Other miniApps

Multilevel Checkpoint.

Level-1 (Local Storage) :
SSD, PCM, NVM. Fastest checkpoint level. Low reliability, transient failures

Level-2 (Partner Copy) :
Ckpt. Replication. Fast copy to neighbor node. tolerates multiple node failures (depending on location).

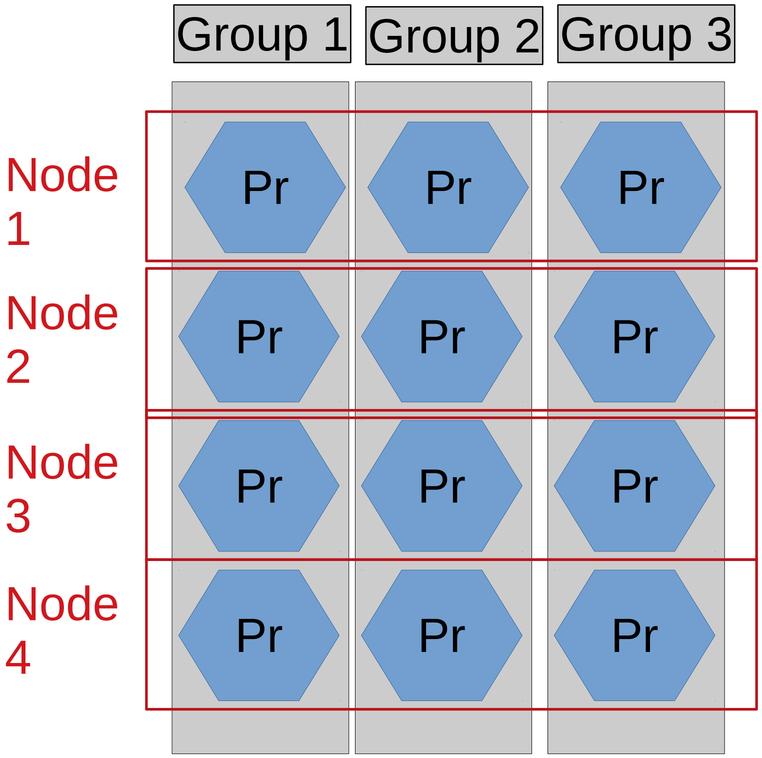
Level-3 (RS Encoding) :
Ckpt. Encoding. Slow for large checkpoints. Tolerates Multiple node failures (independent of location)

Level-4 (File System (PFS)) :
Classic Ckpt. Slowest of all levels. The most reliable. Power outage.

Multiple levels offer multiple:

- Resiliency levels
- Checkpoint overheads
- Checkpoint intervals
- Power consumption

