# ACR:

# Amnesic Checkpoint and Recovery

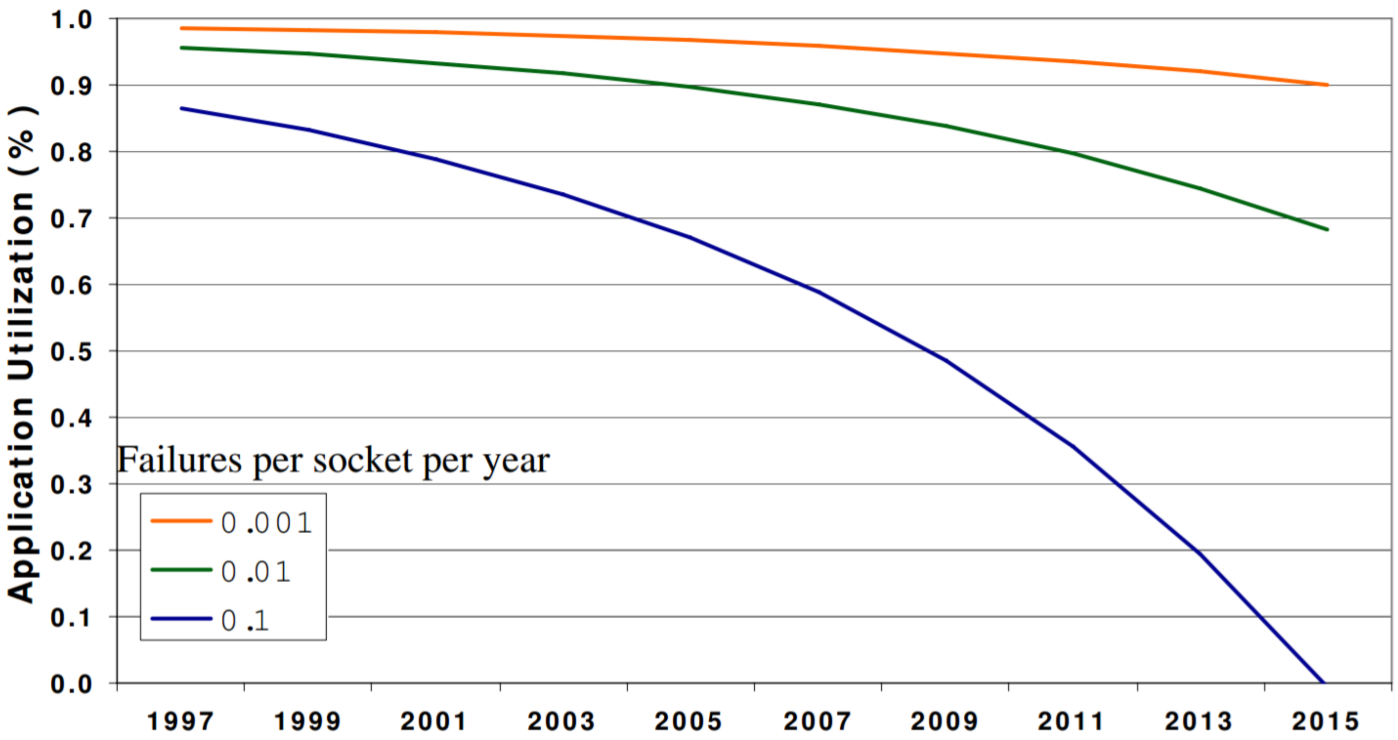
**Ismail Akturk**
University of Missouri, Columbia
akturki@missouri.edu

Ulya R. Karpuzcu
University of Minnesota, Twin Cities
ukarpuzc@umn.edu

University of Missouri

University of Minnesota

# Motivation




P. Kogge, et al., “Exascale computing study: Technology challenges in achieving exascale systems,” 2008.

# Amnesic Checkpointing and Recovery

- **Challenge:**
  - Checkpoint &rollback/recovery overheads quickly dominates as the failure rate increases
  - Need to find ways to mitigate Checkpoint &rollback/recovery overheads
- **Idea:** reduce the volume of data to be checkpointed by relying on cost-effective **recomputation**
  - Eliminate values from checkpoint set if they are recomputable (cost effectively)
  - Recomputation of eliminated values is necessary only on recovery (which is less frequent than checkpointing)

# Amnesic Checkpointing and Recovery

$$o_{chk} = \#_{chk} \times o_{wr,chk}$$

# Amnesic Checkpointing and Recovery

$$o_{chk} = \#_{chk} \times o_{wr,chk}$$

$$o_{chk,ACR} = \#_{chk} \times o_{wr,ACR}$$

# Amnesic Checkpointing and Recovery

$$o_{chk} = \#_{chk} \times o_{wr,chk}$$

$$o_{chk,ACR} = \#_{chk} \times o_{wr,ACR}$$

$$o_{recovery} = \#_{recovery} \times (o_{waste} + o_{rollback})$$

# Amnesic Checkpointing and Recovery

$$o_{chk} = \#_{chk} \times o_{wr,chk}$$

$$o_{chk,ACR} = \#_{chk} \times o_{wr,ACR}$$

$$o_{recovery} = \#_{recovery} \times (o_{waste} + o_{rollback})$$

$$o_{recovery,ACR} = \#_{recovery} \times (o_{waste,ACR} + o_{rollback,ACR} + o_{recomp,ACR})$$

# Amnesic Checkpointing and Recovery

$$o_{chk} = \#_{chk} \times o_{wr,chk}$$

$$o_{chk,ACR} = \#_{chk} \times o_{wr,ACR}$$

$$o_{recovery} = \#_{recovery} \times (o_{waste} + o_{rollback})$$

$$o_{recovery,ACR} = \#_{recovery} \times (o_{waste,ACR} + o_{rollback,ACR} + o_{recomp,ACR})$$

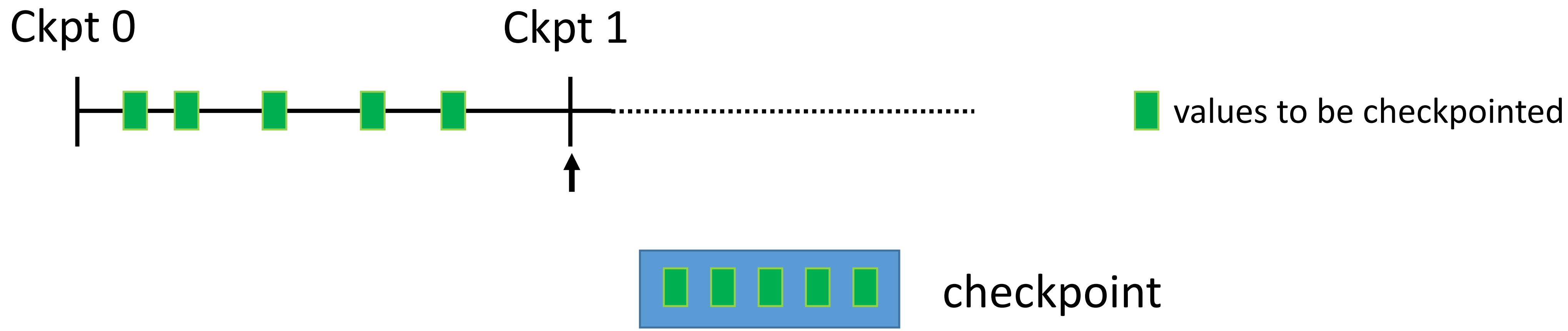
$$o_{recovery,ACR} \leq o_{recovery} \quad \textbf{\textit{iff}} \quad (o_{rollback,ACR} + o_{recomp,ACR}) \leq o_{rollback}$$

# Baseline Checkpointing

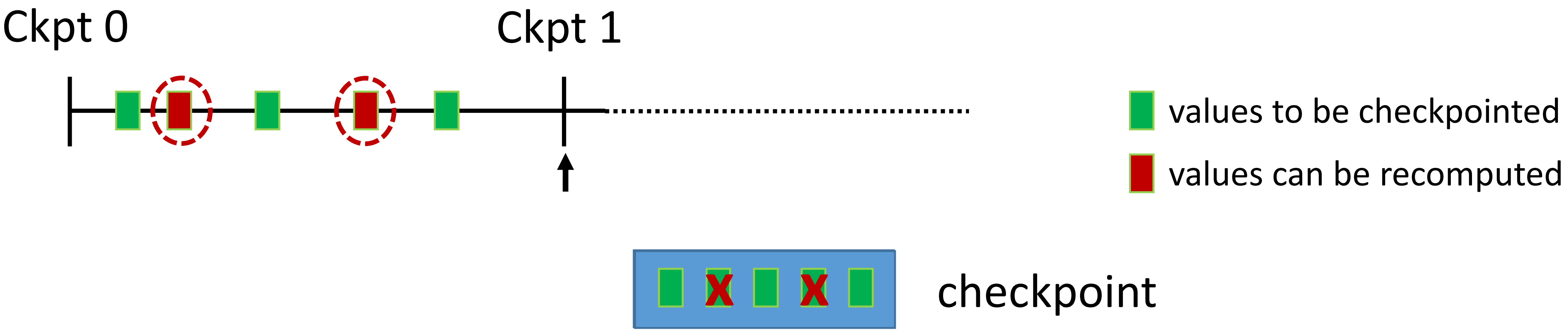
- Global checkpointing
- In-memory
- Log-based

