Time travelling in multicore processors

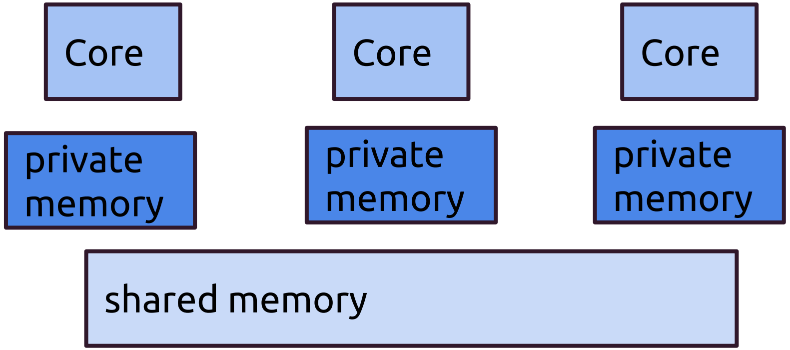
Henry Liu and Ethan Zou

Outline

1. **Background on multicore/distributed systems**
2. TARDIS Protocol
3. Optimizations and Evaluations
   a. Delta Timestamps
   b. Various Lease Predictor Protocols
4. Future Work and Acknowledgements

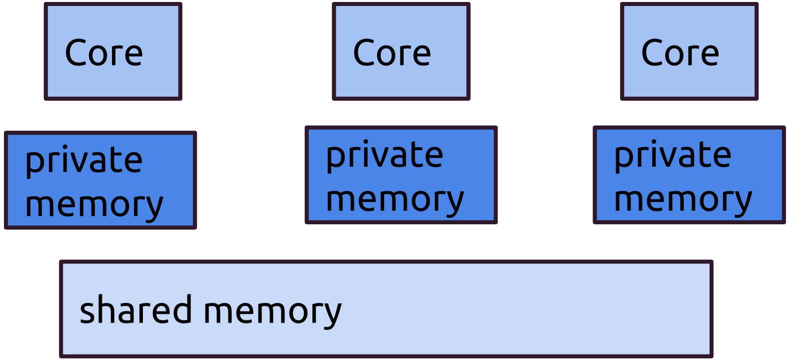
Background

- cores equivalent to processors
- faster performance → multiple cores
- data is shared by different cores, we need shared memory



Coherence

- If one processor modifies the data, how can other processors know the latest value?
- having stale data and writing stale data results in error and **incoherence**



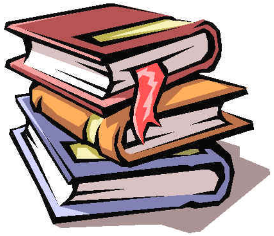
Outline

1. Background on multicore/distributed systems
2. **TARDIS Protocol**
3. Optimizations and Evaluations
   a. Delta Timestamps
   b. Various Lease Predictor Protocols
4. Future Work and Acknowledgements

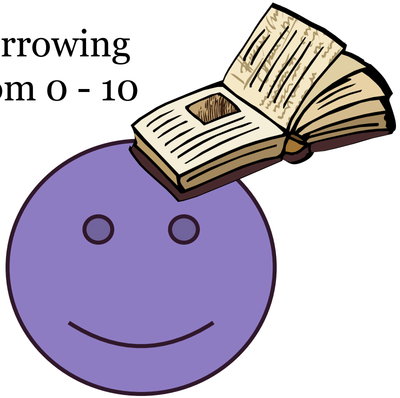
Tardis

- Recently proposed protocol
- Very scalable and simple
- Uses timestamps to logically organize shared memory and ensure coherence
- Allows for "time traveling" of operations since they don't have to be done in sequence of physical time