Topology aware clustering



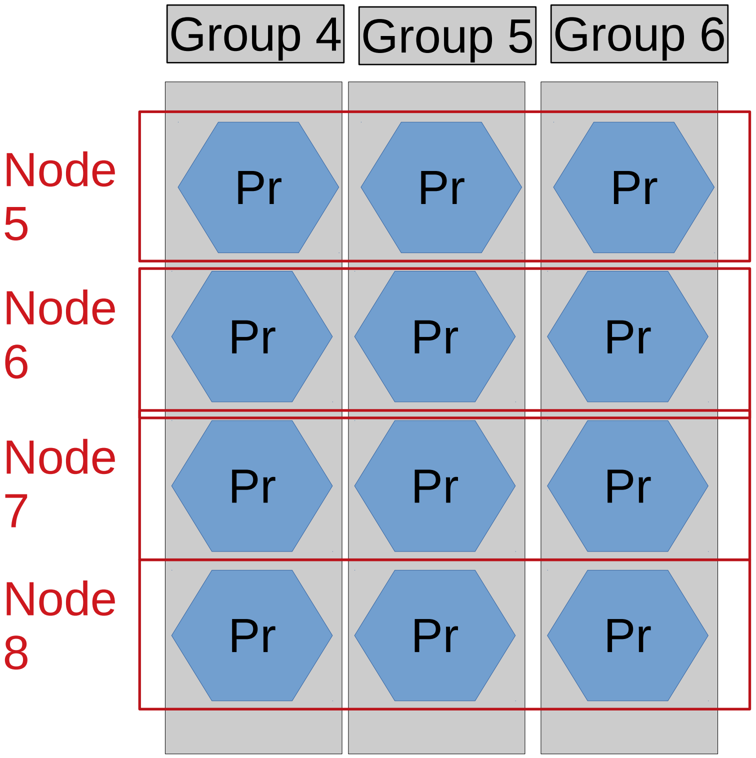
Group 1 Group 2 Group 3

Node 1: Pr Pr Pr

Node 2: Pr Pr Pr

Node 3: Pr Pr Pr

Node 4: Pr Pr Pr



Group 4 Group 5 Group 6

Node 5: Pr Pr Pr

Node 6: Pr Pr Pr

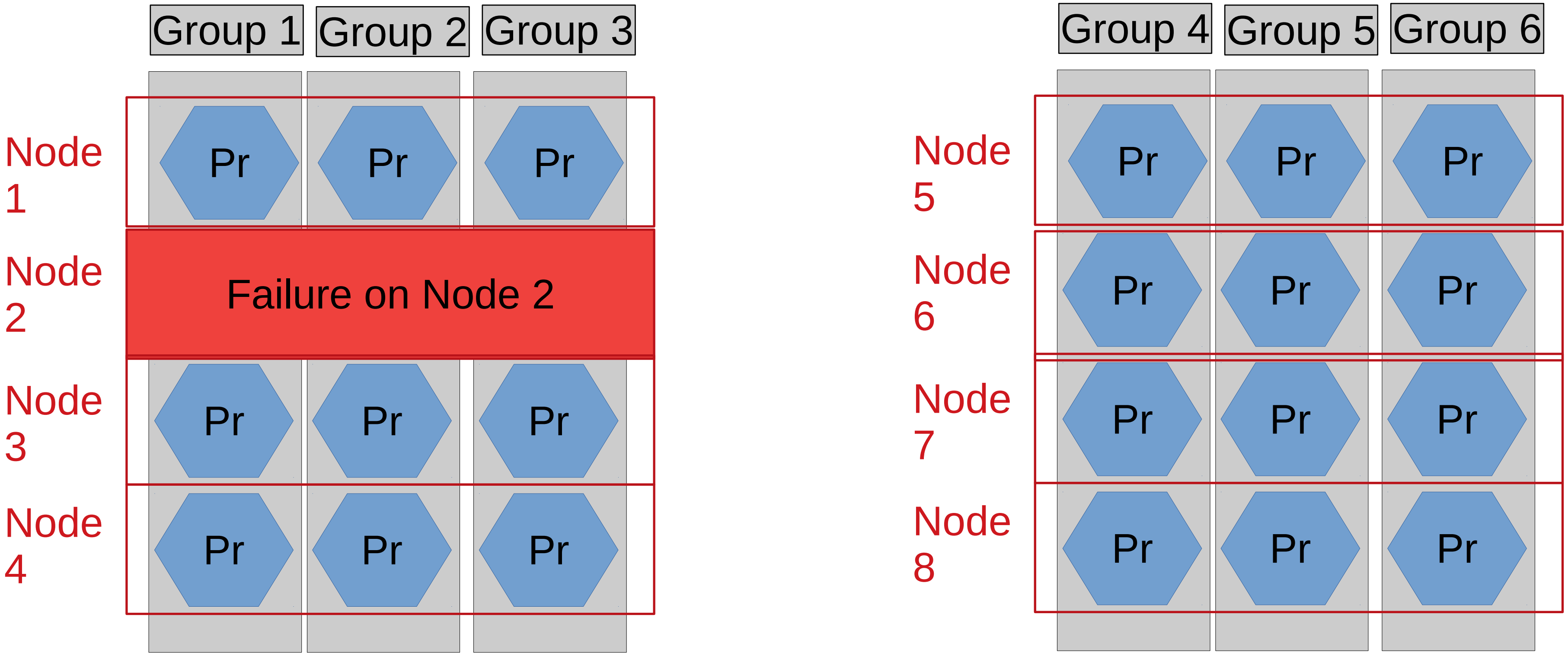
Node 7: Pr Pr Pr

Node 8: Pr Pr Pr

- Automatic process location recognition
- Intelligent clustering

- Enhanced reliability for node crashes
- Automatic repositioning after failure

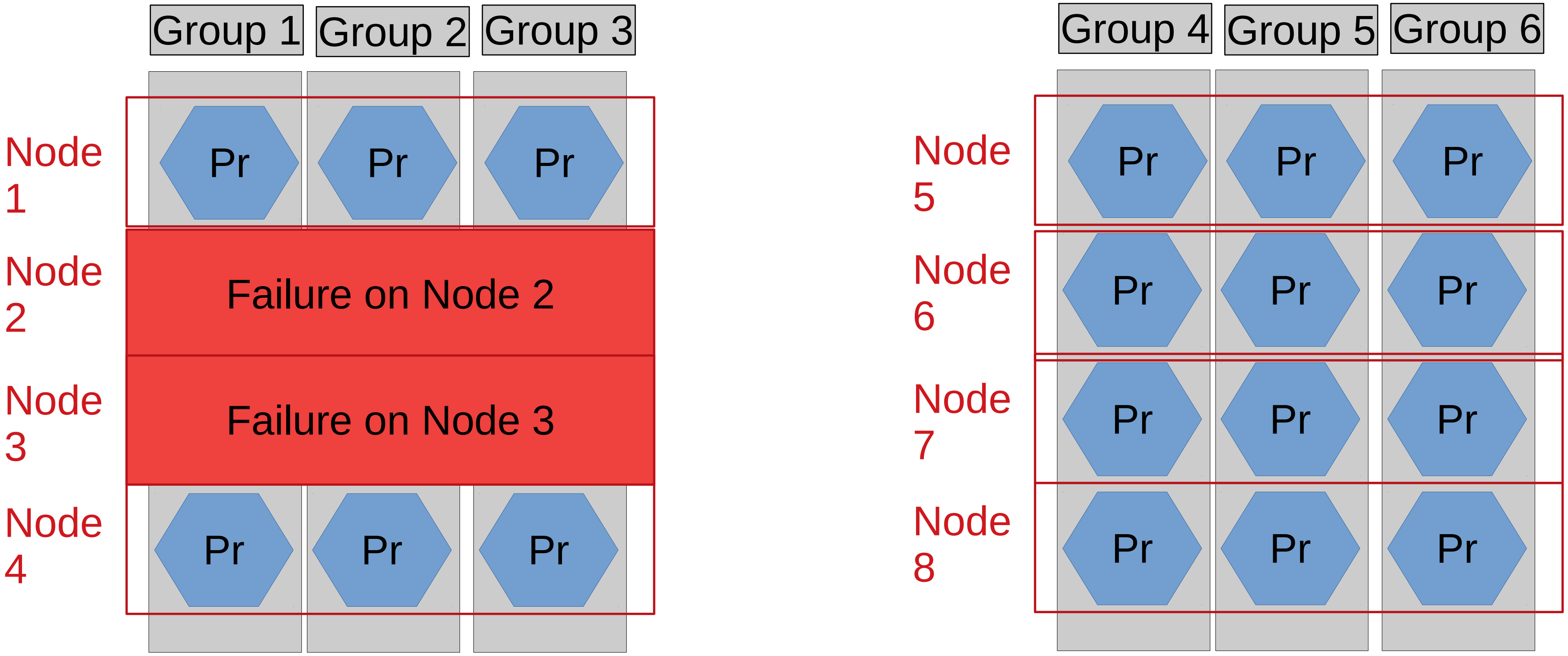
Level 2 Recovery



Level 2 Recovery: Node can recover from the data stored into partner node.

If me and partner fail we cannot recover.

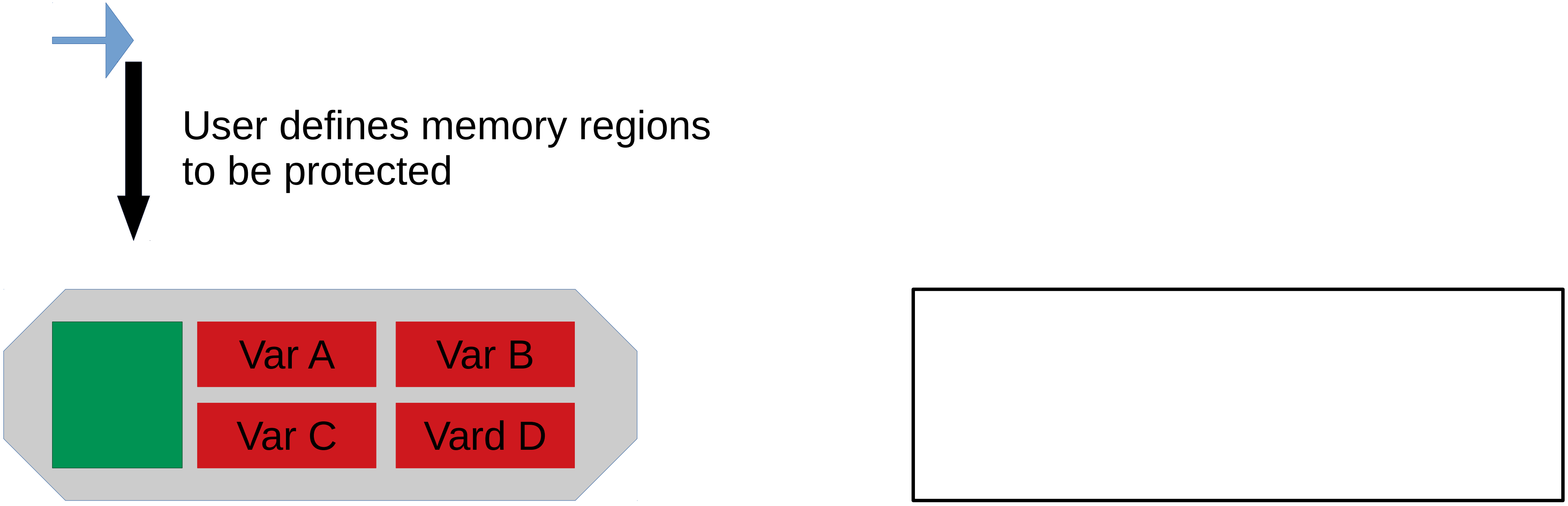
Level 3 Recovery



Level 3 Recovery: Node can recover using the Rs encoded files.

If more than the half nodes of the group fail I cannot recover

Checkpoint Methods (Normal)



User defines memory regions to be protected

Memory occupied by the memory.

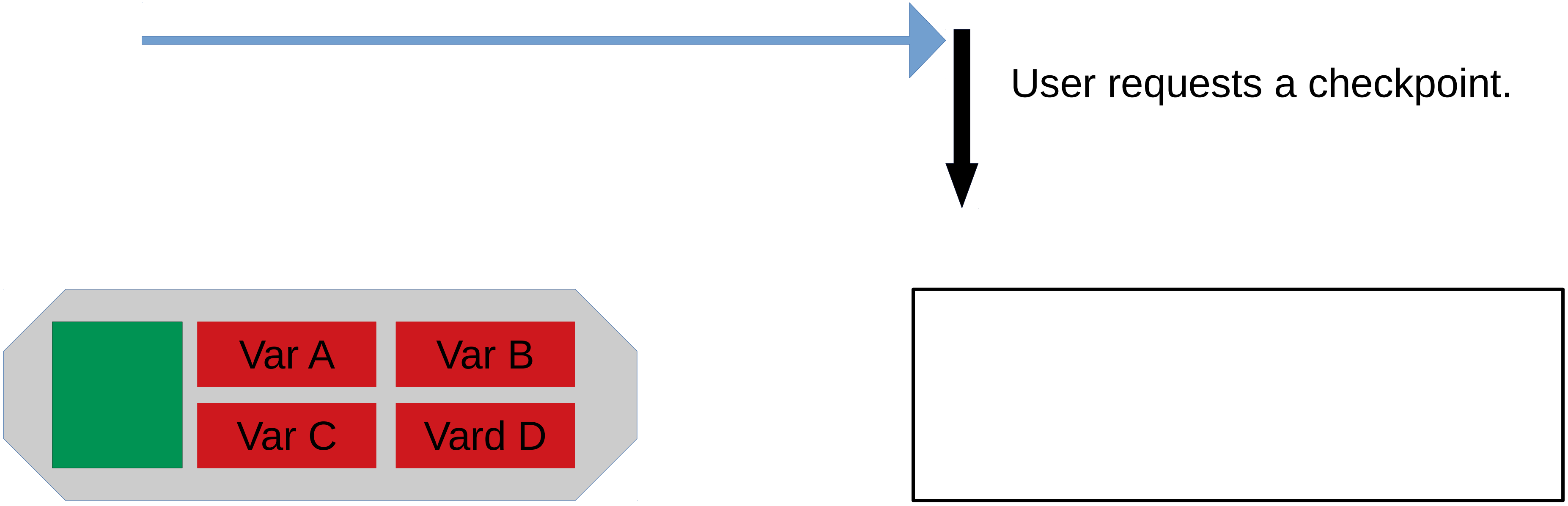
Checkpoint file(s).