# Baseline Checkpointing

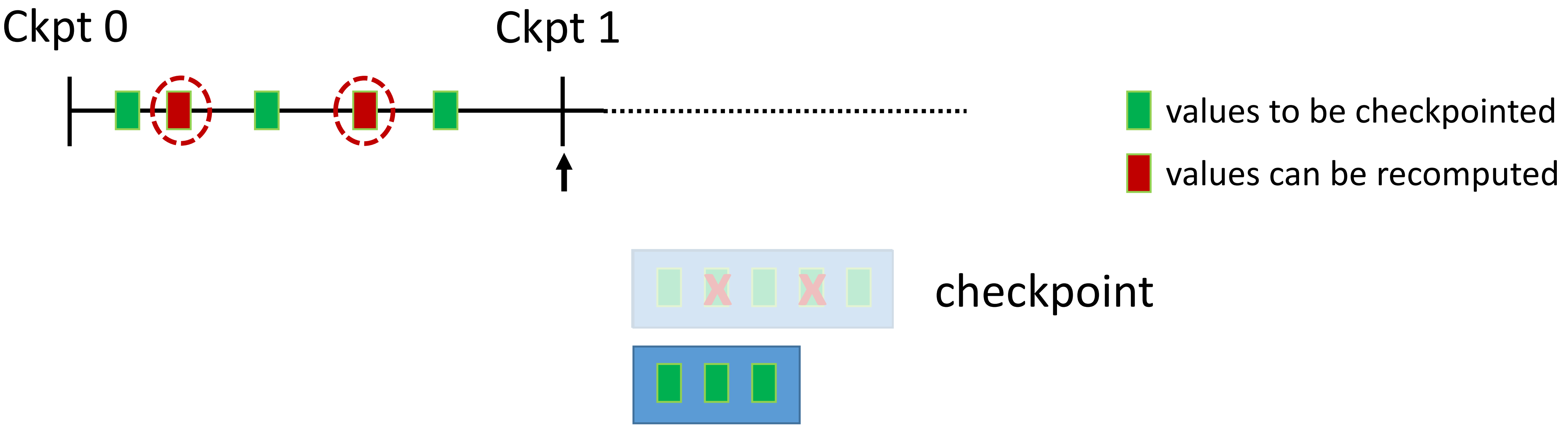
# Baseline Checkpointing



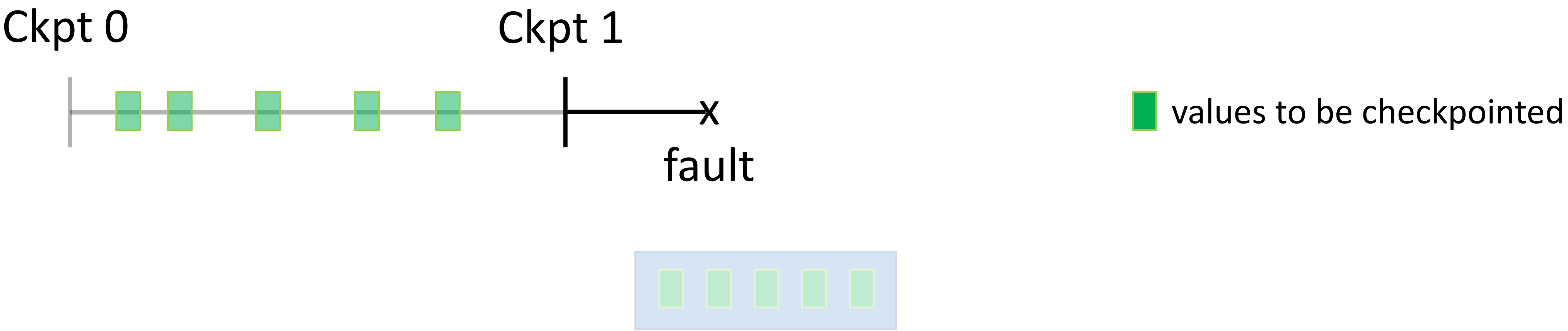
# Amnesic Checkpointing



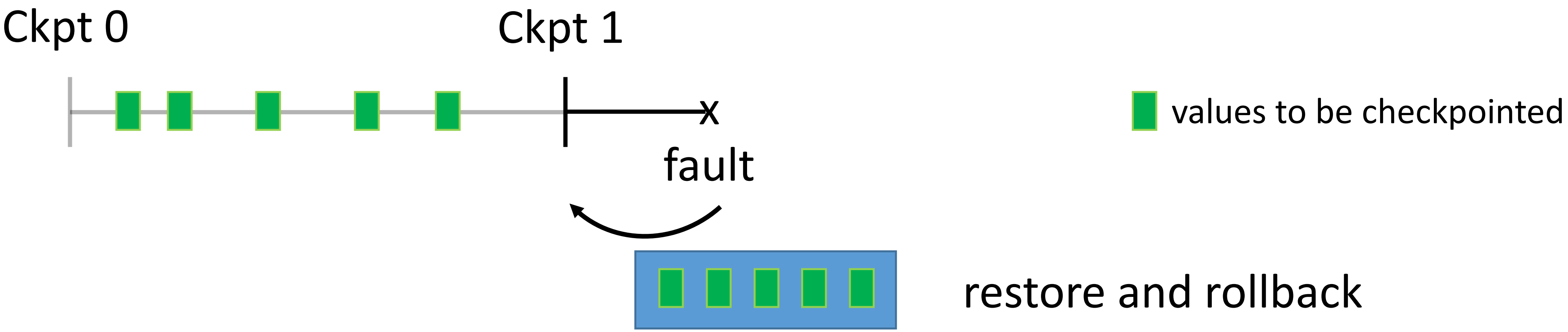
# Amnesic Checkpointing



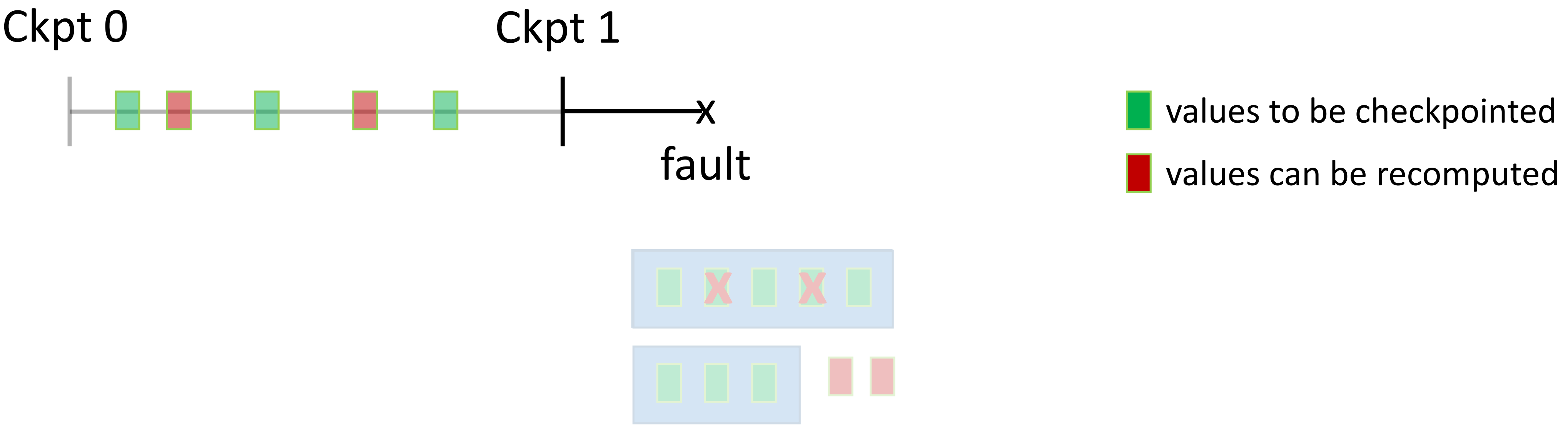
# Baseline Recovery



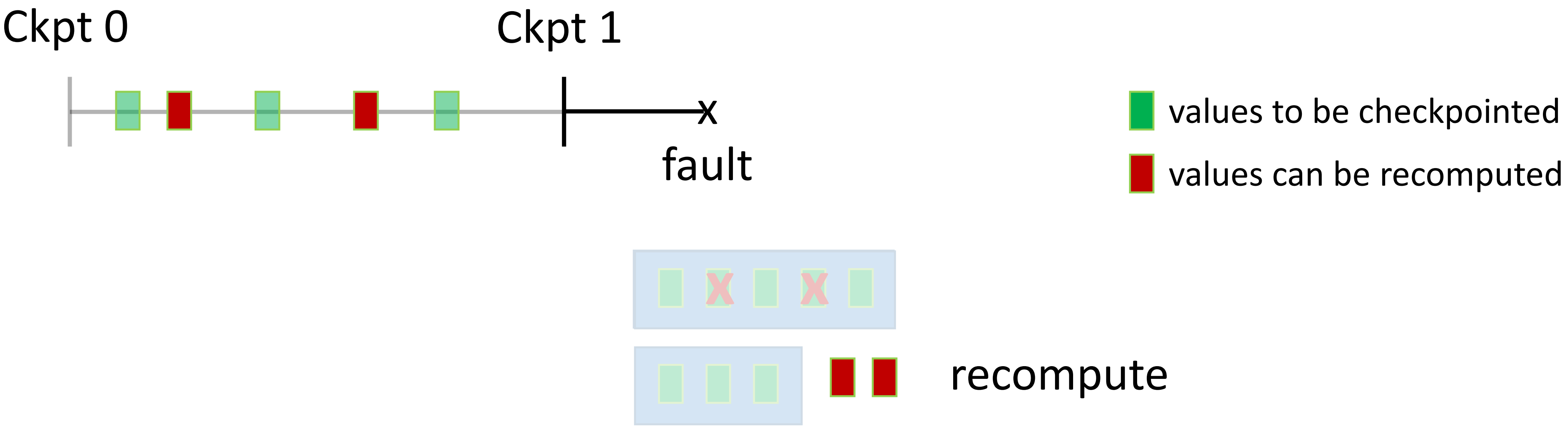
# Baseline Recovery



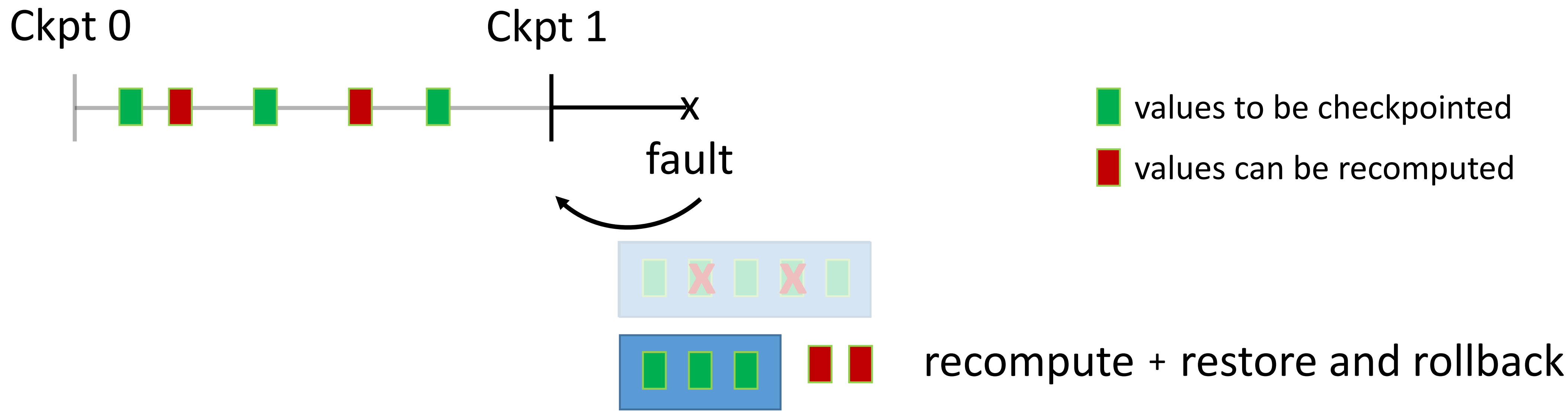
# Amnesic Recovery



# Amnesic Recovery



# Amnesic Recovery



# How to Recompute a Value?

```
int sumArr[10]
load (i)
load (j)
load (k)
k = i / j
while ( i <= 10 )
    sumArr[i] = i +j
    j = i * j
    incr i
    if (k > 1 )
        incr k
store (sumArr)
```