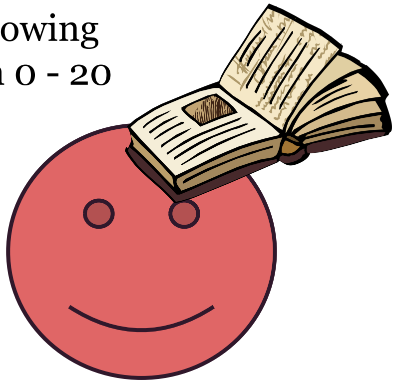
Library Example



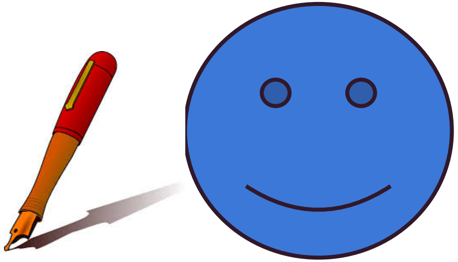
Borrowing
from 0 - 10



Borrowing
from 0 - 20



Wants to edit, so jumps
in time and edits at 21



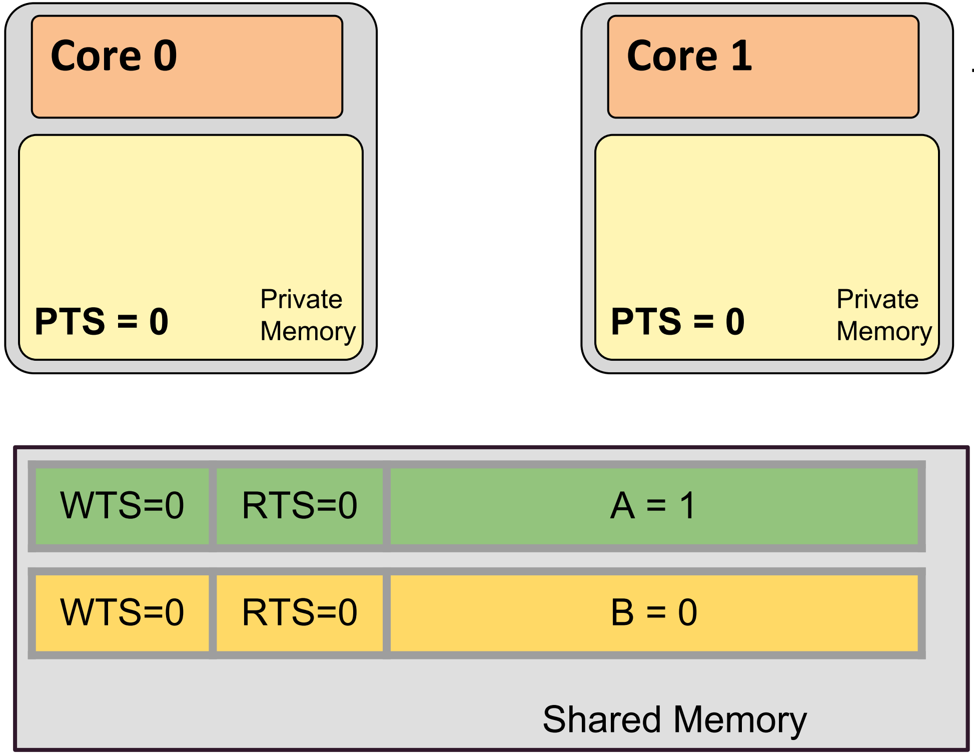
TARDIS Protocol

- Each cacheline in Tardis has a Read TimeStamp (RTS) and a Write TimeStamp (WTS)
- WTS - time of last store
- RTS - time of last read
- Private memory - data loaded at timestamp before rts
- Shared memory - rts is the longest private memory lease
- Cacheline Structure:

