Explicitly Parallel Programming with Shared-Memory is Insane: At Least Make it Deterministic!

Joe Devie ark Oskin

Parallel Programming is Hard

- Race conditions make life difficult

```
barrier_wait();
x = 1;
```

```
barrier_wait();
x = 2;
```

```
return x;
```

100%

100%

Deterministic parallel execution would be nice!

Wouldn't it be nice if...

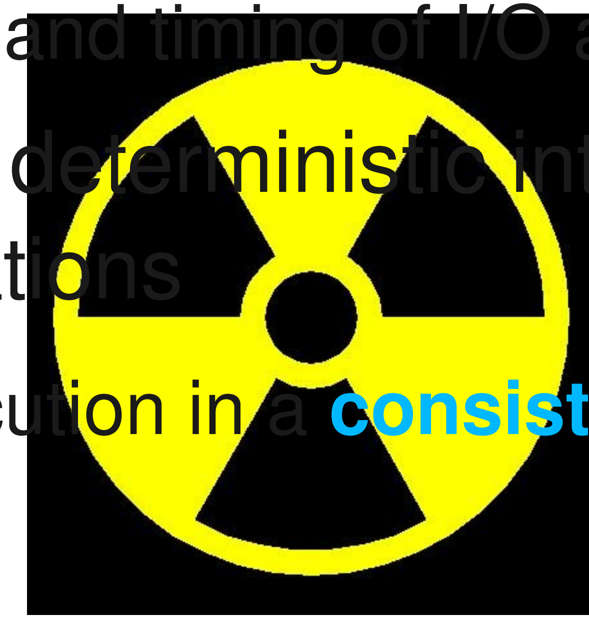
- ...execution were reproducible on a machine?
  - No more heisenbugs!
  - Run parallel programs forwards and backwards
- ...execution were reproducible *across* machines?
  - Reduces the parallel testing coverage problem to the single-threaded testing coverage problem
  - Can reproduce bugs found in the field
  - Increases robustness of deployed parallel code

Related Work

- Deterministic, implicitly-parallel languages
  - StreamIt [ASPLOS 2002], Jade [TOPLAS 1998]
  - Typically domain-specific
- Record+replay
  - RecPlay [TOCS 1999], FDR [ISCA 2003], Rerun [ISCA 2008], DeLorean [ISCA 2008]
  - Log ordering of memory operations
- Serialize execution (Simics [Computer 2002])

DMP: A Deterministic Multiprocessor

- Determinism: same input yields same output
  - What is “input”?
  - Input is value and timing of I/O and OS events
- DMP provides deterministic interleaving of memory operations
  - Serialize execution in a **consistent**, but **arbitrary** way