🟩 Non-Protected user variable

🟥 Protected user variable

⬜ Memory region not allocated by user.

Checkpoint Methods (Normal)



User requests a checkpoint.

Var A

Var B

Var C

Vard D

Memory occupied by the memory.

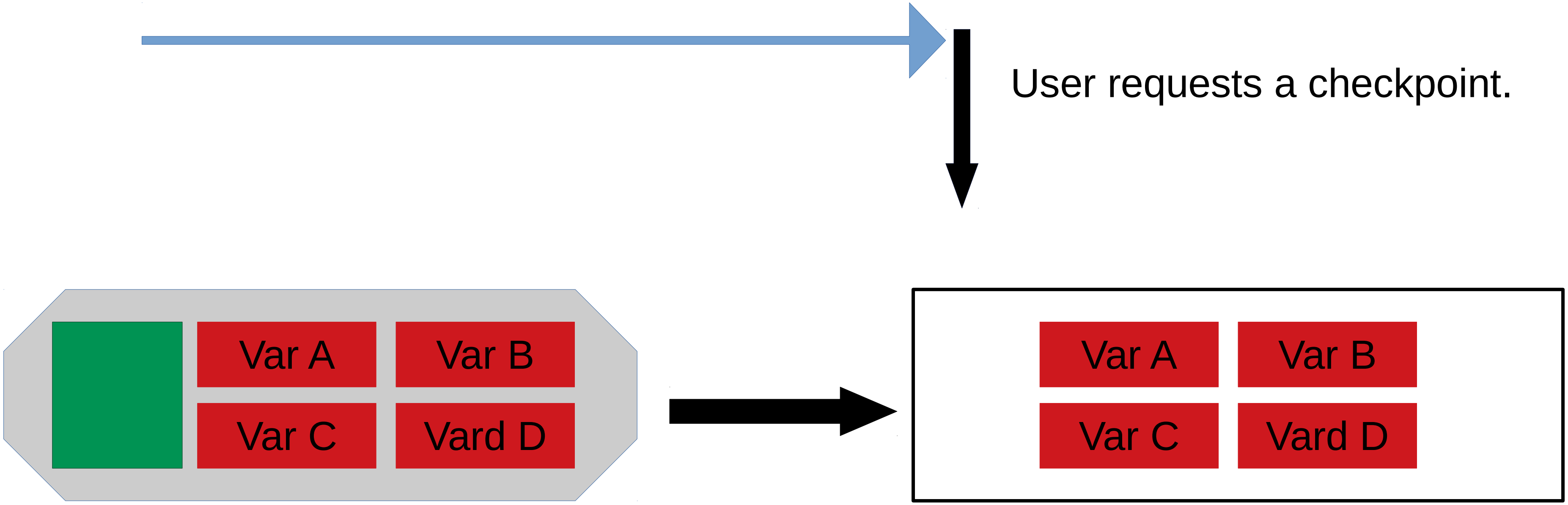
Checkpoint file(s).

🟩 Non-Protected user variable

🟥 Protected user variable

⬜ Memory region not allocated by user.

Checkpoint Methods (Normal)



User requests a checkpoint.

Var A

Var B

Var C

Vard D

Var A

Var B

Var C

Vard D

Memory occupied by the memory.

Checkpoint file(s).

🟩 Non-Protected user variable

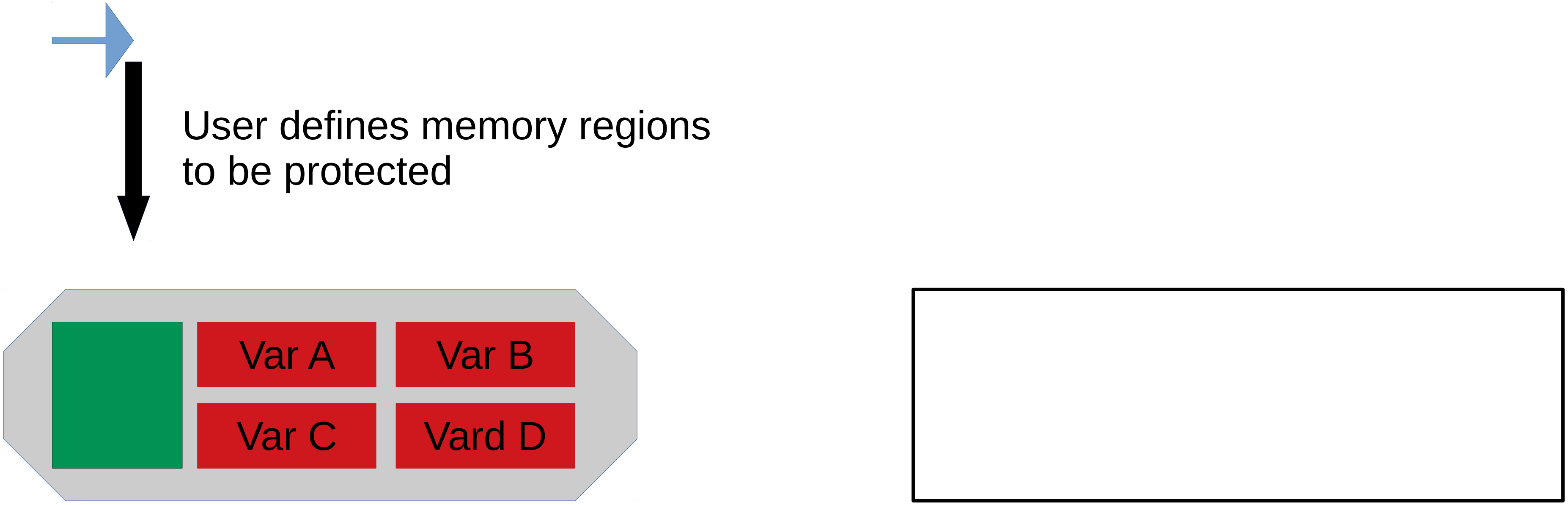
🟥 Protected user variable

⬜ Memory region not allocated by user.

LEGaTO

www.legato-project.eu

Checkpoint Methods (Incremental Checkpoint)



User defines memory regions to be protected

Memory occupied by the memory.

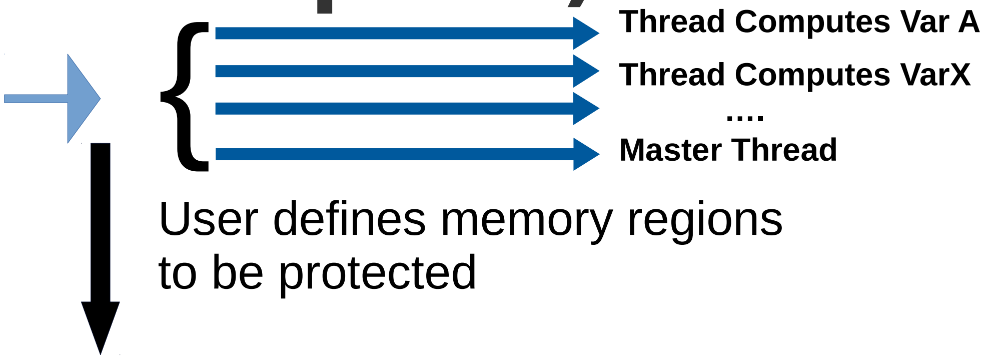
Checkpoint file(s).

🟩 Non-Protected user variable

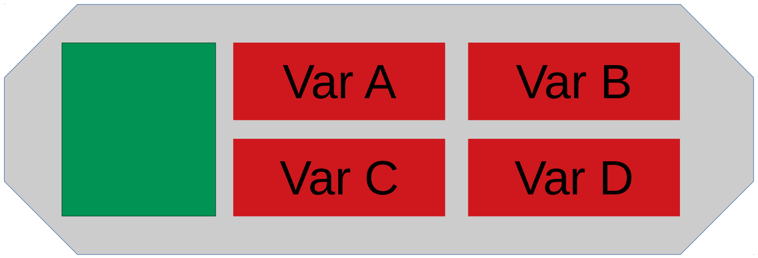
🟥 Protected user variable

⬜ Memory region not allocated by user.