WTS	RTS	Data

TARDIS Example

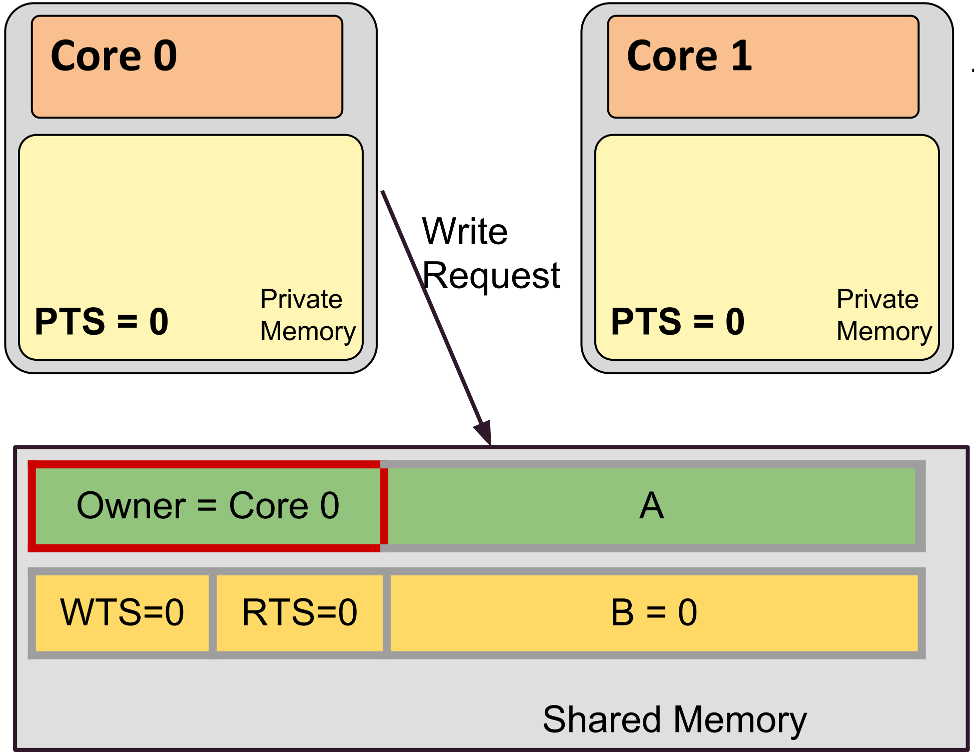
Tasks:
`Set A=2`
`Print B`



Tasks:
`Set B=3`
`Print A`

TARDIS Example

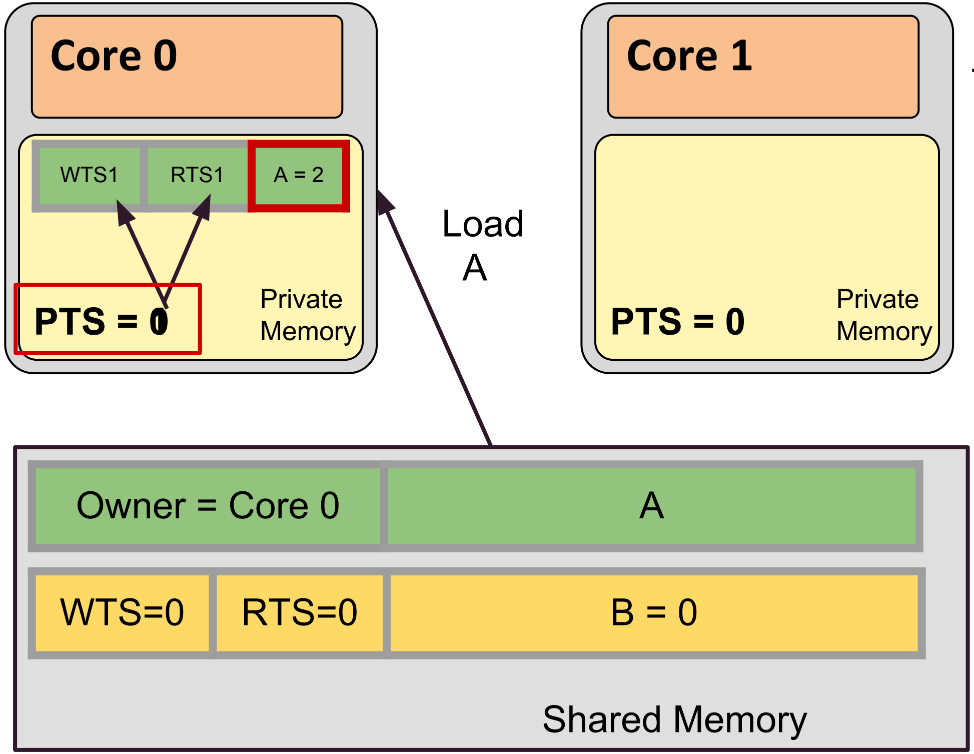
Tasks:
```
Set A=2
Print B
```



Tasks:
```
Set B=3
Print A
```

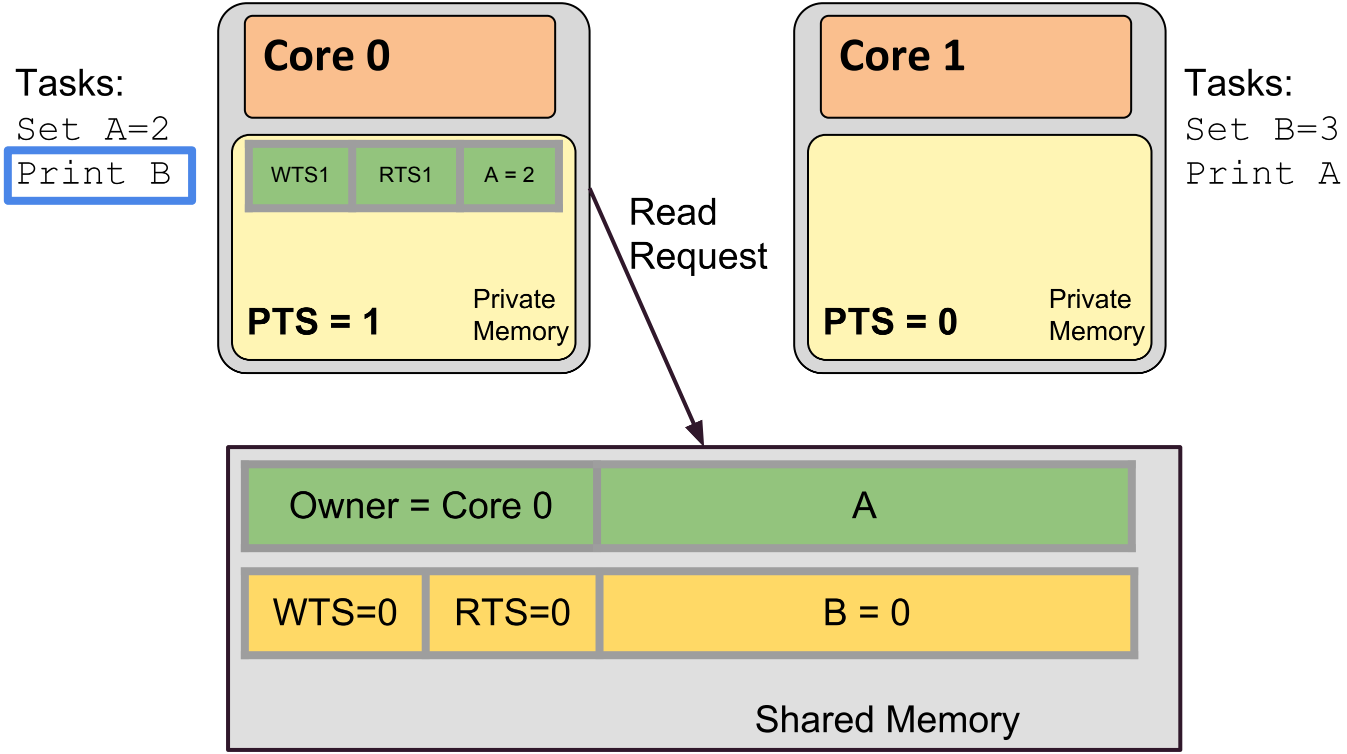