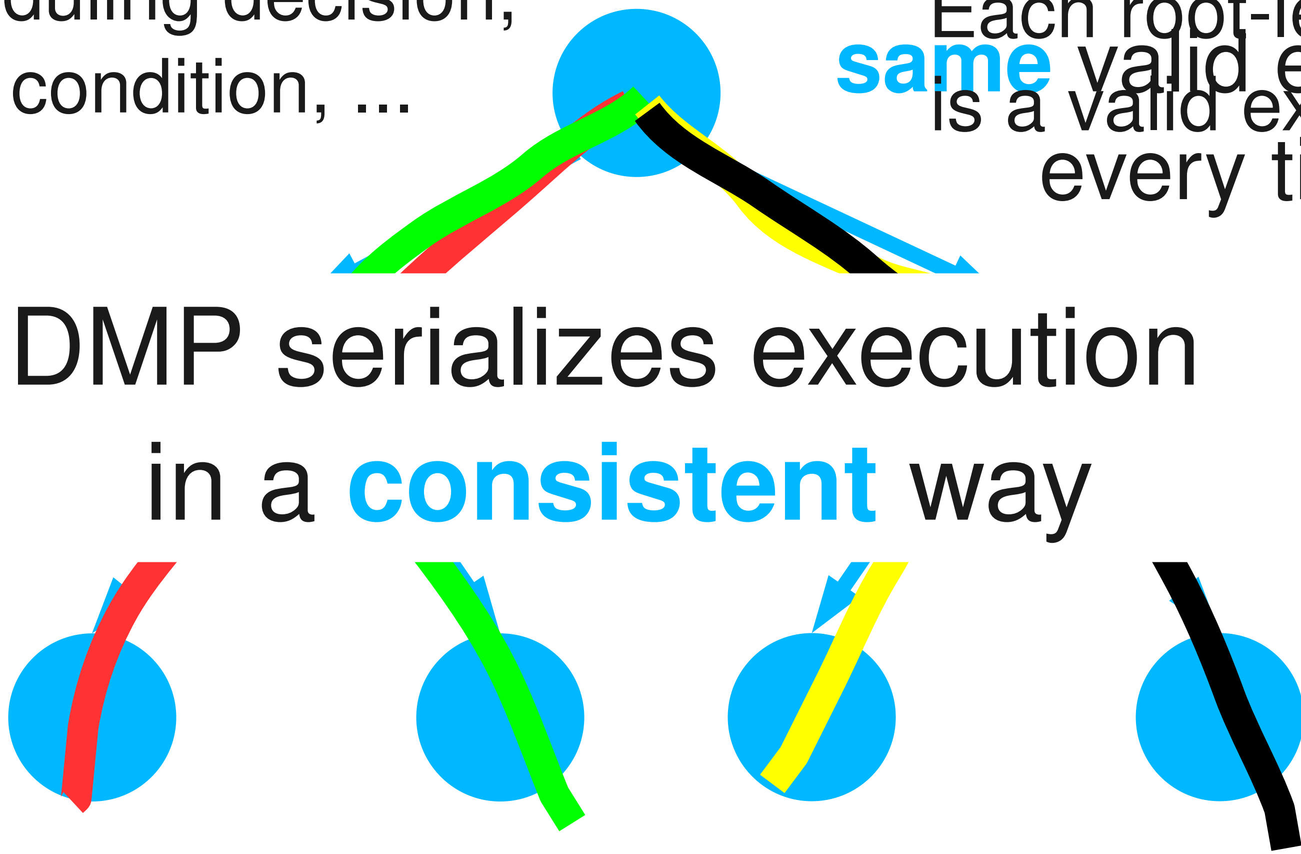


Valid Non-deterministic Executions

= lock,
scheduling decision,
race condition, ...

Each root-leaf path
is a valid execution

DMP picks the
same valid execution
every time



DMP serializes execution
in a **consistent** way

Serialized Execution

DT

```
store A
store P
```

```
store B
load A
```

```
store P'
store B
```

“Deterministic Token” gets passed after every insn

```
store A
```

```
store B
```

```
store P'
```

```
store P
```

```
load A
```

```
store B
```

6 steps

Serialized Execution

DT

```
store A
store P
```

```
store B
load A
```

```
store P'
store B
```

…n after every insn is expensive

…fter each n-insn **quantum** instead

```
store A
```

```
store P
```

```
store B
```

```
load A
```

```
store P'
```

```
store B
```

One of many possible serializations

6 steps

Serialized Execution: Flow of Data

A particular serialization enforces
a particular flow of data

`store A`
`store P`

`store B`
`load A`

`store P'`
`store B`

DMP enforces the same serialization across program runs

DMP Interface and Implementation

- Interface: deterministically serialized execution
  - Preserves program behavior across runs
- Naïve implementation:
  - Execute insns in round-robin order
  - *n*x slowdown on *n* threads :-(
- Better implementation: “OoO superscalar”
  - Serialize only **when** necessary
  - Serialize only **for as long as** necessary

Recovering Parallelism

- Parallelize thread-private accesses
  - Sharing Table
- Speculatively parallelize execution
  - Transactional Memory (TM)
- TM + Speculative Value Forwarding
  - TM-Forward
- Smarter Quantum Building

DMP-Sharing Table: Exploiting Thread-private Data

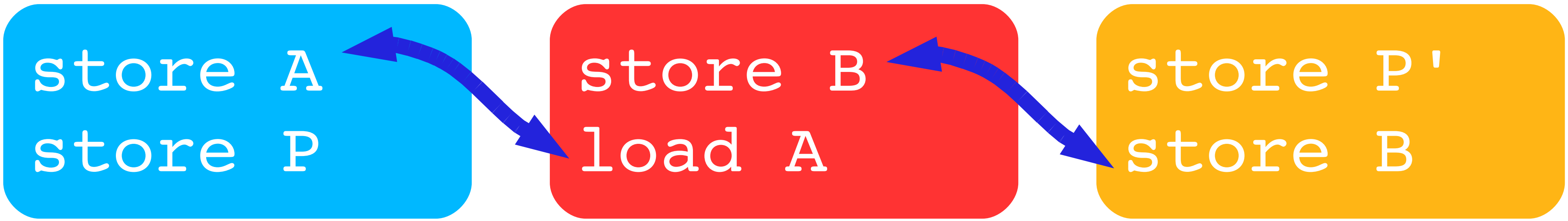
```
store A
store P
```

```
store B
load A
```

```
store P'
store B
```

- Thread-private accesses can't affect other threads
  - Okay to execute private accesses in parallel
- Sharing Table: locations are Shared or Private
  - Shared = S state, Private = M/E state
  - Need to hold DT to update sharing table