Checkpoint Methods (Incremental Checkpoint)



User defines memory regions to be protected



Memory occupied by the memory.

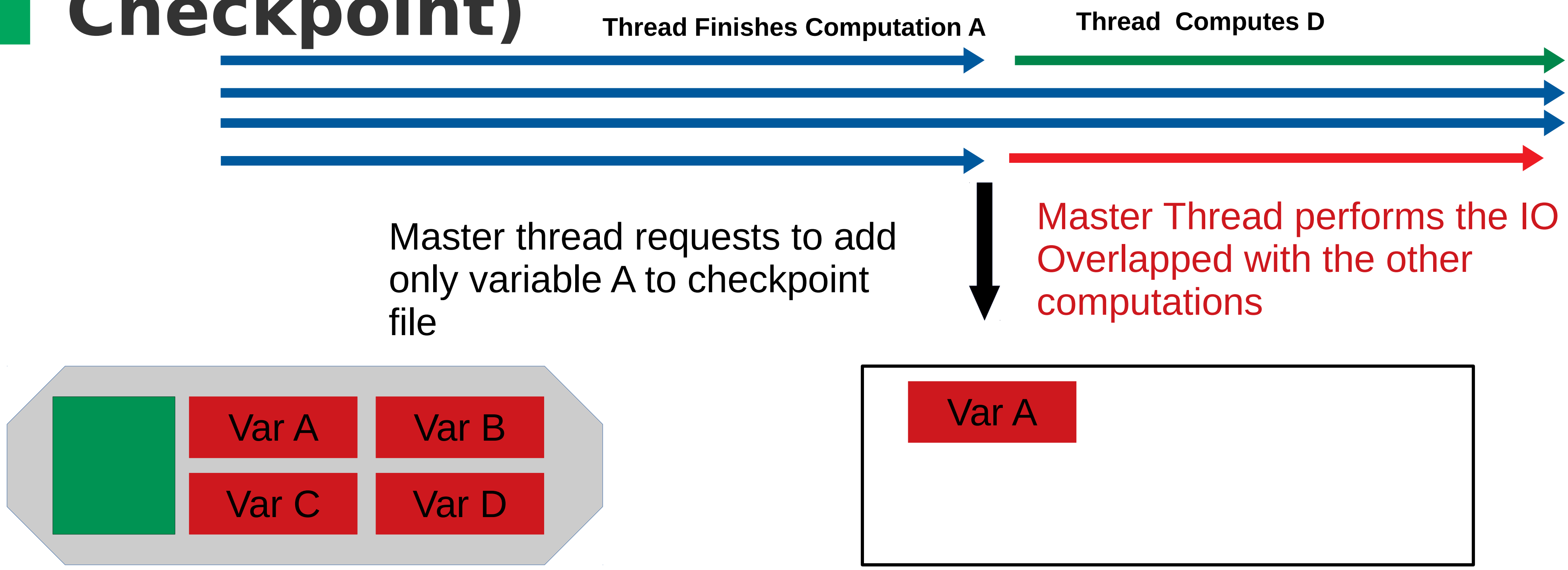
Checkpoint file(s).

🟩 Non-Protected user variable

🟥 Protected user variable

⬜ Memory region not allocated by user.

Checkpoint Methods (Incremental Checkpoint)



Master thread requests to add only variable A to checkpoint file

Master Thread performs the IO Overlapped with the other computations

Memory occupied by the memory.

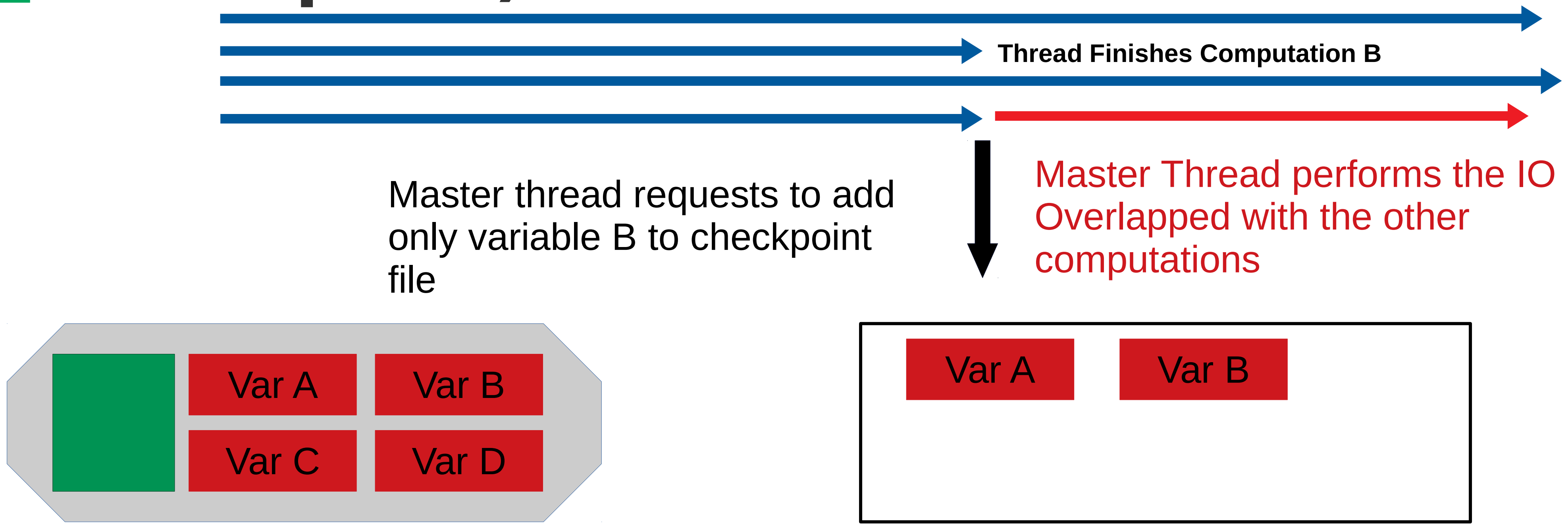
Checkpoint file(s).

🟩 Non-Protected user variable

🟥 Protected user variable

⬜ Memory region not allocated by user.

Checkpoint Methods (Incremental Checkpoint)



Master thread requests to add only variable B to checkpoint file

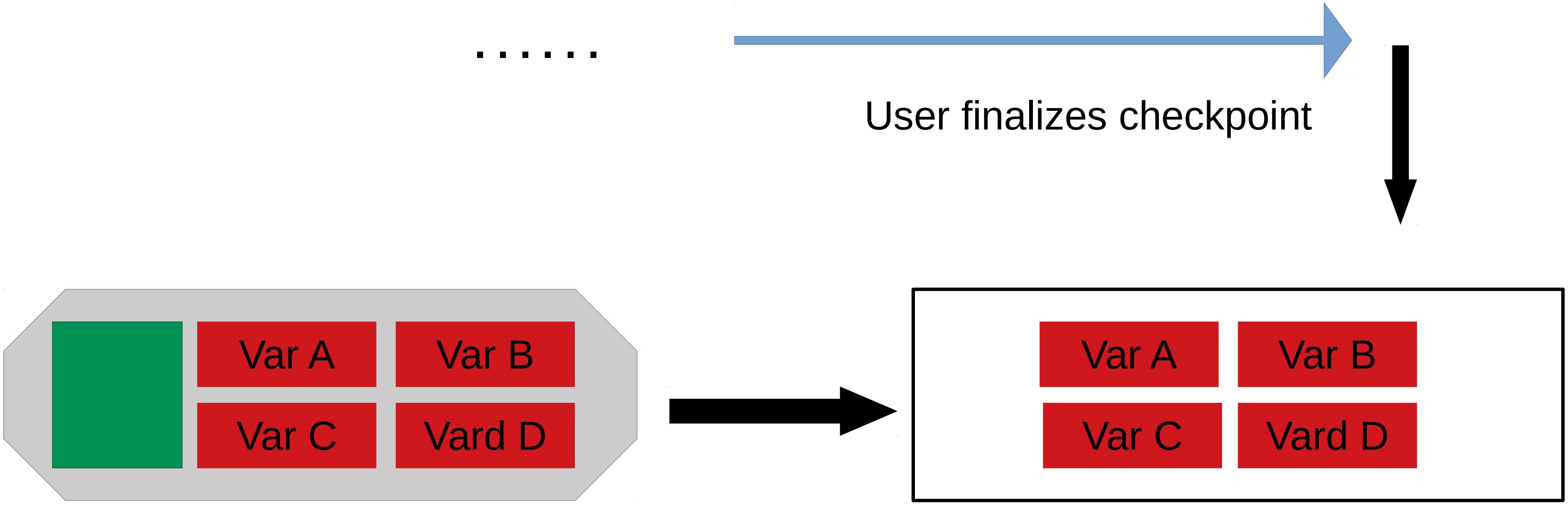
Master Thread performs the IO Overlapped with the other computations

Memory occupied by the memory.