TARDIS Example

Tasks:
```
Set A=2
Print B
```



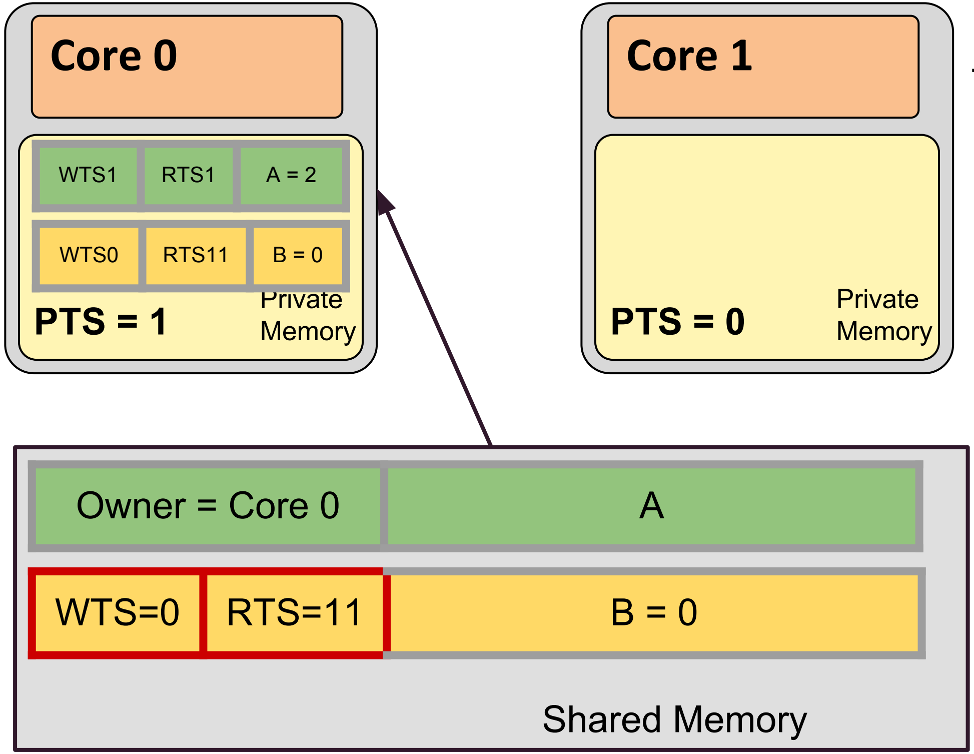
Tasks:
```
Set B=3
Print A
```

TARDIS Example



TARDIS Example

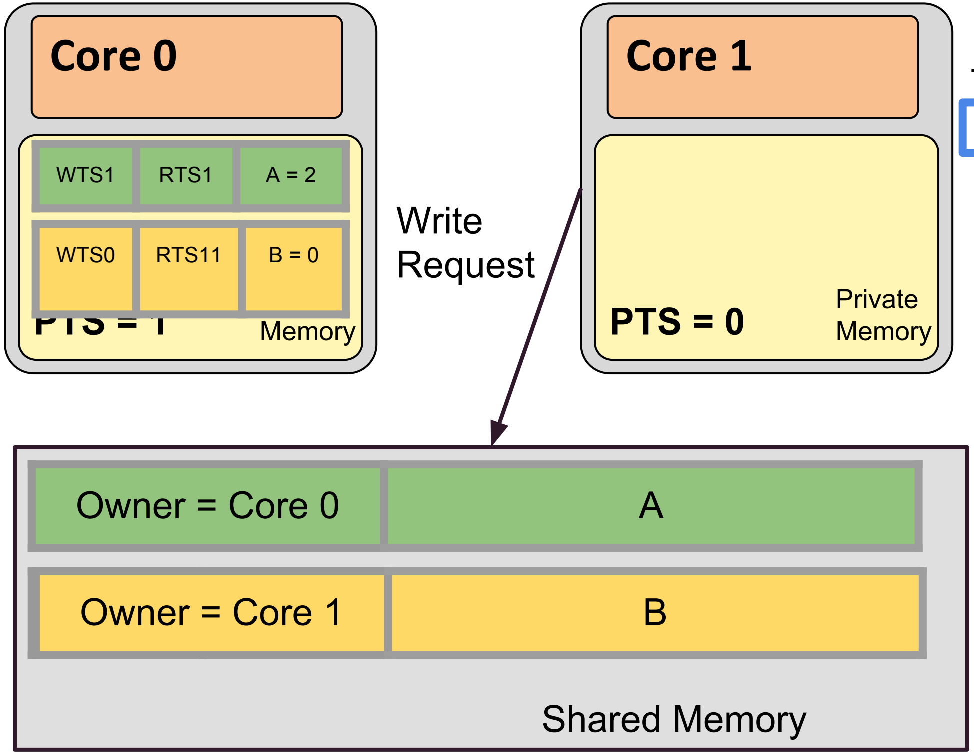
Tasks:
Set A=2
Print B



Tasks:
Set B=3
Print A

TARDIS Example

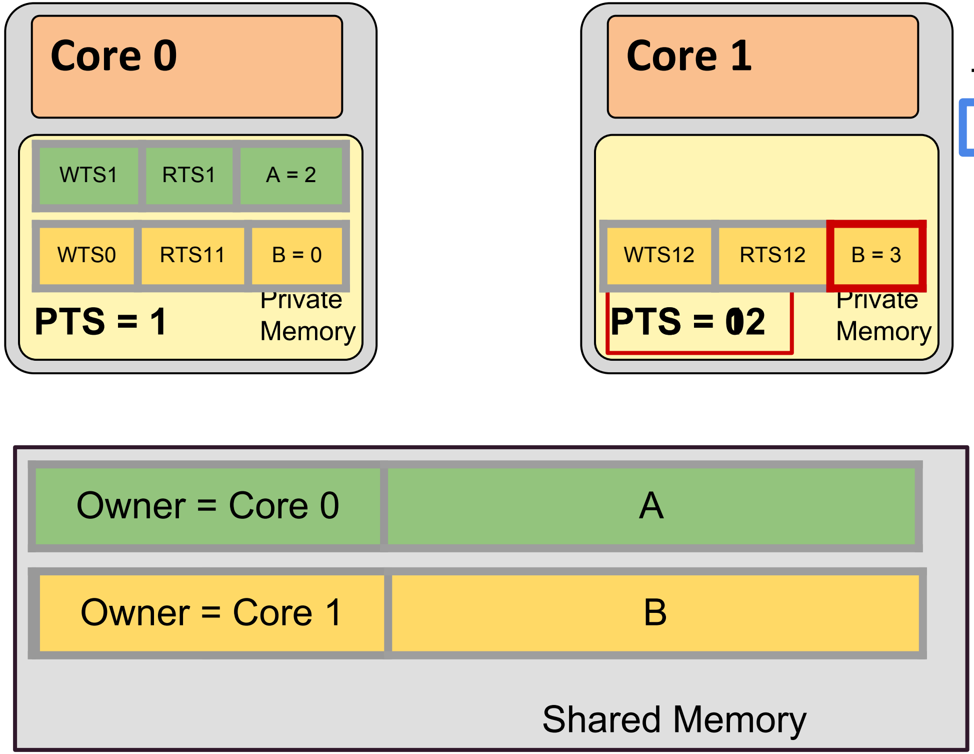