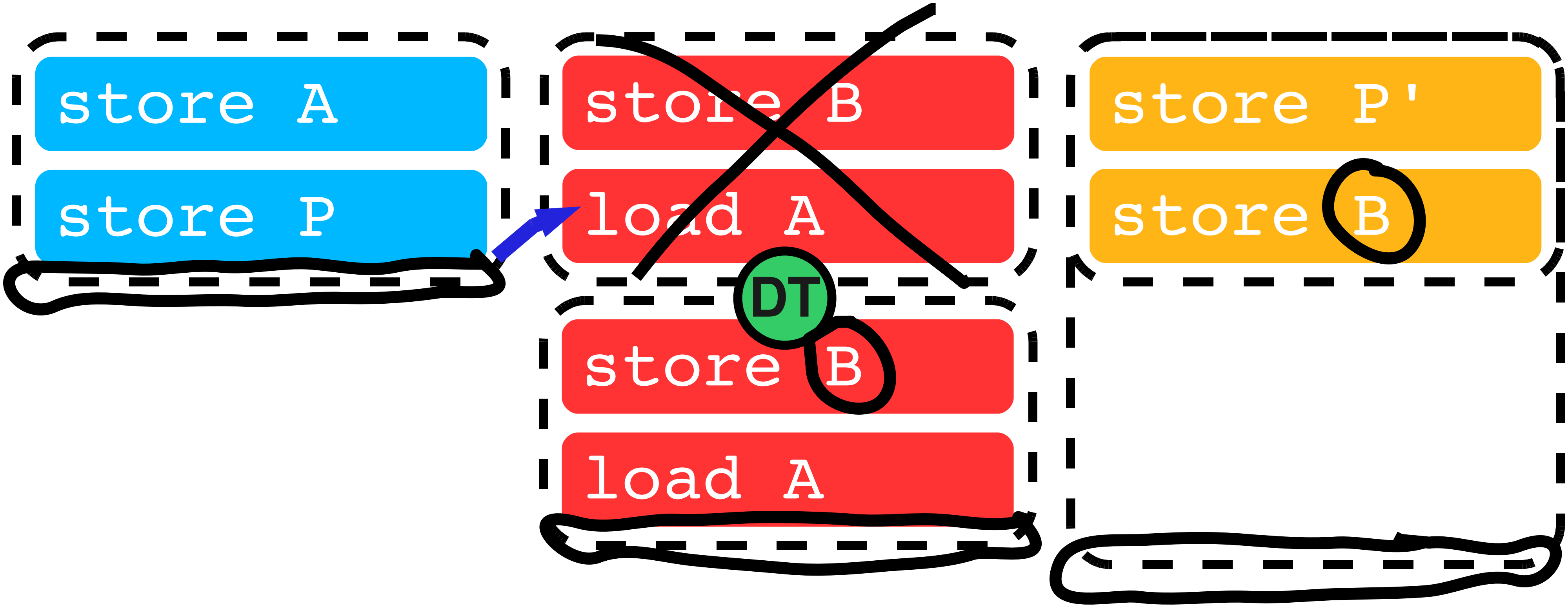
DMP-TM: Leveraging Speculation



- Execute quanta as implicit transactions
  - Quanta execute speculatively in parallel
    - Abort+retry if serialization was violated
  - **Commit quanta in order** (need DT to commit)

DMP-TM Execution

DMP-TM Execution



Ordering+isolation = “memory renaming”
WAW/WAR are “false” conflicts

4 ~~6~~ steps

DMP-TM-Forward: Speculative Value Forwarding

```
store A
store P
```

```
store B
load A
```

```
store P'
store B
```

- Speculatively forward values to "future" quanta
  - Can potentially avoid squashes even with true (RAW) data dependences
  - Must squash yourself if data you were forwarded is overwritten by "past" quantum
  - When you squash, must squash all your consumers

TM-Forward Execution

2 ~~4~~ ~~6~~ steps

Same **serial** flow of data,

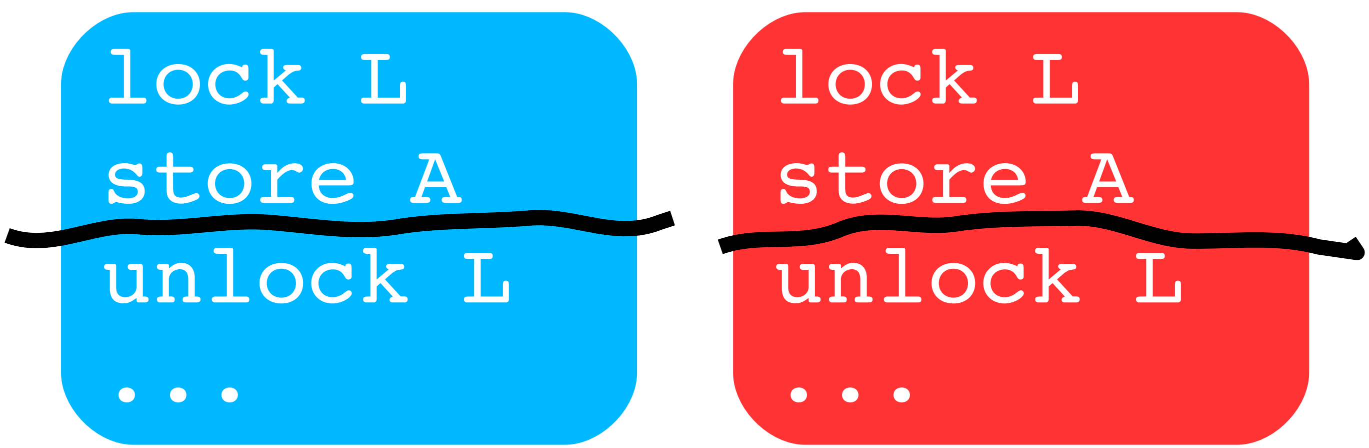
but **highly parallel** execution!