# How to Recompute a Value?

```
int sumArr[10]
load (i)
load (j)
load (k)
k = i / j
while ( i <= 10 )
    sumArr[i] = i +j
    j = i * j
    incr i
    if (k > 1 )
        incr k
store (sumArr)
```

# How to Recompute a Value?

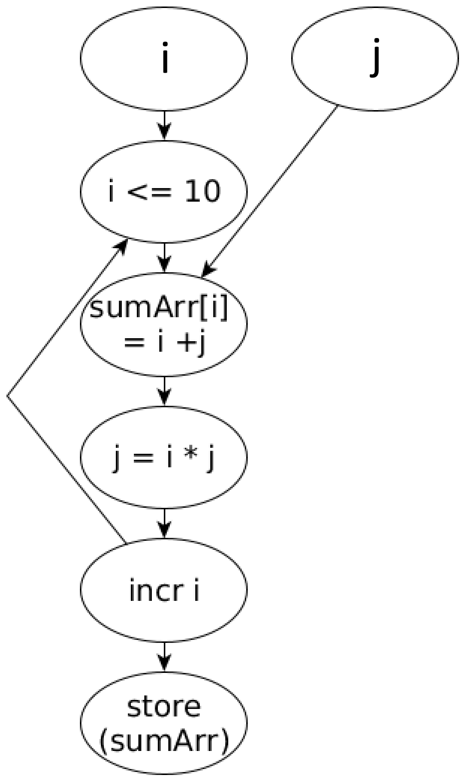
```
int sumArr[10]
load (i)
load (j)
load (k)
k = i / j
while ( i <= 10 )
    sumArr[i] = i +j
    j = i * j
    incr i
    if (k > 1 )
        incr k
store (sumArr)
```

# How to Recompute a Value?

```
int sumArr[10]
load (i)
load (j)
load (k)
k = i / j
while ( i <= 10 )
    sumArr[i] = i +j
    j = i * j
    incr i
    if (k > 1 )
        incr k
store (sumArr)
```

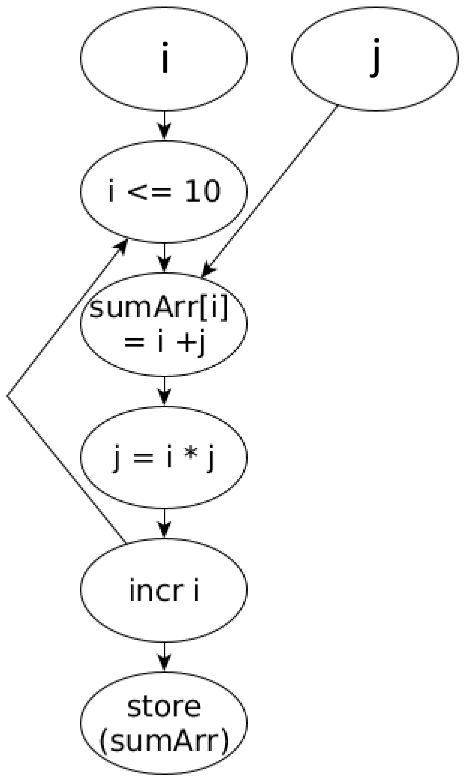
# How to Recompute a Value?

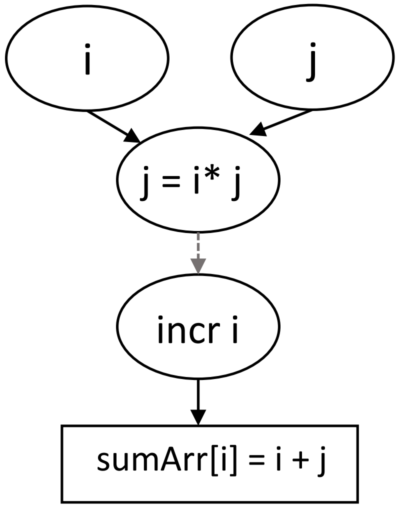
```
int sumArr[10]
load (i)
load (j)
load (k)
k = i / j
while ( i <= 10 )
    sumArr[i] = i +j
    j = i * j
    incr i
    if (k > 1 )
        incr k
store (sumArr)
```



# How to Recompute a Value?

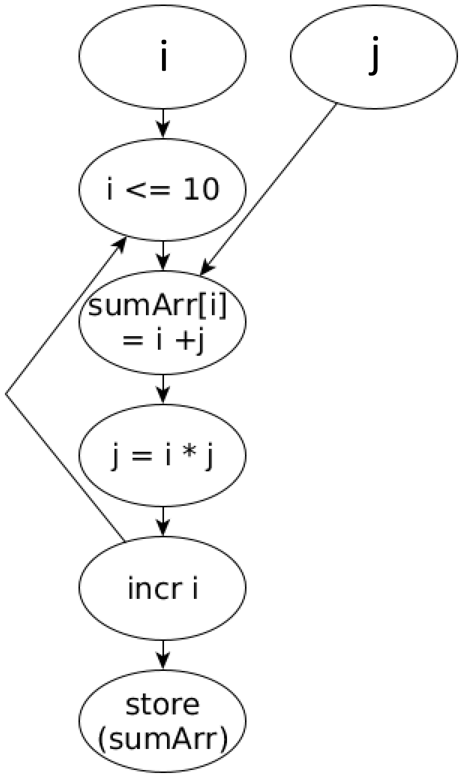
```
int sumArr[10]
load (i)
load (j)
load (k)
k = i / j
while ( i <= 10 )
      sumArr[i] = i +j
      j = i * j
      incr i
      if (k > 1 )
            incr k
store (sumArr)
```

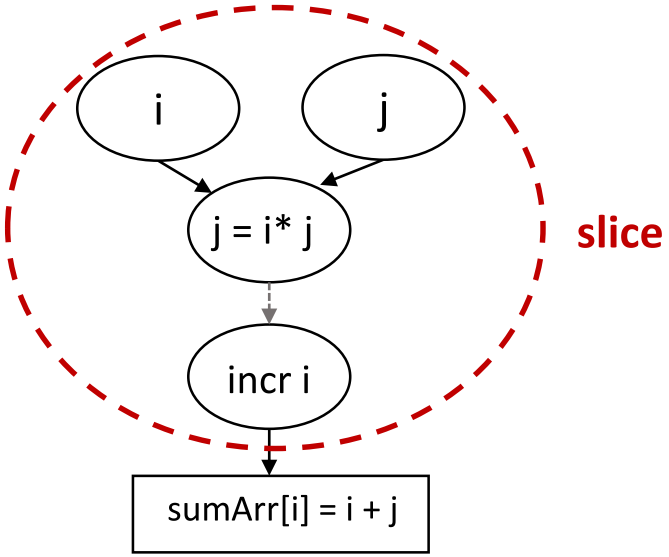


# How to Recompute a Value?

```
int sumArr[10]
load (i)
load (j)
load (k)
k = i / j
while ( i <= 10 )
    sumArr[i] = i +j
    j = i * j
    incr i
    if (k > 1 )
        incr k
store (sumArr)
```

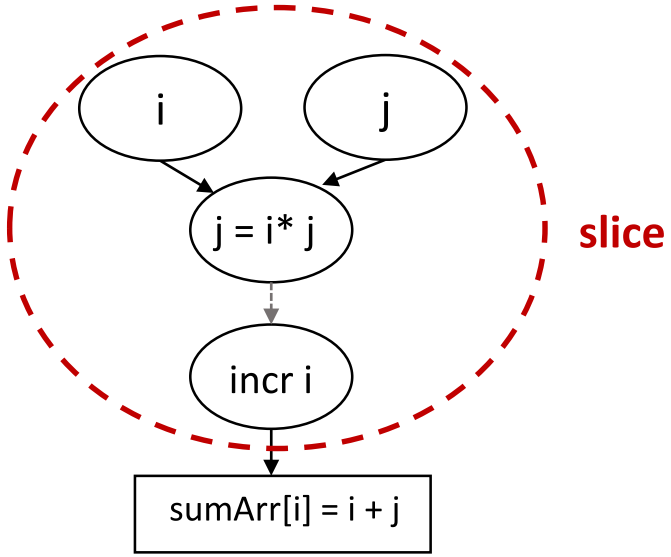


# How to form Slices?

- Compiler identifies them
  - Select cost-effective ones (i.e., short ones)