Tasks:
`Set A=2`
`Print B`



Tasks:
`Set B=3`
`Print A`

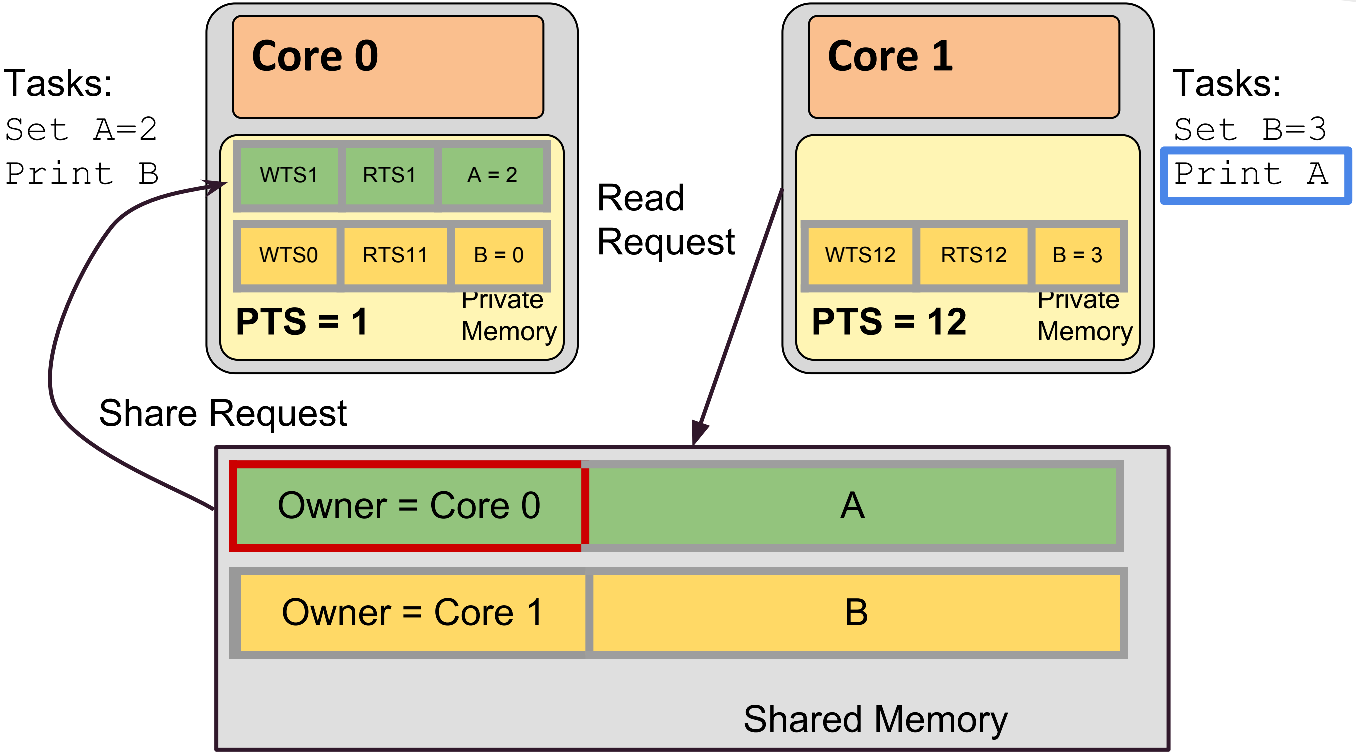
TARDIS Example

Tasks:
```
Set A=2
Print B
```



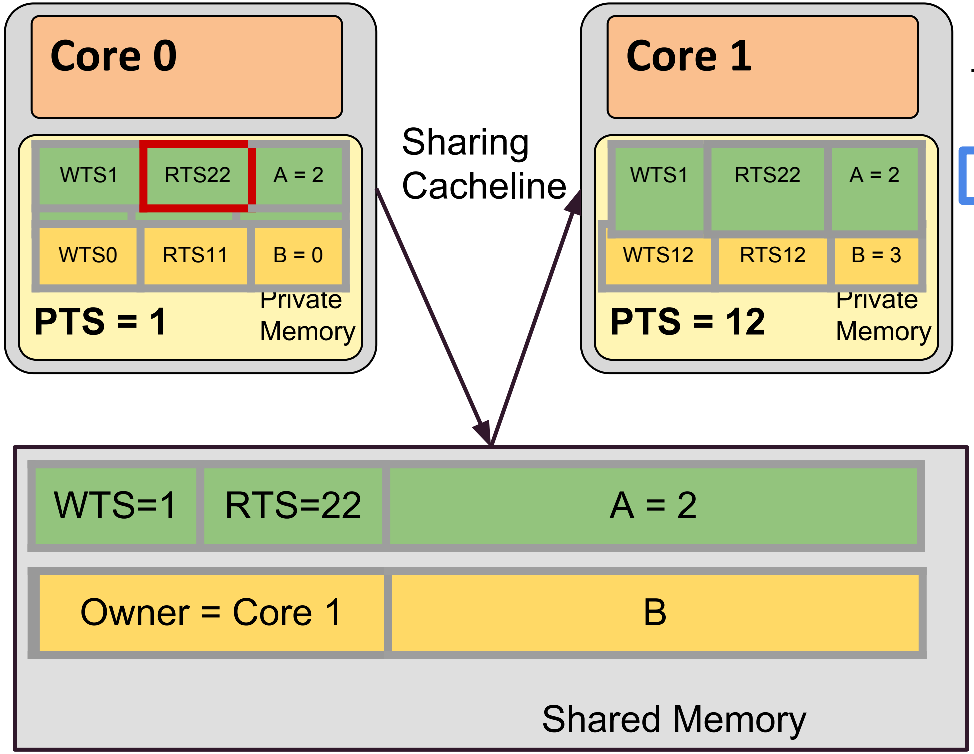
Tasks:
```
Set B=3
Print A
```

TARDIS Example



TARDIS Example

Tasks:
```
Set A=2
Print B
```