Recovering Parallelism

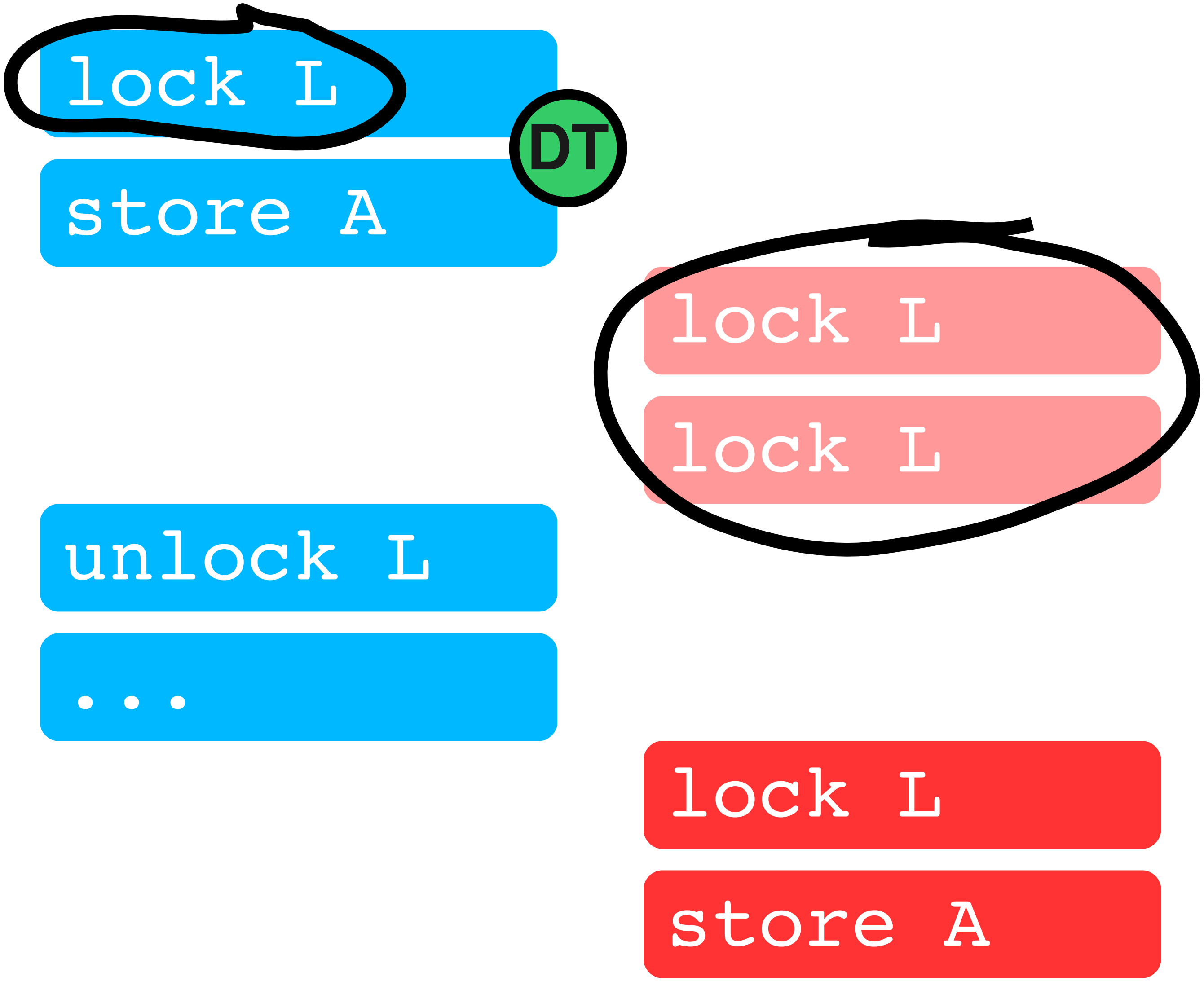
- Sharing Table
  - Parallelizes accesses to thread-private data
  - Non-speculative
- TM and TM-Forward
  - Speculation allows for more parallelism
  - “memory renaming” means fewer squashes
- Smarter Quantum Building