Checkpoint file(s).

- 🟩 Non-Protected user variable
- 🟥 Protected user variable
- ⬜ Memory region not allocated by user.

Checkpoint Methods (Incremental Checkpoint)



....

User finalizes checkpoint

Var A

Var B

Var C

Vard D

Var A

Var B

Var C

Vard D

Memory occupied by the memory.

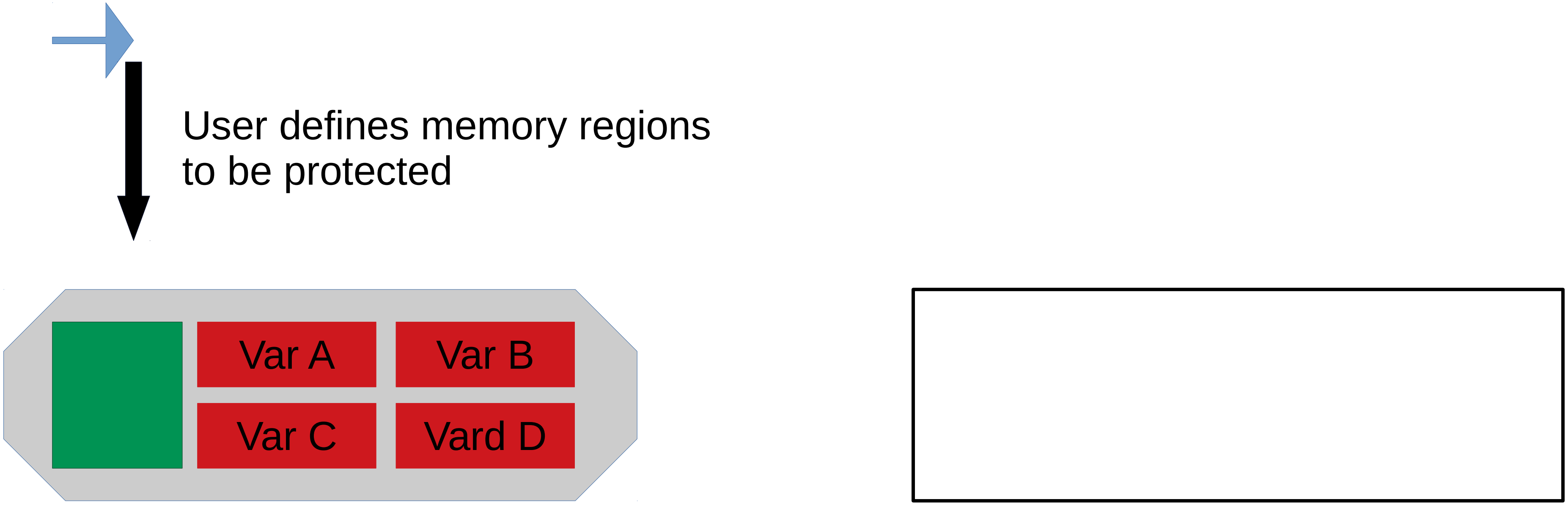
Checkpoint file(s).

🟩 Non-Protected user variable

🟥 Protected user variable

⬜ Memory region not allocated by user.

Checkpoint Methods (differential Checkpoint (dCP))



User defines memory regions to be protected

Memory occupied by the memory.

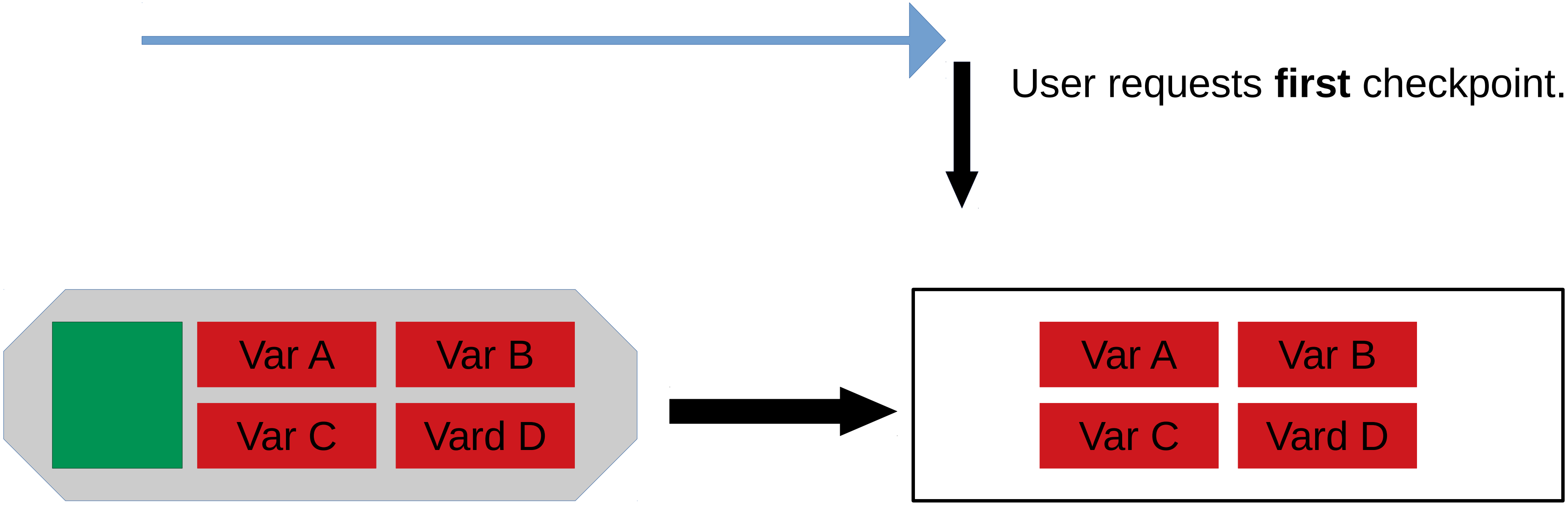
Checkpoint file(s).

🟩 Non-Protected user variable

🟥 Protected user variable

▨ Memory region not allocated by user.

Checkpoint Methods (dCP)



User requests **first** checkpoint.

Var A Var B

Var C Vard D

Var A Var B

Var C Vard D

Memory occupied by the memory.

Checkpoint file(s).

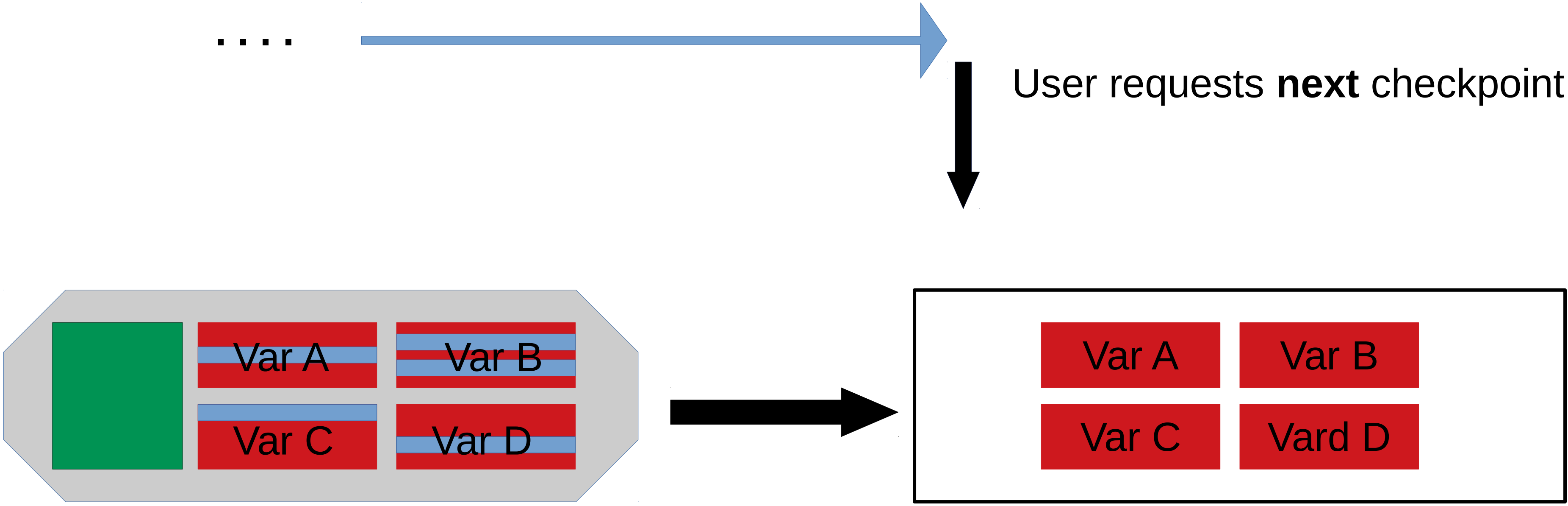
🟩 Non-Protected user variable

🟥 Protected user variable

⬜ Memory region not allocated by user.