Tasks:
```
Set B=3
Print A
```

Done

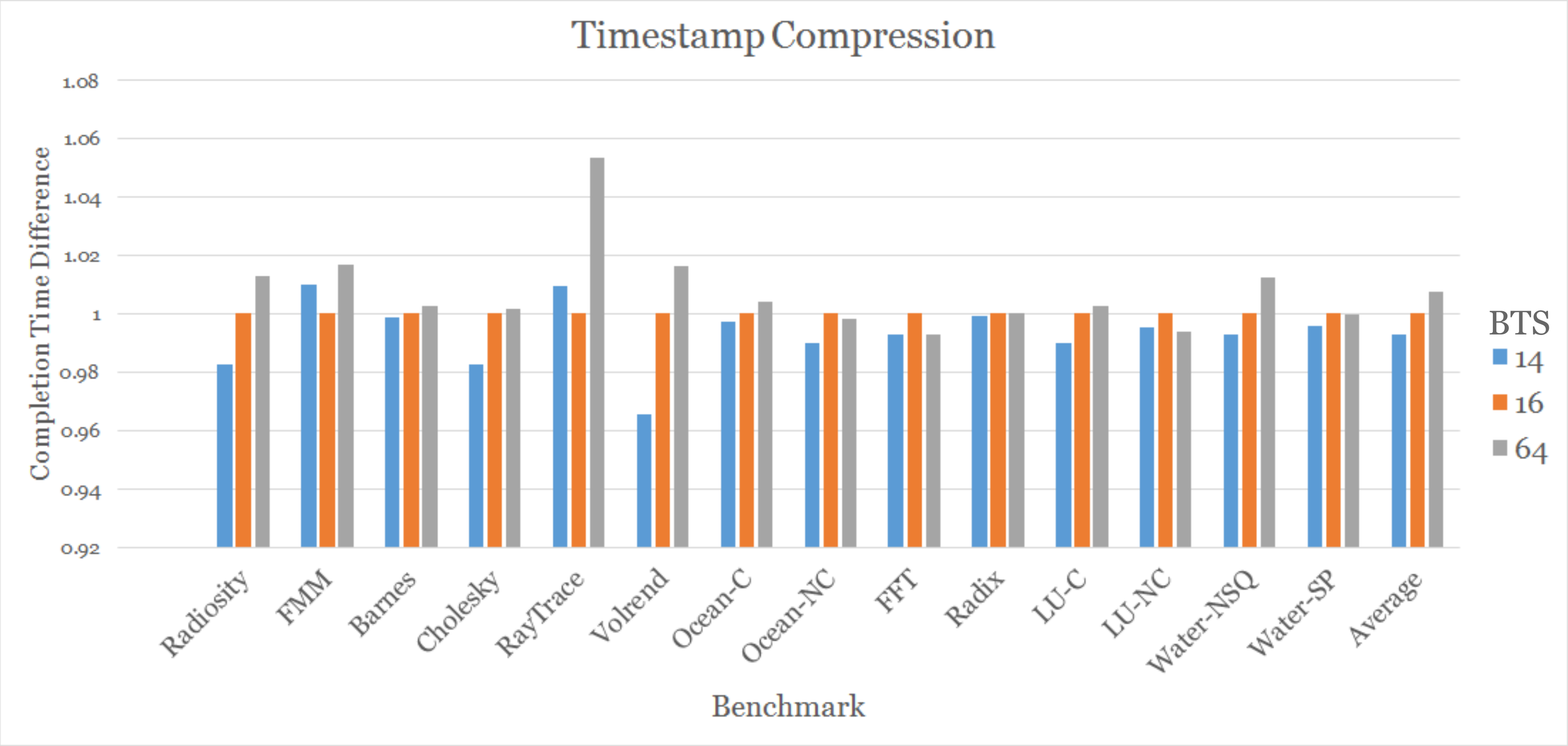
Outline

1. Background on multicore/distributed systems
2. TARDIS Protocol
3. **Optimizations and Evaluations**
   a. **Delta Timestamps**
   b. Various Lease Predictor Protocols
4. Future Work and Acknowledgements