Quantum Building

- Building quanta by just counting dynamic insns is simple, but can be slow

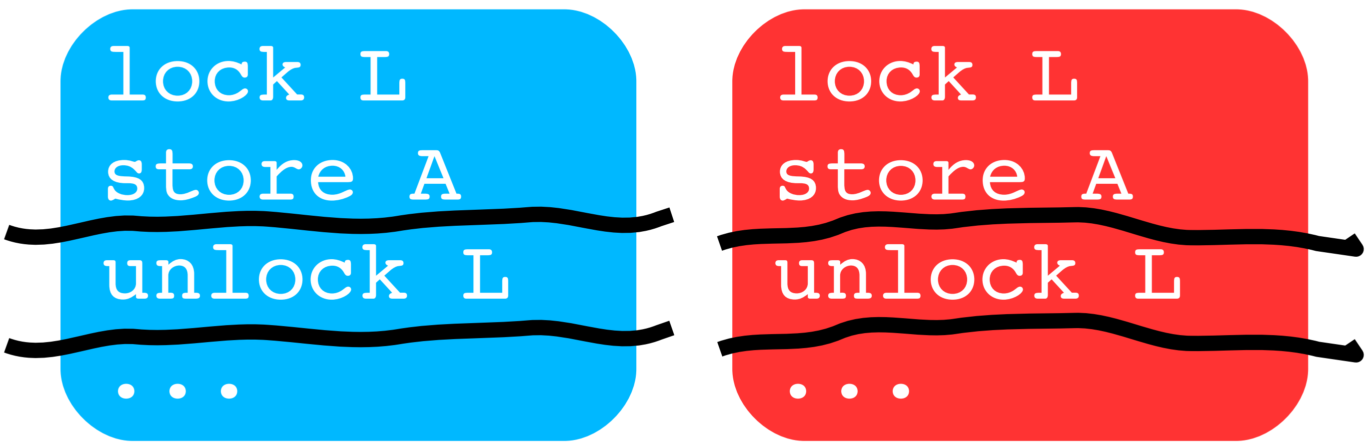


Naïve Quantum Building: Convoying



Smarter Quantum Building

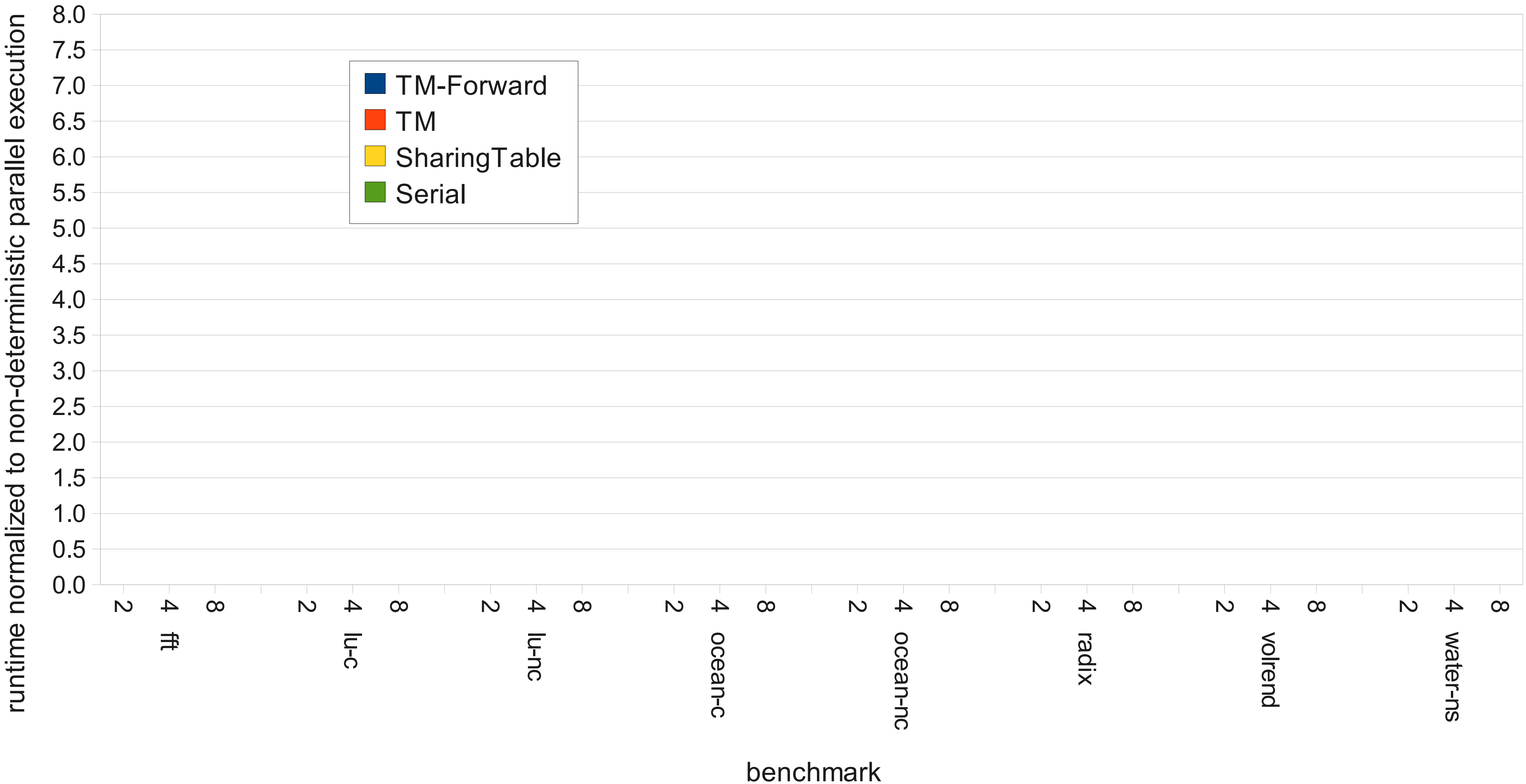
- Enclose critical sections in a single quantum!
  - Start new quantum after an “unlock”
- Other quantum building strategies in paper