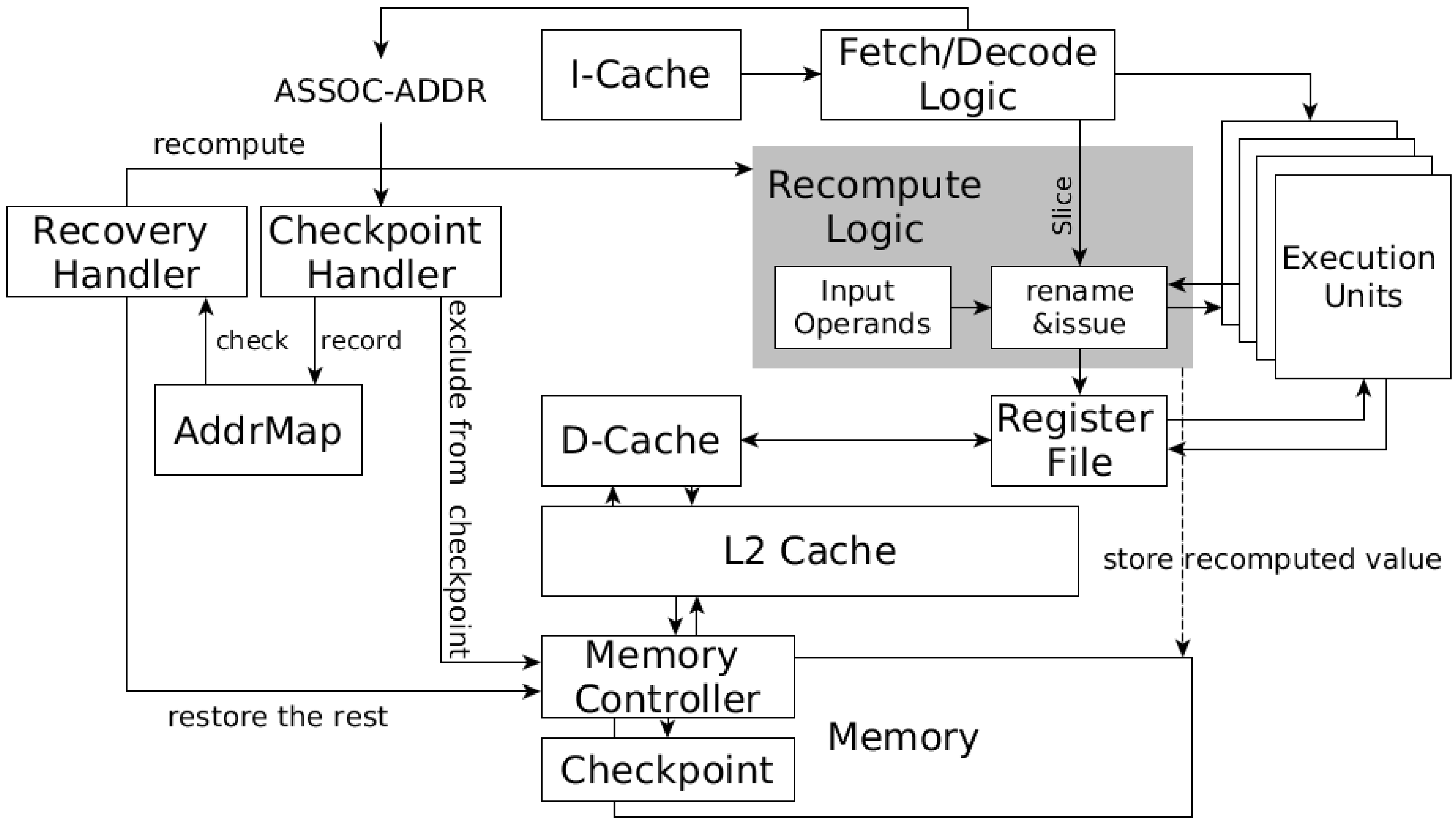
$$(o_{rollback,ACR} + o_{recomp,ACR}) \leq o_{rollback}$$



# How to use Slices?

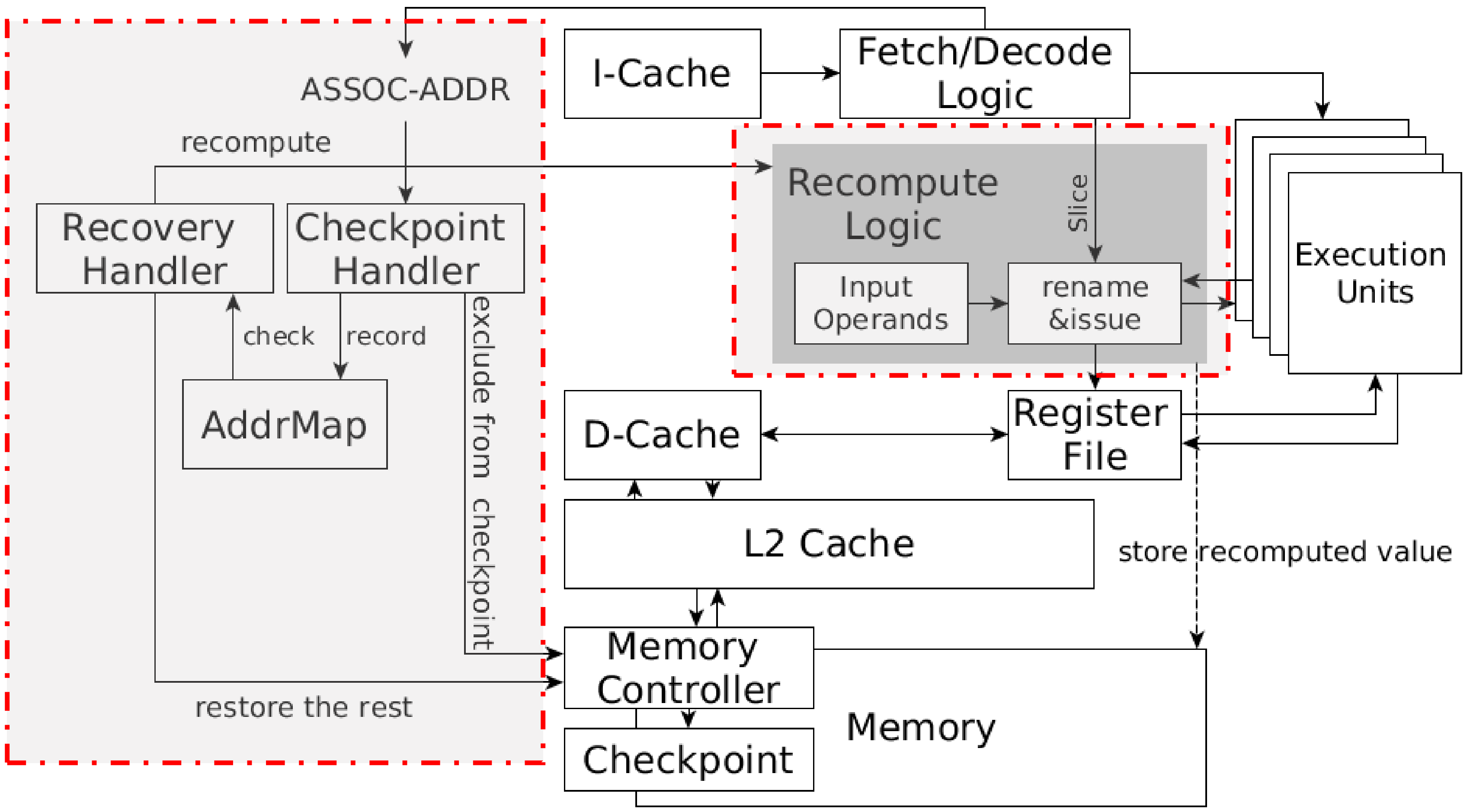
- Need to know start address of the slice
- Need to communicate it to runtime
  - **ASSOC-ADDR:** <memory address,  slice address>
    - automatically executed with the corresponding store
    - <memory address,  slice address> is recorded in buffer: **AddrMap**
- If recovery needed
  - Look into AddrMap for active Slices
  - Recompute values whose Slices are recorded in AddrMap