Timestamp Compression

- Timestamp size should be small for space efficiency
- Data is 512 bits; timestamp originally 64 bits each (25% of data)
- Wts and rts are usually fairly close, so we use a base timestamp (bts) and a delta (difference) = rts-wts to represent rts and wts
- We then ran tests to determine the optimal bts
- Now 16 bits each (6.25% of data)

Timestamp Compression

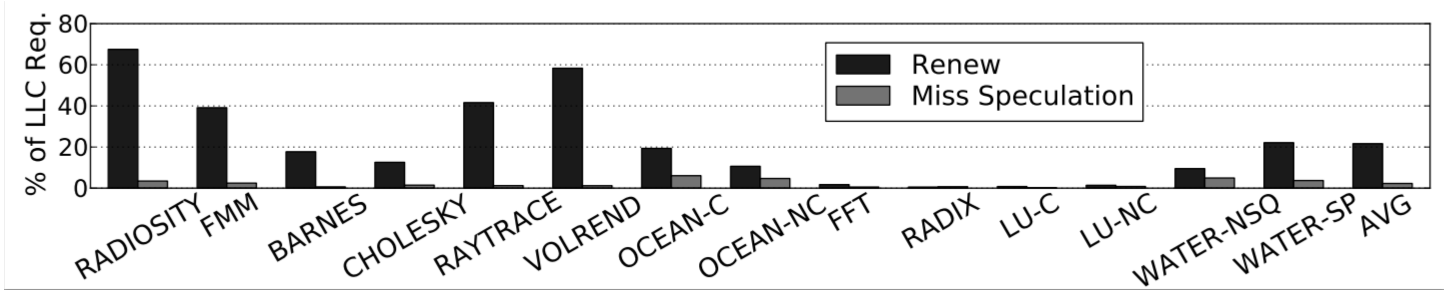


Outline

1. Background on multicore/distributed systems
2. TARDIS Protocol
3. **Optimizations and Evaluations**
   a. Delta Timestamps
   b. **Various Lease Predictor Protocols**
4. Future Work and Acknowledgements

The Renewal Problem

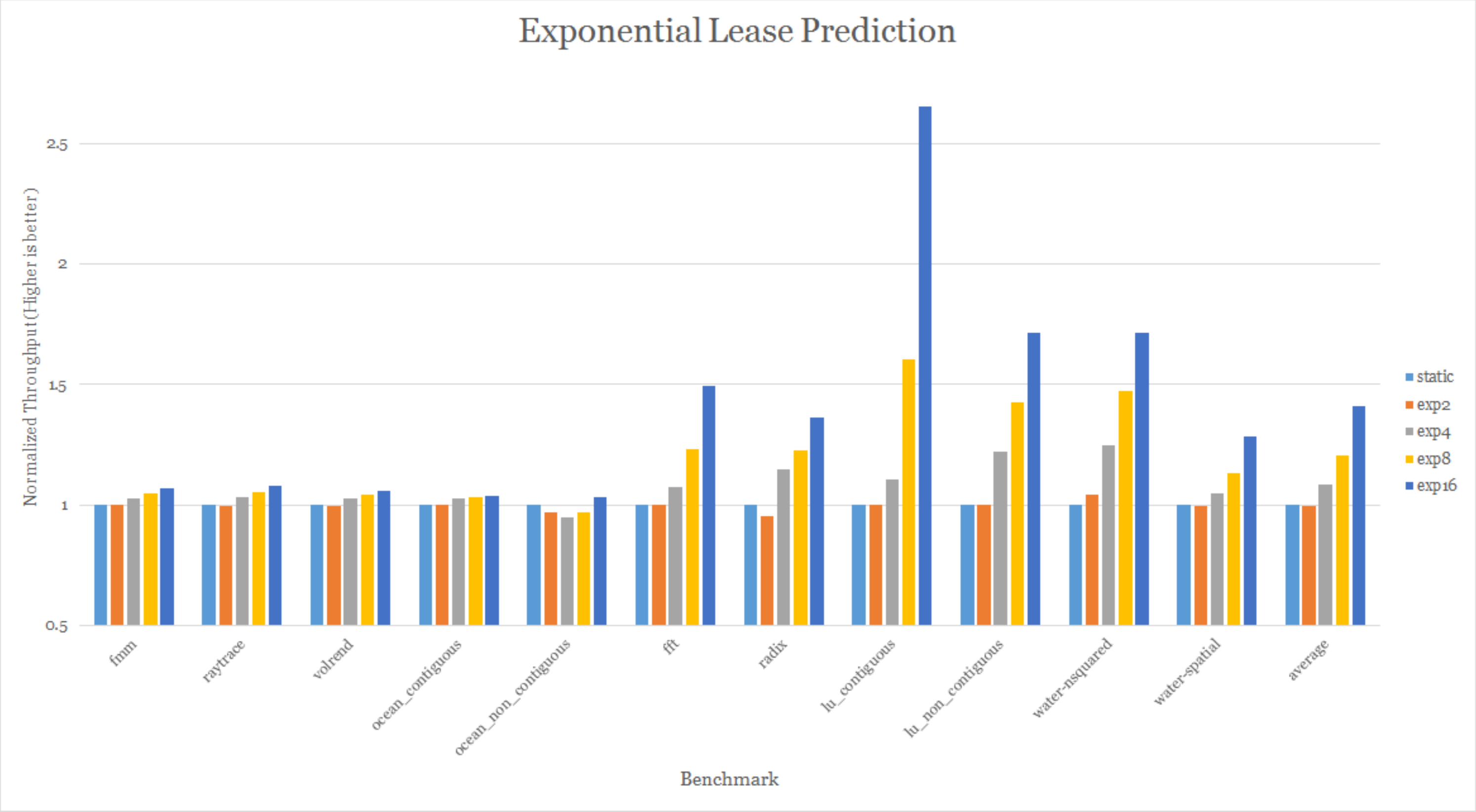
- if we keep modifying data, timestamps will increase by the arbitrary value of 10
  - read-write intensive, want the lease to be something much less than 10
- read-only data, we keep renewing it, lease can be very large
- renew requests incur extra latency and network traffic