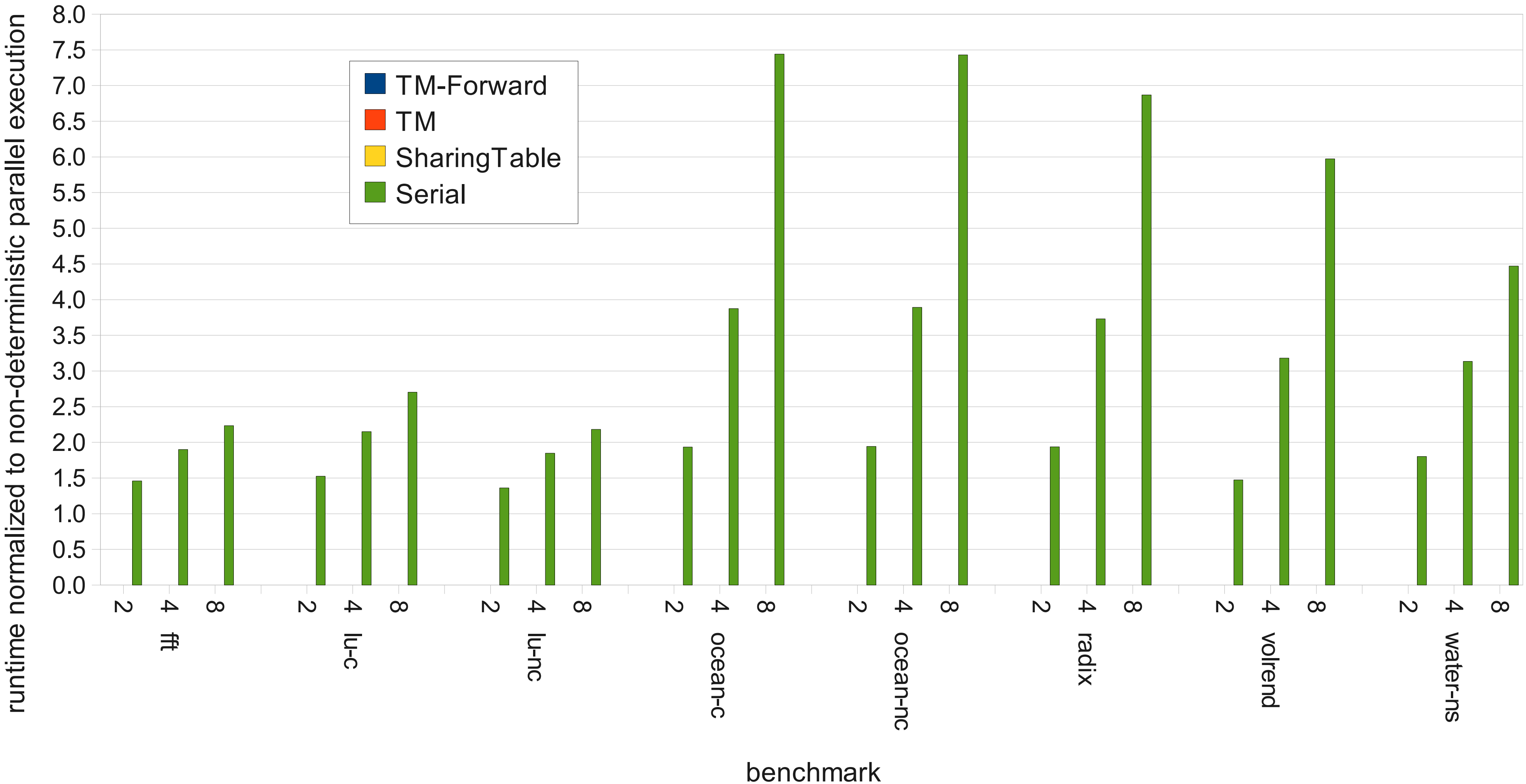
Experimental Methodology

- Simulator using PIN
  - Functionally models effects of serialization
  - Models address conflicts, limited TM buffering
  - Assume 1 IPC, free commits
- SPLASH2 benchmark suite