In the dCP case on the first checkpoint all data are stored in the checkpoint

Checkpoint Methods (dCP)



User requests **next** checkpoint.

Memory occupied by the memory.

Checkpoint file(s).

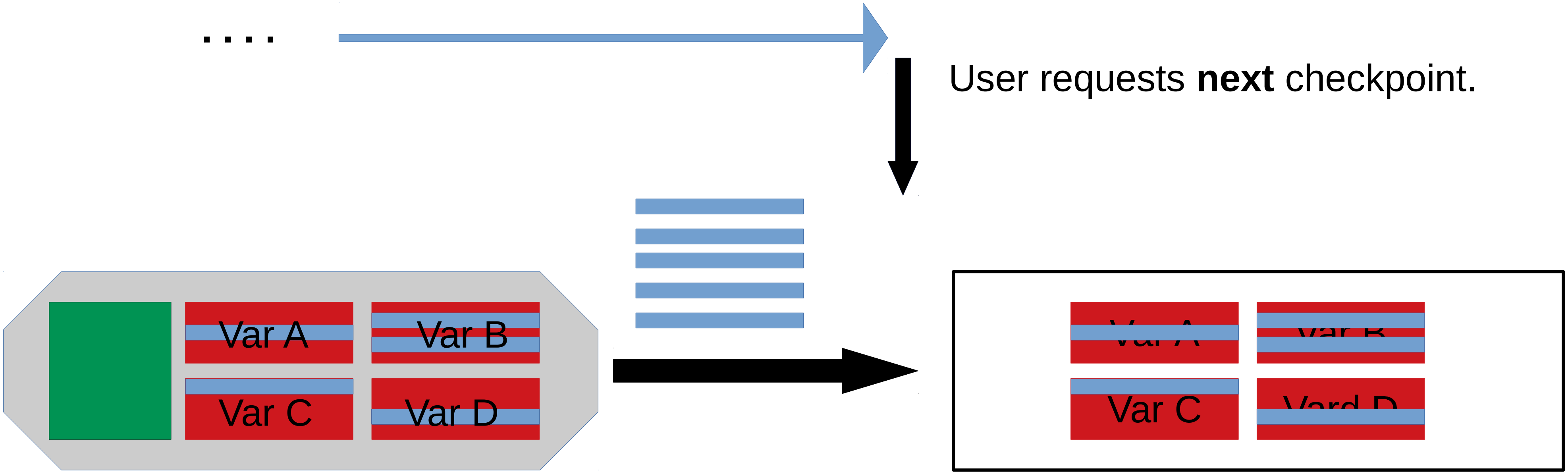
🟩 Non-Protected user variable

🟥 Protected user variable

⬜ Memory region not allocated by user.

🟦 **Data that have changed in comparison to previous checkpoint**

Checkpoint Methods (dCP)



User requests **next** checkpoint.

Memory occupied by the memory.

Checkpoint file(s).

🟩 Non-Protected user variable

🟥 Protected user variable

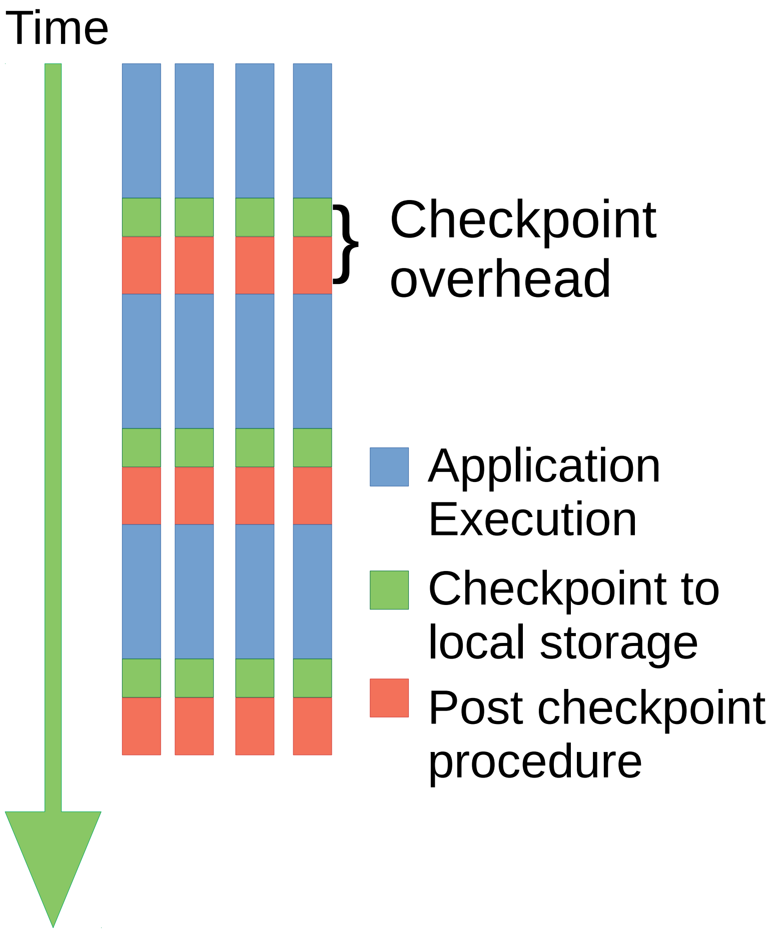
⬜ Memory region not allocated by user.

🟦 **Data that have changed in comparison with previous checkpoint**

Store only the changed data to the checkpoint file. ----→ Reduces IO.

Asynchronous post-processing

Synchronous post-processing

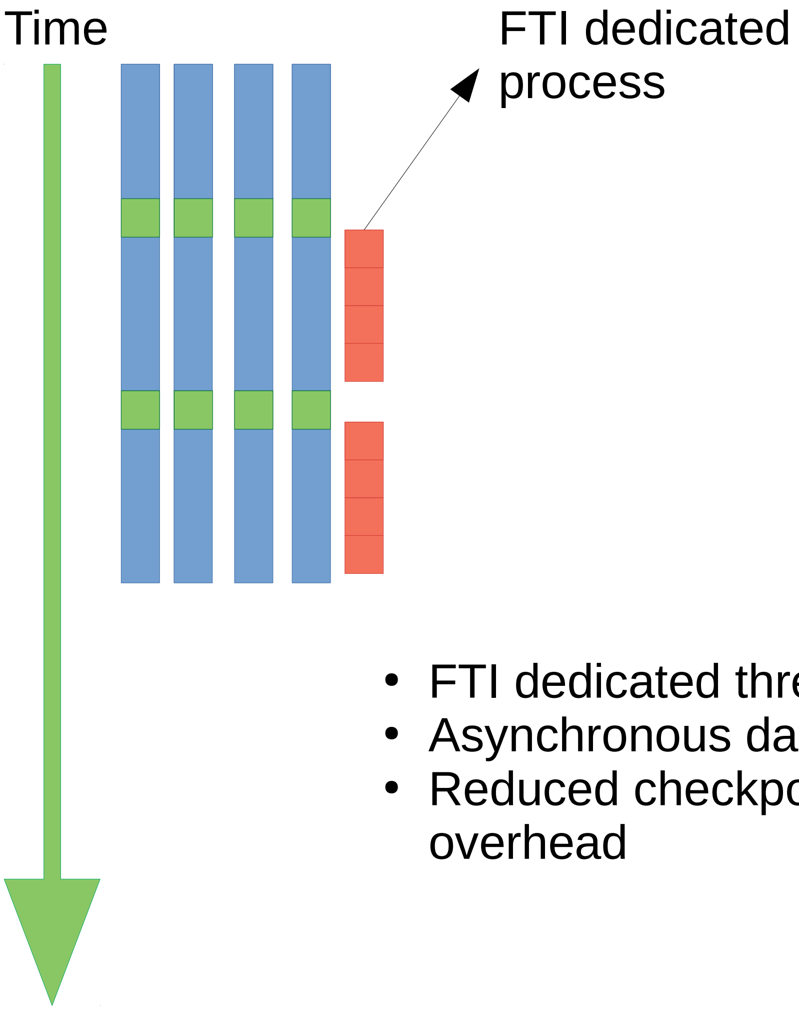


Time

Checkpoint overhead

- 🟦 Application Execution
- 🟩 Checkpoint to local storage
- 🟥 Post checkpoint procedure