# Amnesic Checkpoint and Recovery



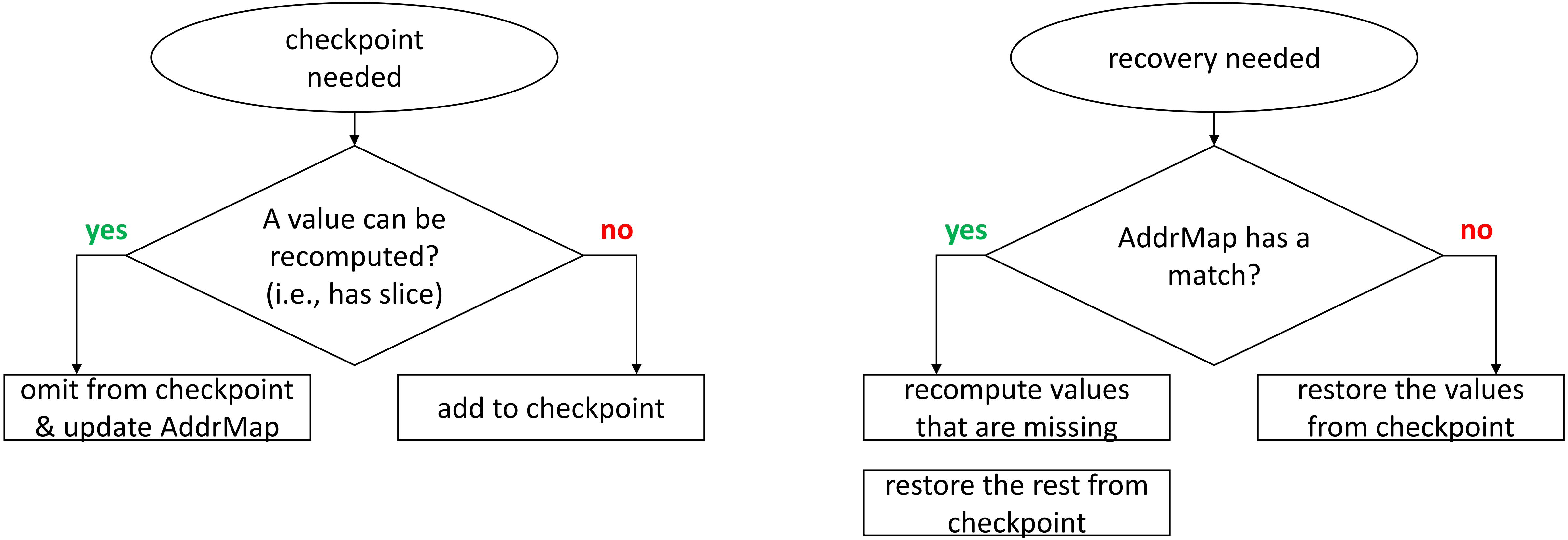

ACR: Amnesic Checkpointing and Recovery

# Amnesic Checkpoint and Recovery




ACR: Amnesic Checkpointing and Recovery

# Amnesic Checkpoint and Recovery



# Evaluation

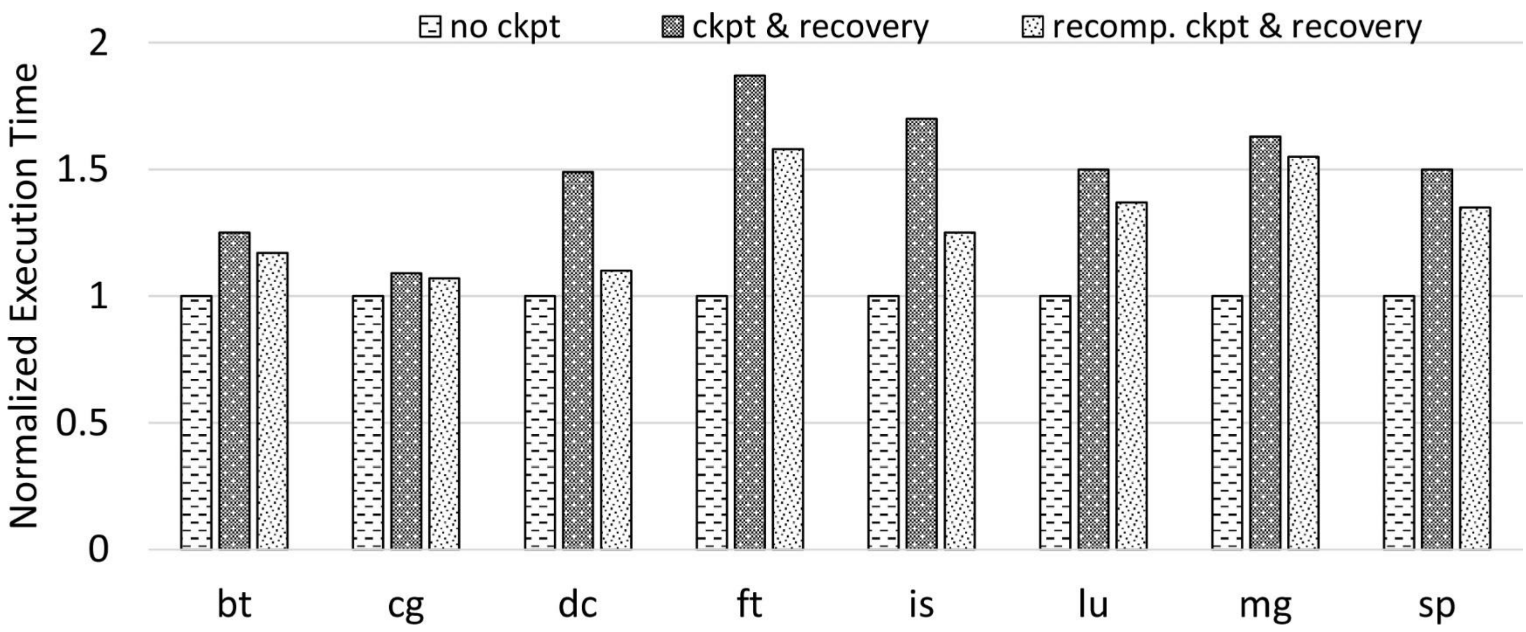
# Recomputation Enabled Checkpointing – Setup

- NAS benchmarks
- Amnesic compiler pass mimicked by binary instrumenting Pintool
- Microarchitecture and scheduler implemented in Snipersim

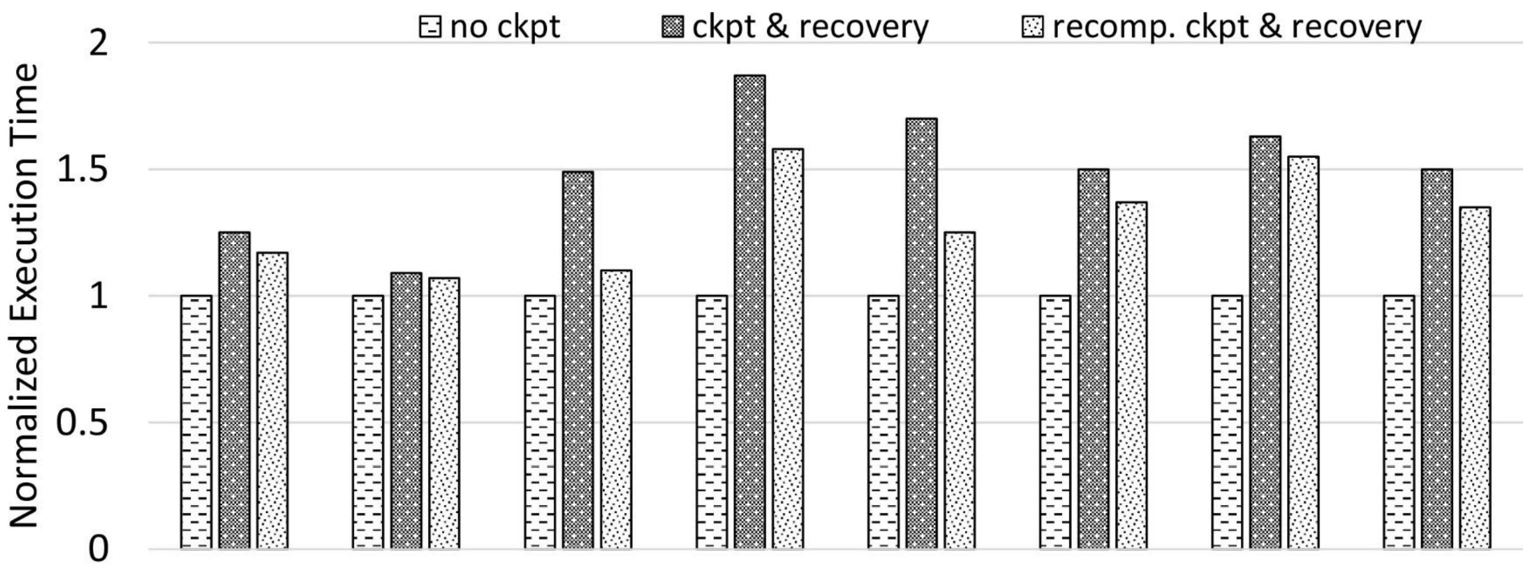
- 1.09 GHz, 4-issue, inorder core: 8/16/32 cores and 8/16/32 threads
- L1: 32KB, 4-way
- L2: 512KB, 8-way

- Ideal Baseline (no checkpoint, no recovery)
- Global coordinated
- Local coordinated