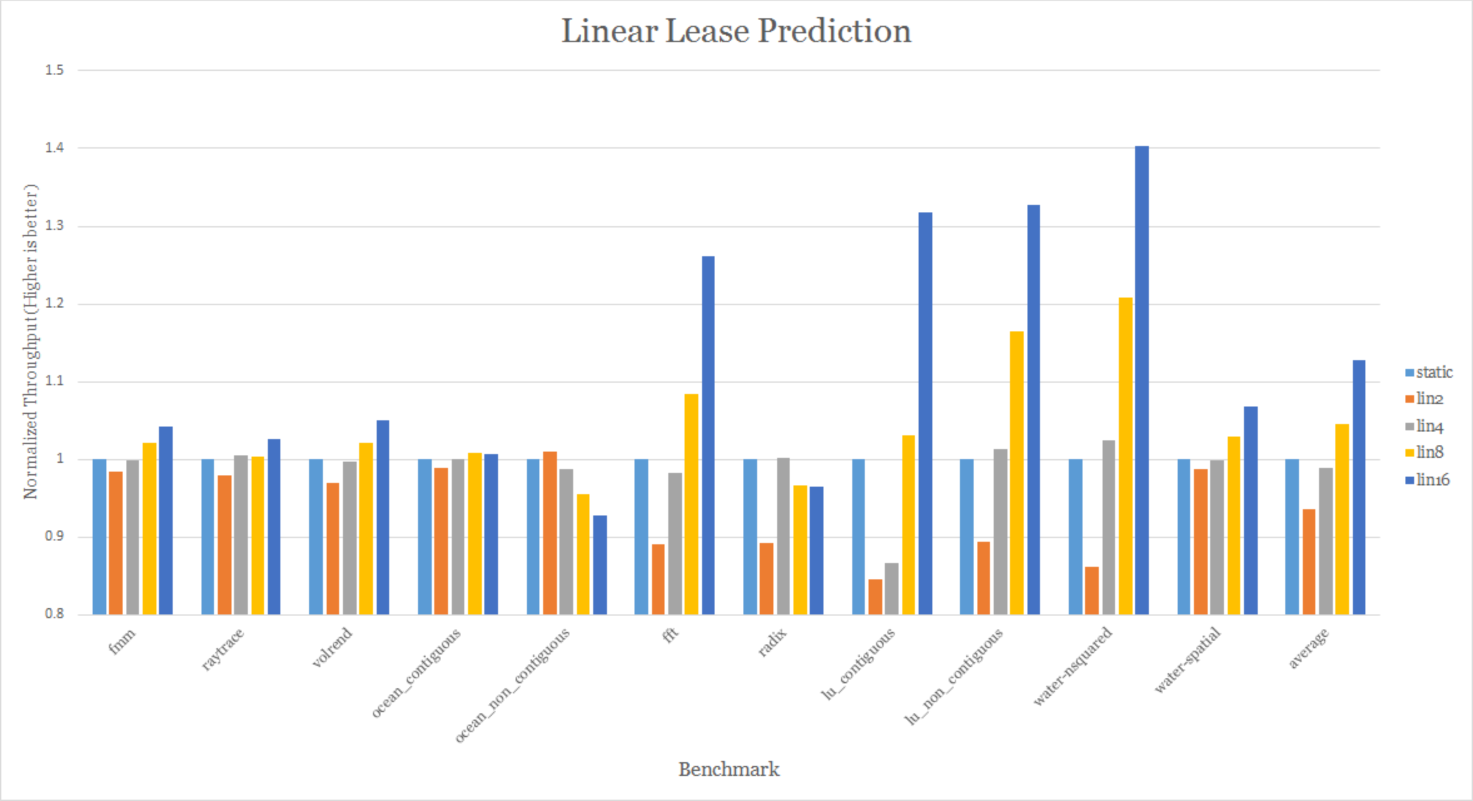
Minimizing Renewals

- an adaptively changing lease
  - lines that are written to frequently should have a small lease
  - lines written to less frequently/read-only should have longer lease
- two basic protocols
  - exponentially growing lease
  - linearly growing lease

Evaluations of Lease Protocols



Evaluations of Lease Protocols



Outline

1. Background on multicore/distributed systems
2. TARDIS Protocol
3. Optimizations and Evaluations
    a. Delta Timestamps
    b. Various Lease Predictor Protocols
4. **Future Work and Acknowledgements**

Future Work

- better lease prediction algorithm
- Renew in batches
- Renew in the background
- Techniques to slow down timestamp increment
- Further timestamp compression

Acknowledgements

We would like to thank:

- our parents
- our mentor, Xiangyao Yu and Professor Srini Devadas
- the MIT PRIMES program