Asynchronous post-processing



Time

FTI dedicated process

- FTI dedicated threads
- Asynchronous data transfer
- Reduced checkpoint overhead

How to use.

Functions:

- ***FTI_Init()***
- ***FTI_Protect()***
- ***FTI_Snapshot()***
- ***FTI_Finalize()***

Communicator:

- **FTI_COMM_WORLD**

```
int main(int argc, char **argv) {
        MPI_Init(&argc, &argv);
        FTI_Init(“conf.fti”, MPI_COMM_WORLD);
        double *grid;
        int i, steps=500, size=10000;
        initialize(grid);
        FTI_Protect(0, &i,1, FTI_INTG);
        FTI_Protect(1, grid, size,FTI_DFLT);
        for (i=0; i<steps; i++) {
                FTI_Snapshot();
                kernel1(grid);
                kernel2(grid);
                comms(FTI_COMM_WORLD);
        }
        FTI_Finalize();
        MPI_Finalize();
        return 0;
}
```

FTI_Init(...)

FTI_Init(confFile, communicator):

- Read/parse configuration file
- Recognizes whether is a restart or not
- Creates checkpoint directories
- Detect topology of the system
- Regenerates/moves data upon recovery
- Splits the communicator (optional)

FTI_Protect(...)

FTI_Protect(ID,pointer,size,type):

- Stores metadata of the protected variable
- FTI can predict size of checkpoints
- Useful for data compression/aggregation
- Can be reset during the execution
- User can create new FTI types
- Required in order to write/read ckpt. data

FTI_Snapshot()

FTI_Snapshot():

- Measures (global average) iteration length
- Exponential decay for global agreement
- Translates from minutes to iterations
- Test if it is time for a checkpoint
- It saves the checkpoint as requested
- It loads the checkpoint upon recovery
- Planning to integrate notifications

Beyond FTI_Snapshot

FTI_Checkpoint(ID, lvl):

- Takes a checkpoint with id ID and level lvl

FTI_Status():

- Returns the status (initial run or restart)

FTI_Recover():

- It recovers from last available checkpoint

Incremental Checkpoint

New Functions:

- **FTI_InitICP()**
- **FTI_AddVarICP()**
- **FTI_FinalizeICP()**

Communicator:

- **FTI_COMM_WORLD**

```
int main(int argc, char **argv) {
        MPI_Init(&argc, &argv);
        FTI_Init(“conf.fti”, MPI_COMM_WORLD);
        double *grid1 , *grid2;
        int i, steps=500, size=10000;
        initialize(grid);
        FTI_Protect(0, &i,1, FTI_INTG);
        FTI_Protect(1, grid1, size,FTI_DFLT);
        FTI_Protect(2, grid2, size,FTI_DFLT);
        for (i=0; i<steps; i++) {
                FTI_InitICP( i, (i +1)%4 +1 , 1 );
                FTI_AddVarICP(0);
                kernel1(grid);
                FTI_AddVarICP(1);
                kernel2(grid);
                FTI_AddVarICP(2);
                comms(FTI_COMM_WORLD);
                FTI_FinalizeICP();
        }
        FTI_Finalize();
        ….
}
```

Configuration file (1/3)

```
[basic]
# Set to 1 for having 1 FTI dedicated process per node
Head                                        =      1
# Number of processes per node (including FTI dedicated processes)
node_size                                   =     2
# Path where local checkpoints will be stored
ckpt_dir                                    = /path/to/local/storage/
# Path where global checkpoints will be stored
glbl_dir                                    = /path/to/global/storage/
# Path where checkpoints metadata will be stored
meta_dir                                    = /path/to/myhome/.fti/
# Checkpoint interval in minutes for level 1
ckpt_int                                    = 1
# Checkpoint interval in minutes for level 2
ckpt_l2                                     = 2
# Checkpoint interval in minutes for level 3
ckpt_l3                                     = 4
# Checkpoint interval in minutes for level 4
ckpt_l4                                     = 8
```

Configuration file (2/3)

```
[basic]

# Set to 0 to do L2 postprocessing asynchronously by the dedicated process
inline_l2                                         = 0

# Set to 0 to do L3 postprocessing asynchronously by the dedicated process
inline_l3                                         = 0

# Set to 0 to do L4 postprocessing asynchronously by the dedicated process
inline_l4                                         = 0

# Set to 1 to keep the last checkpoint after Finalize
keep_last_ckpt                                    = 0

# Size of the group for RSencoding and Partnercopy ring
group_size                                        = 4

# Set to 1 for verbose mode, 2 for moderate, 3 for silent
Verbosity                                         = 1
```

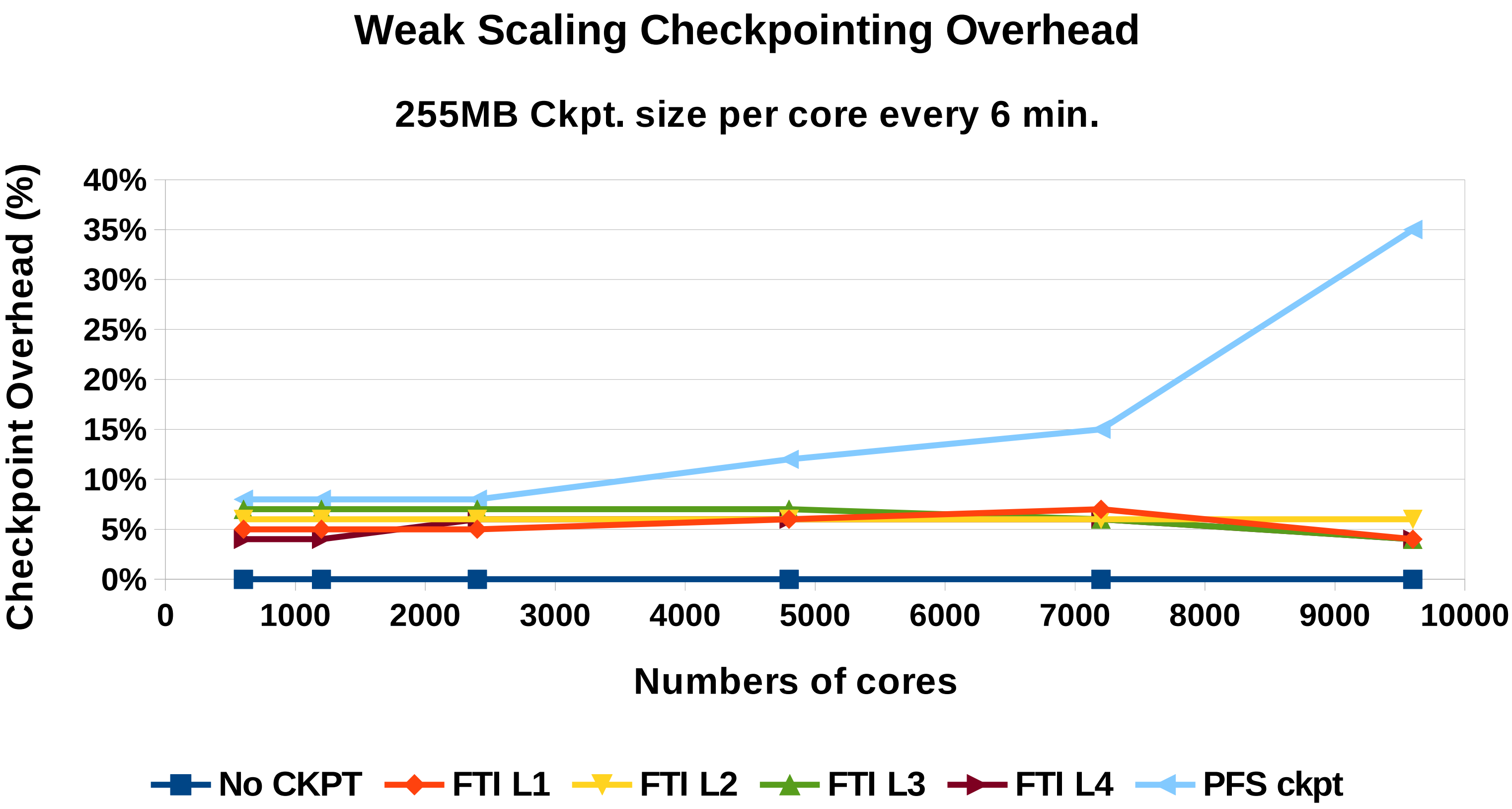
Configuration file (3/3)

```
[restart]
# This will be set to 1 automatically after FTI_Init

Failure                                  = 0

# This will be set to 1 automatically after FTI_Init
exec_id                                  = 20131120_150152

[advanced]
# Block size for communications
block_size                               = 1024

# MPI tag for FTI communications
mpi_tag                                  = 2612

# Set to 1 for local tests in one single node
local_test                               = 0
```