# Performance Overhead – Checkpoint and Recovery

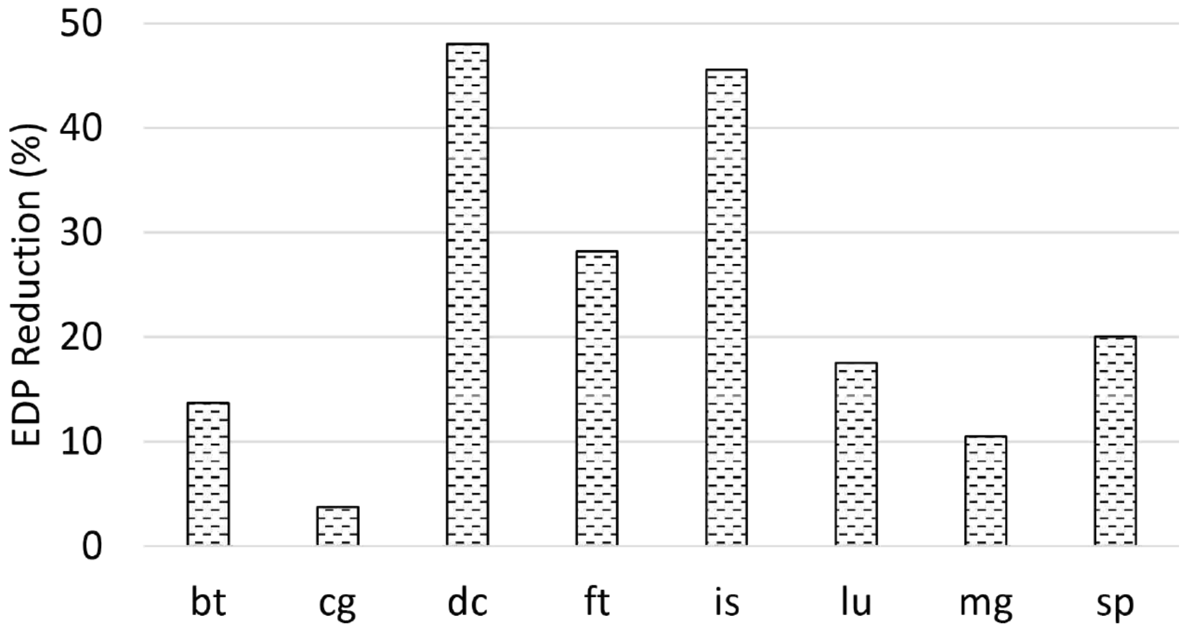


# Performance Overhead – Checkpoint and Recovery

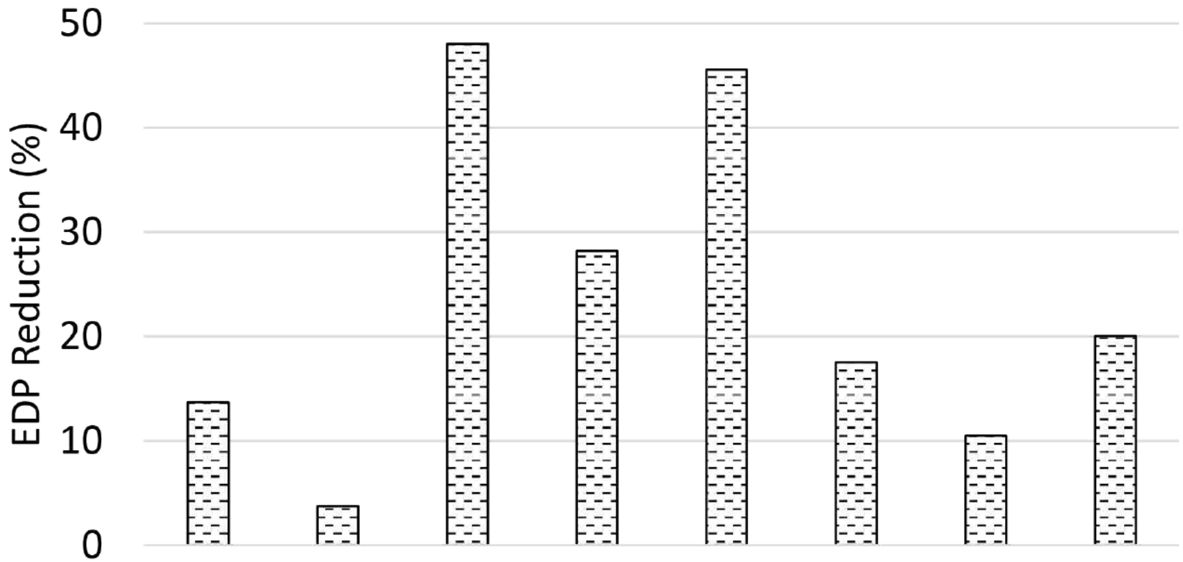



**up to 26.68% (12.39% on average) reduction on checkpoint and recovery overhead**

# EDP Reduction – Recovery

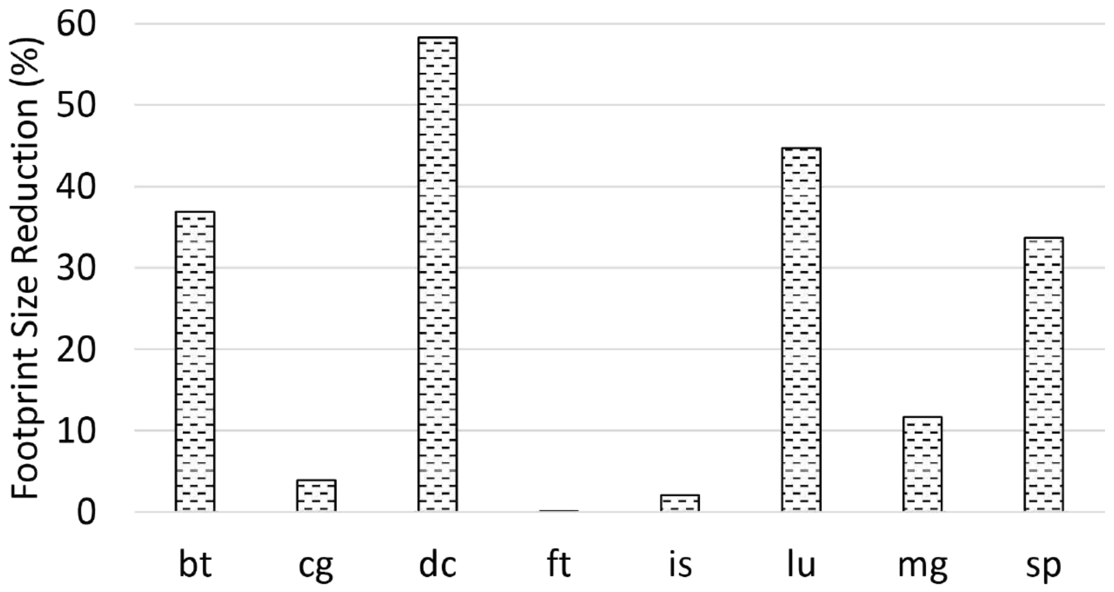


# EDP Reduction – Recovery

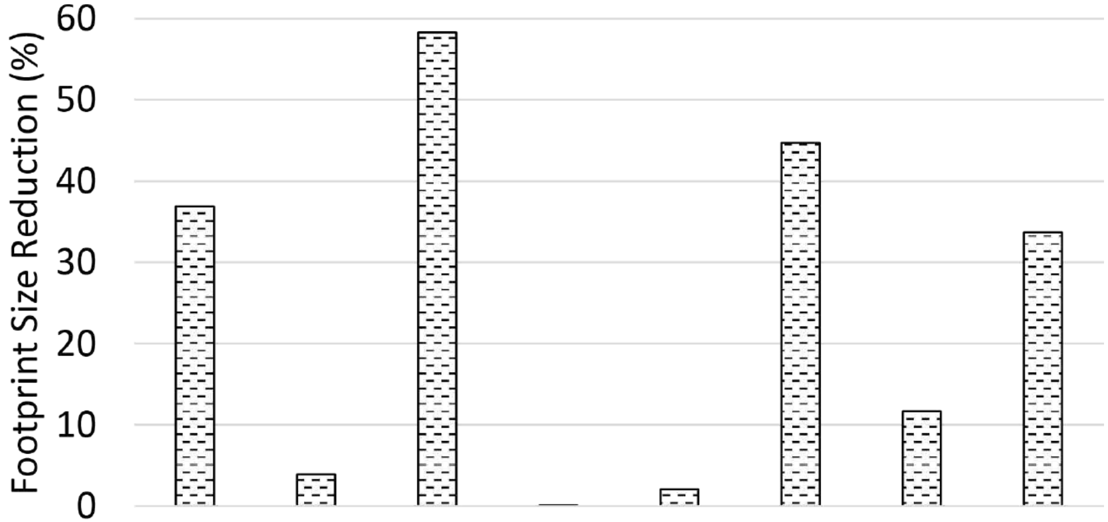



**up to 48.07% (23.41% on average) EDP gain**

# Footprint Size Reduction

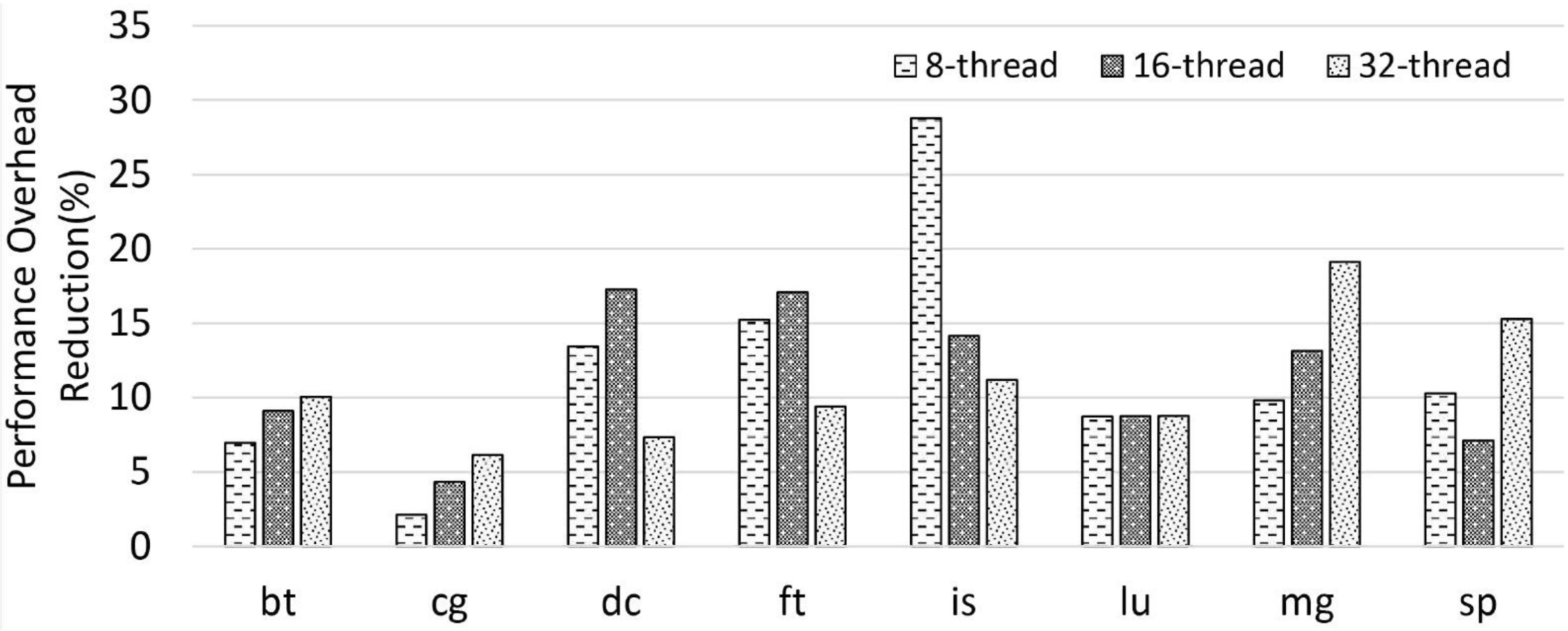


# Footprint Size Reduction

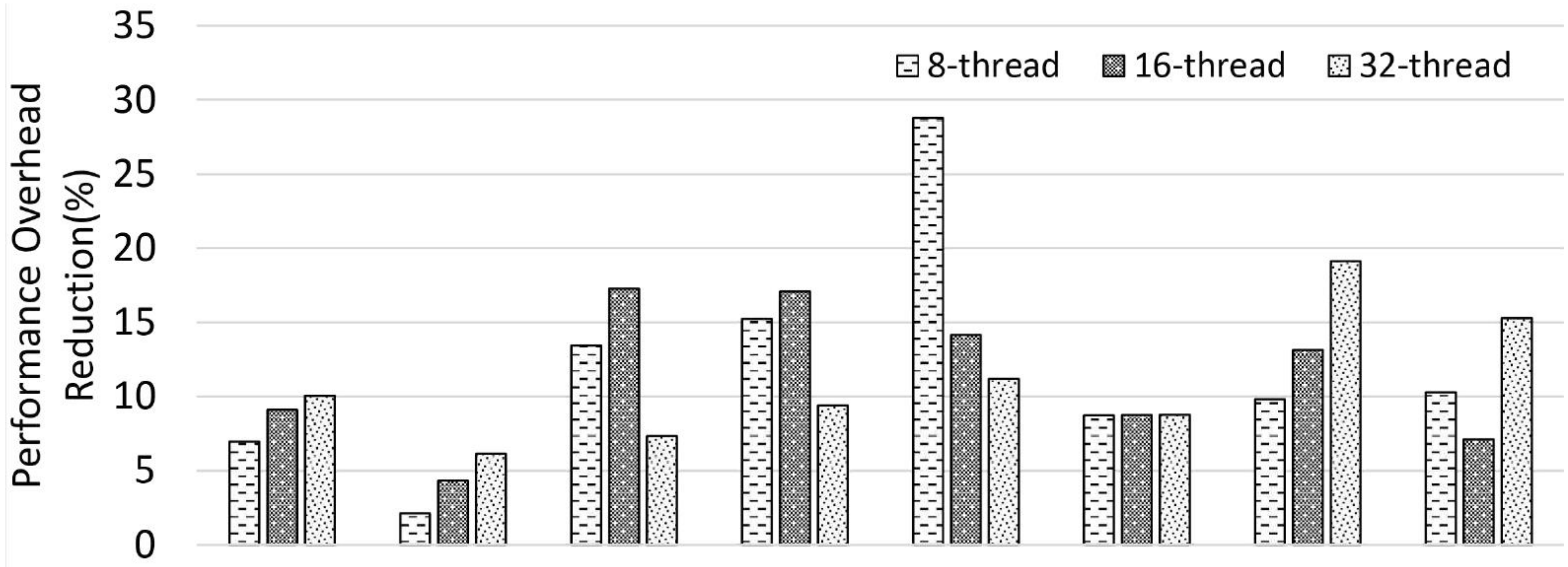