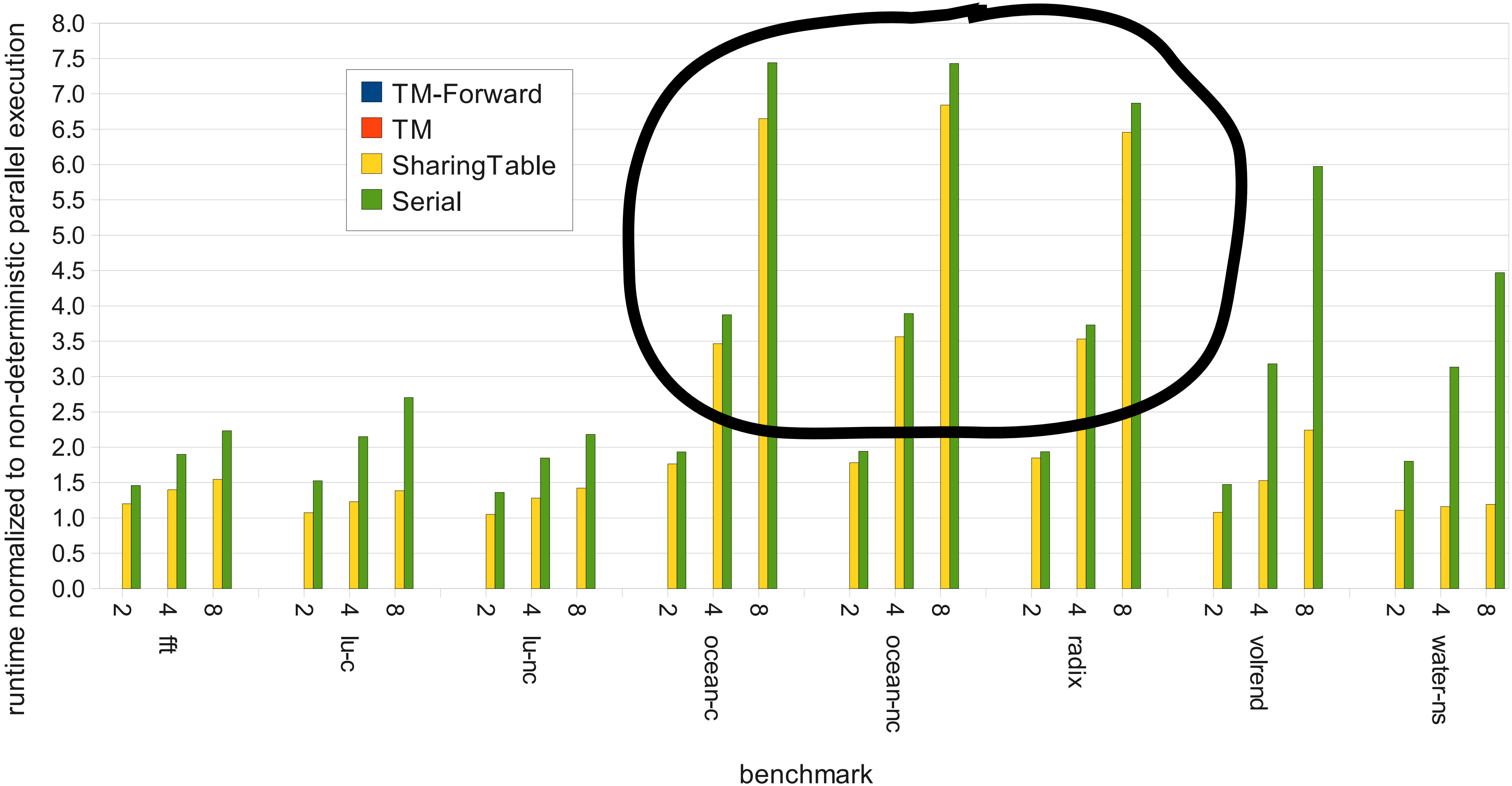
Runtime Overhead



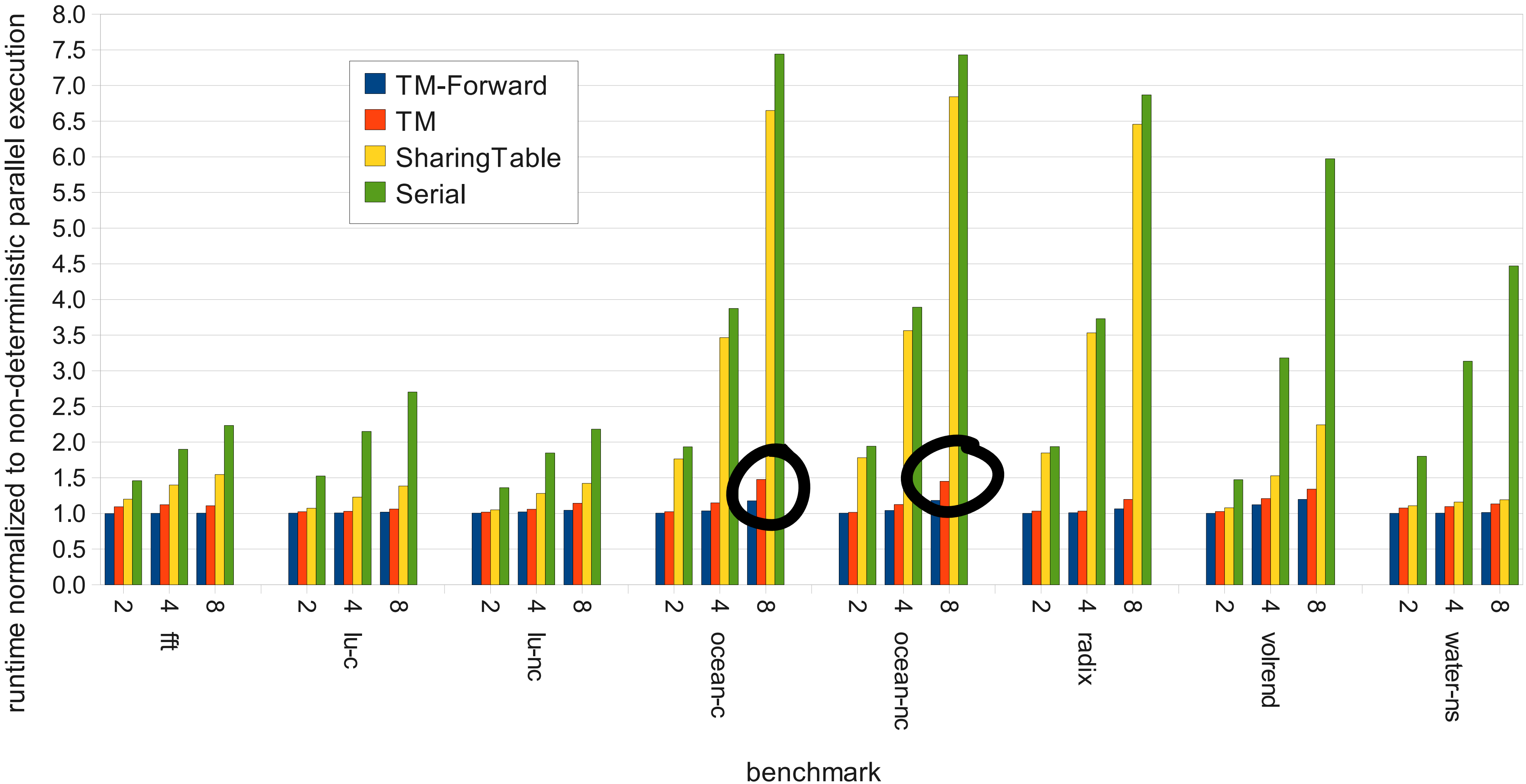
Runtime Overhead



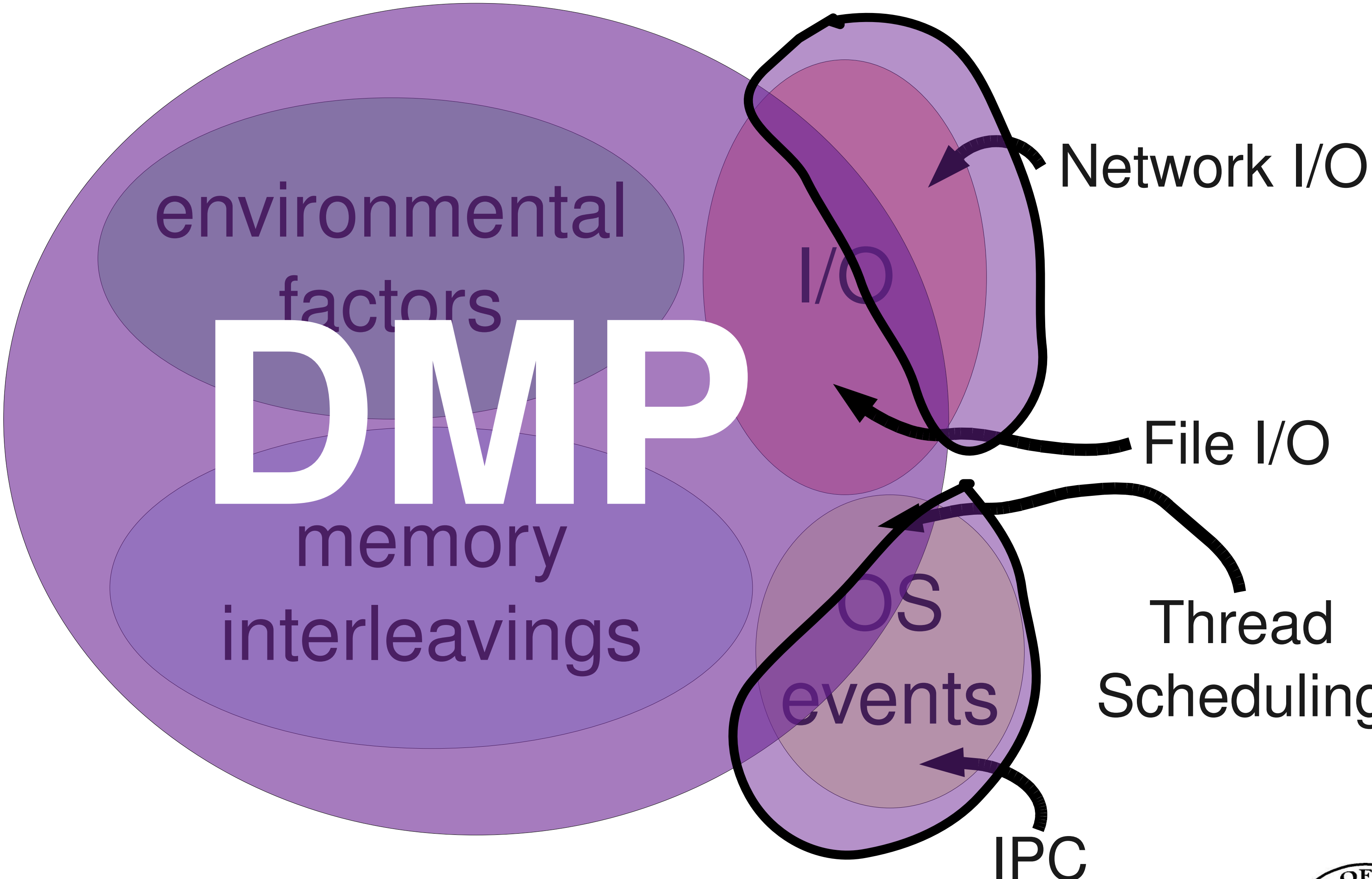
Runtime Overhead



Runtime Overhead



Non-deterministic events



Conclusions

- Determinism is a Good Thing
  - Simplifies debugging, testing and (potentially) deployment of parallel programs
  - We want sequential behavior with parallel performance
- We show several ways to build efficient DMPs
  - No memory log
  - Competitive performance

Questions?