Scaling to ~10K processes

- CURIE supercomputer in France
- SSD on the compute nodes (16 cores)
- HYDRO scientific application
- Using 1 FTI dedicated process per node
- Checkpoint every ~6 minutes
- Weak scaling to almost 10k processes

Scaling to ~10K processes



Scaling to >32K processes

- MIRA supercomputer at ANL (BG\Q)
- Persistent memory compute nodes
- LAMMPS scientific application
- Lennard-Jones simulation of 1.3 billion atoms
- 512 nodes, 64 MPI processes per node (32,678pr.)
- Power monitoring during the entire run
- Checkpoint every ~5 minutes
- Less than 5% overhead on time to completion

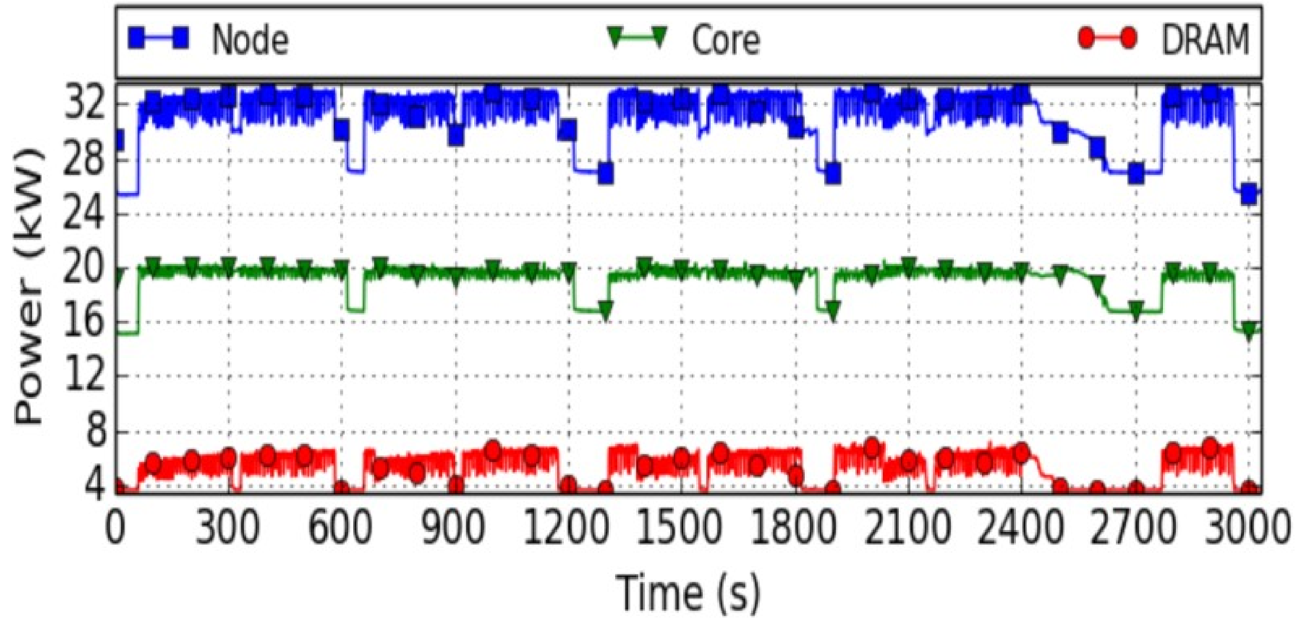
Scaling to >32K processes

Synchronous Checkpoint

Without FTI – dedicated process

Head = 0

Execution: ~ 3000s



Scaling to >32K processes

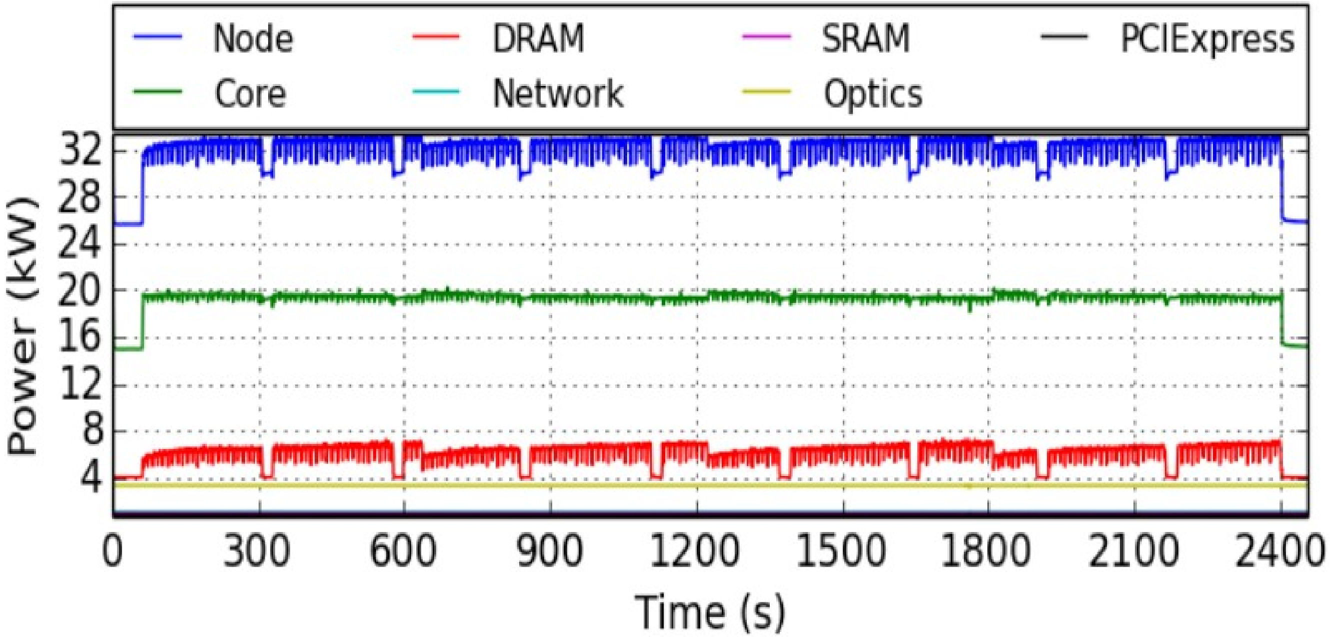
Asynchronous Checkpoint

With FTI – dedicated process

Head = 1

Execution: ~ 2400s

Execution 10 minutes Faster



Features and Limitations

- ✔ FTI can predict time and size of next checkpoints
- ✔ Detailed knowledge of the datasets allows for:
  - ✔ Transparent data compression-verification
  - ✔ Transparent dedicated processes (Comm. Split)
  - ✔ Topology reconstruction upon restart
- ✔ Dynamic checkpoint interval adaptation

- ✗ FTI needs every rank in the given communicator to write a checkpoint file
- ✗ Application level checkpoint (code modification)
- ✗ Coordinated checkpoint, everybody restarts

Thank You!!!!