**up to 58.3% (23.91% on average) memory footprint size reduction**

# Thread Count

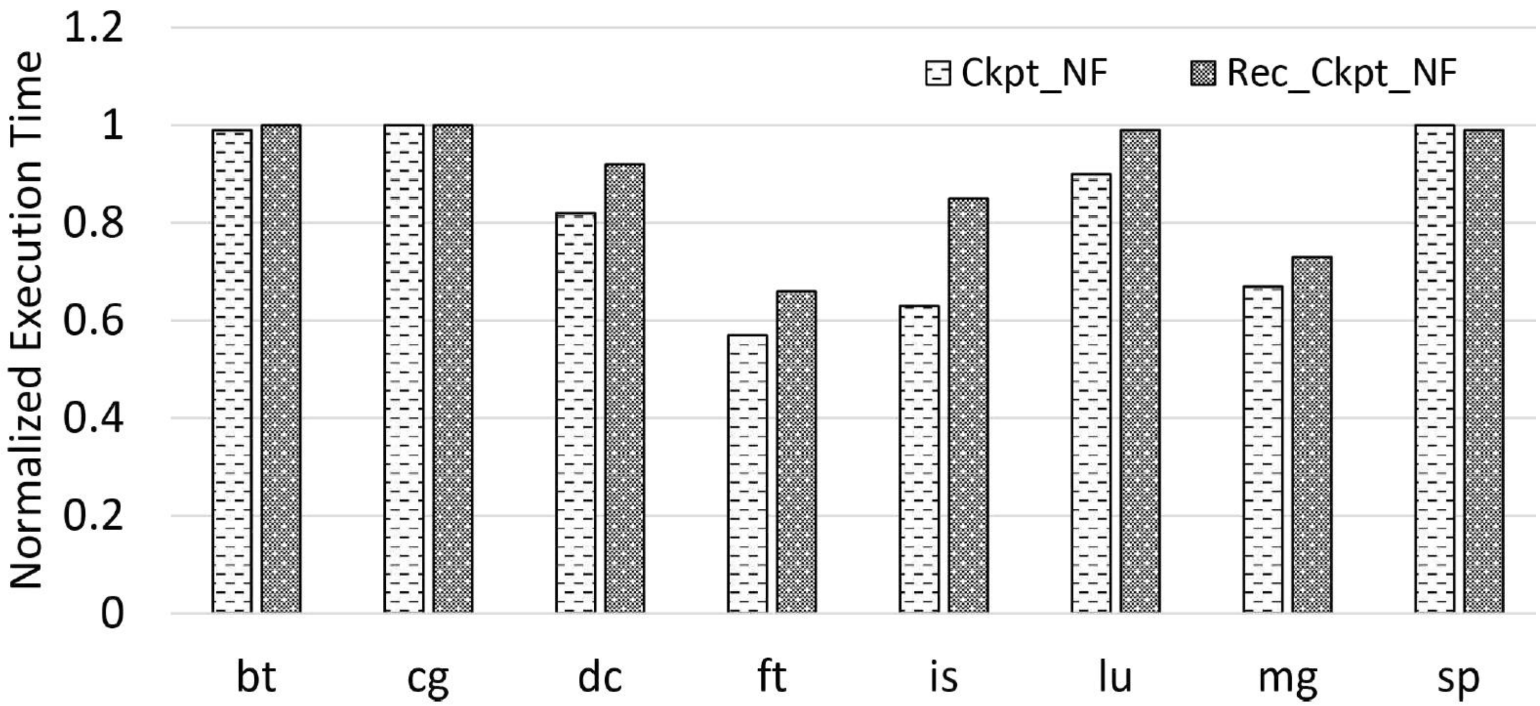


# Thread Count

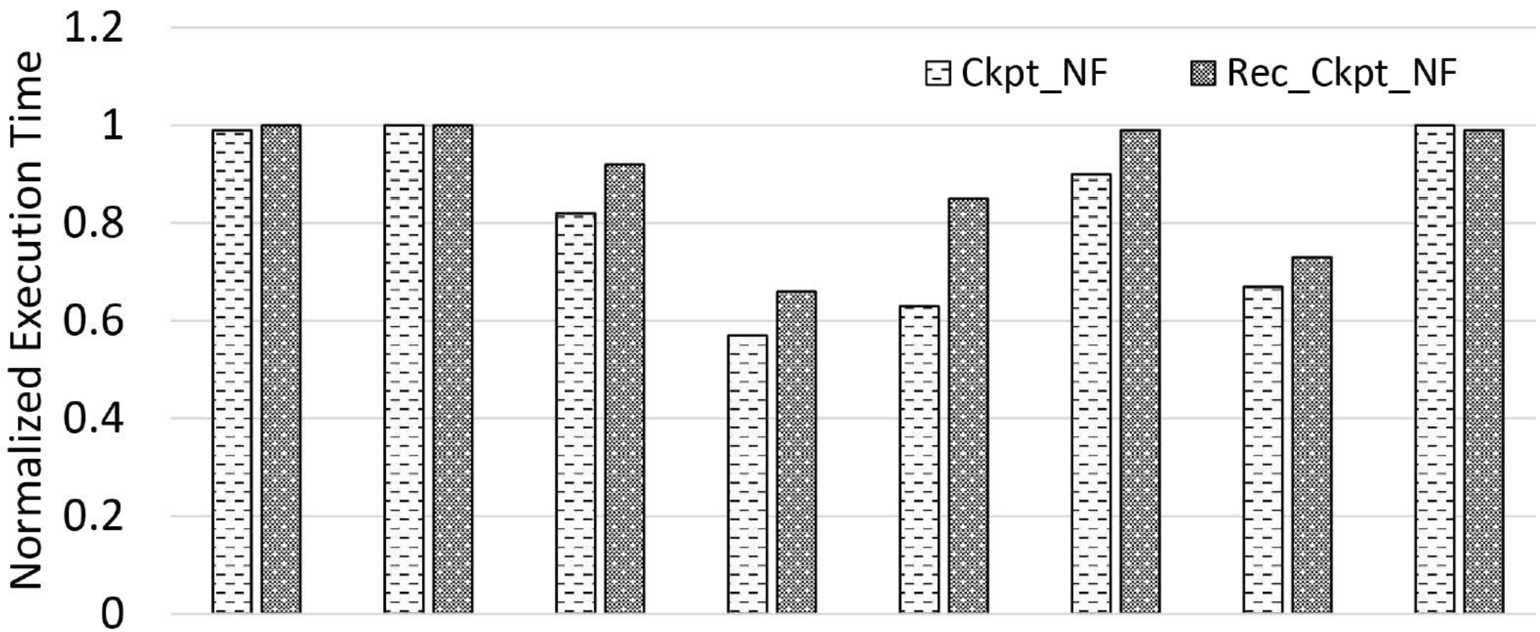



**up to 28%, 17% (is), and 19% (mg) reduction for 8-, 16-, 32-threads,respectively**

# Global vs Coordinated Local Checkpointing

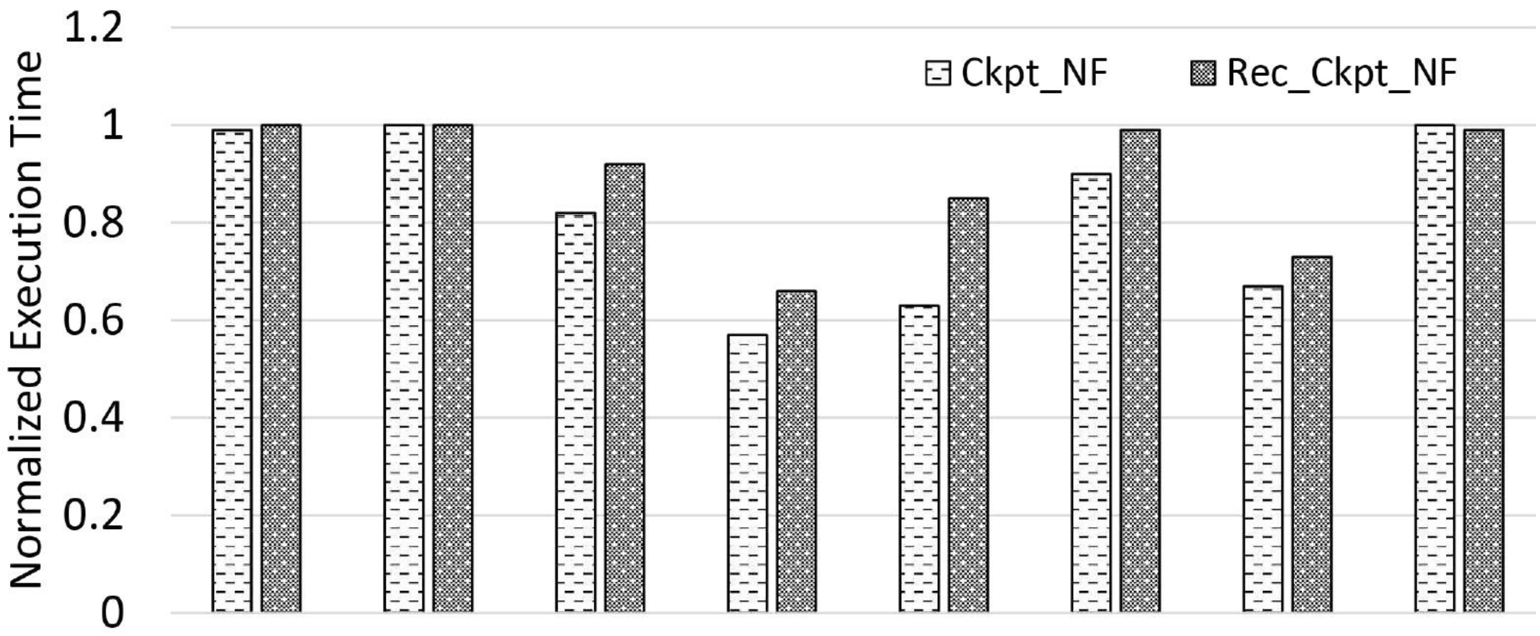


# Global vs Coordinated Local Checkpointing




**up to 42% (ft) reduction w.r.t. Ckpt_NF of global checkpointing**

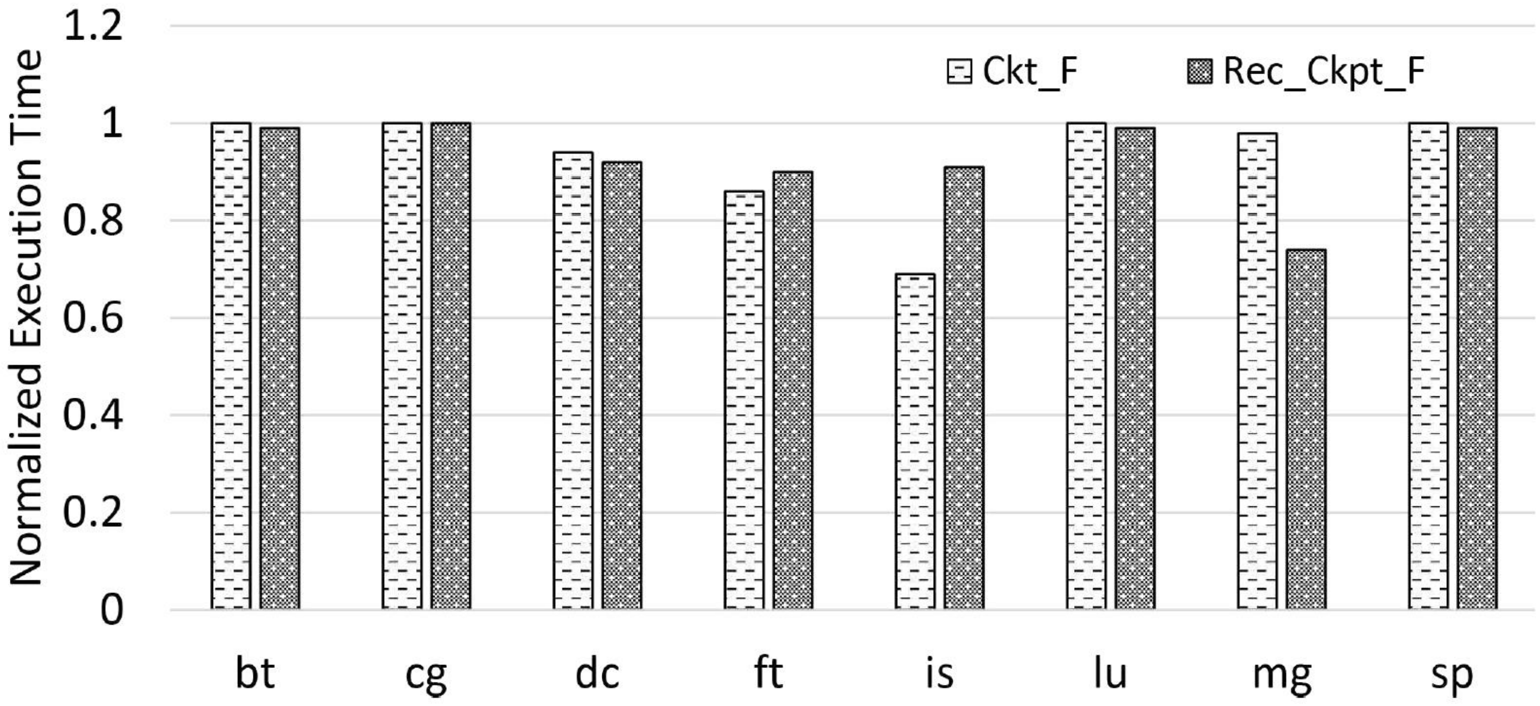
# Global vs Coordinated Local Checkpointing



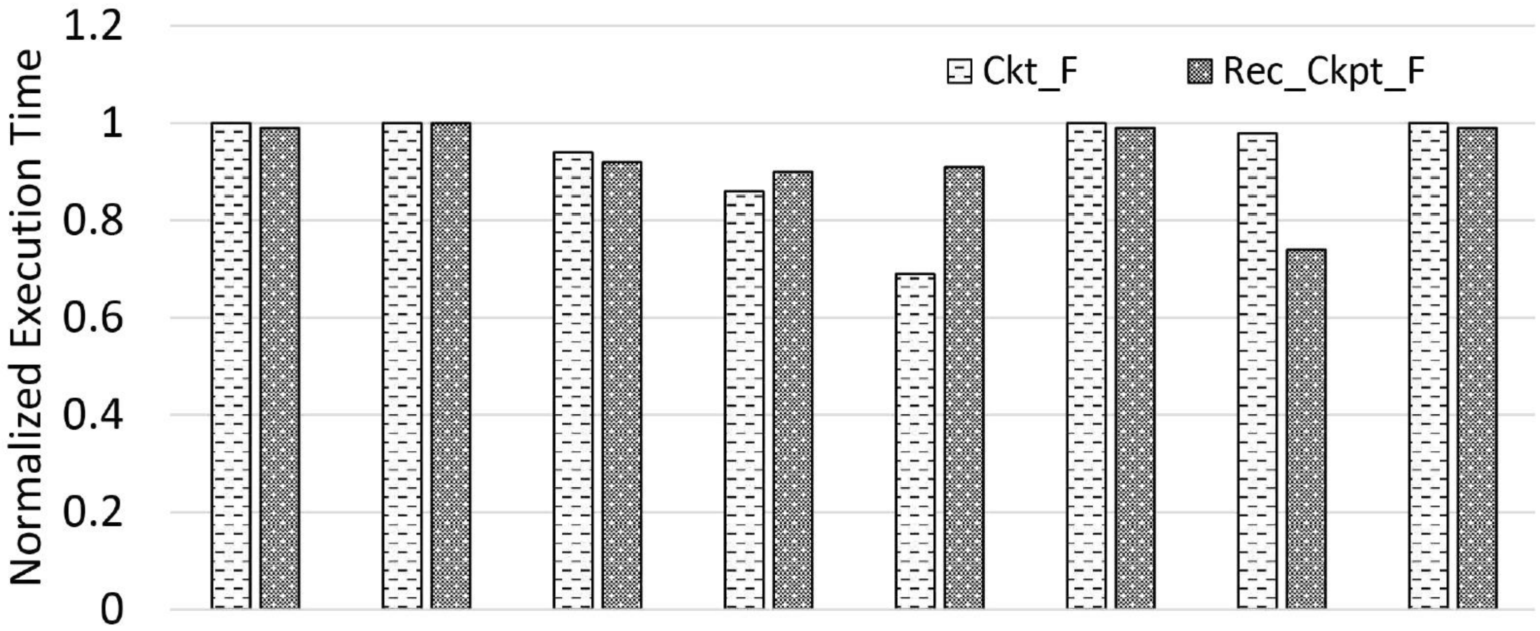

**up to 42% (ft) reduction w.r.t. Ckpt_NF of global checkpointing**

**up to 33% (ft) reduction w.r.t. Rec_Ckpt_NF of global checkpointing**

# Global vs Coordinated Local Checkpointing

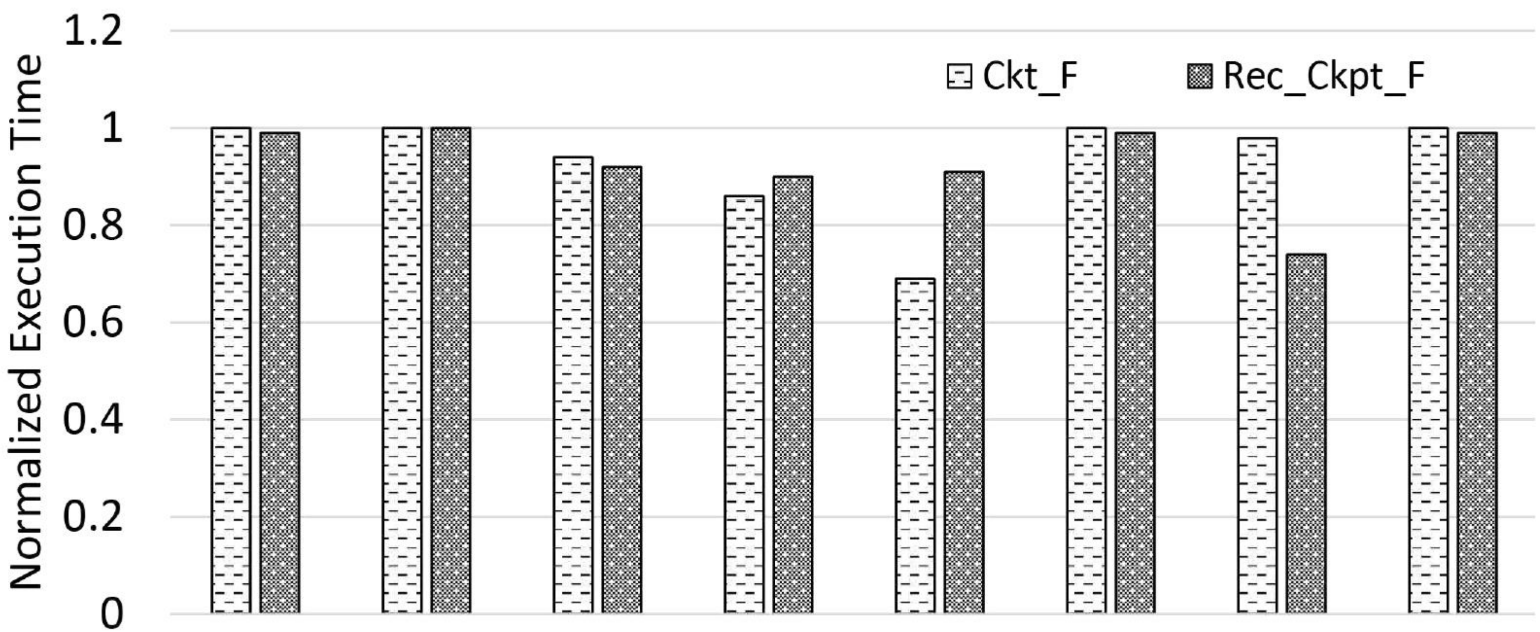


# Global vs Coordinated Local Checkpointing




**up to 31% (is) reduction w.r.t. Ckpt_F of global checkpointing**

# Global vs Coordinated Local Checkpointing




up to 31% (is) reduction w.r.t. Ckpt_F of global checkpointing

up to 26% (mg) reduction w.r.t. Rec_Ckpt_F of global checkpointing

# Summary

- Effective in reducing checkpoint overhead
  - Power and performance
- Reduces checkpoint footprint size (i.e., storage reduction)
- Low-cost recovery

# Questions and Comments

???