Michael Rüfenacht – JMCS Workshop 2013

INTRA-ACTOR PARALLELISM IN THE ACTOR MODEL

Roadmap

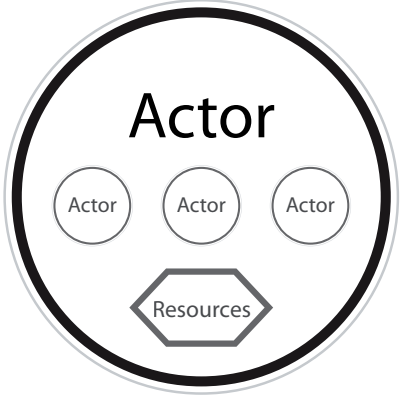
1. **The Actor Model**
2. **Parallel Actor Monitors**
3. **The Unified Model**
4. **Comparison & Conclusion**
5. **Reflection**

1. The Actor Model (Recap)

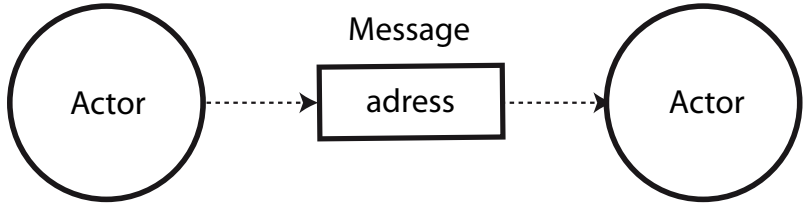
- **Everything is an actor.**
- **An actor can ...**
  - **send/receive** a finite amount of **messages**
  - **spawn** new actors
  - **change** ist internal **state**
- but only process **one message at a time.**

1. The Actor Model (Recap)

Strong Encapsulation



Asynchronous Messaging



Safety & Liveness

No race conditions, data races, deadlocks ....

Particular Order of Events

no guaranteed order of message delivery

1. The Actor Model (Recap)

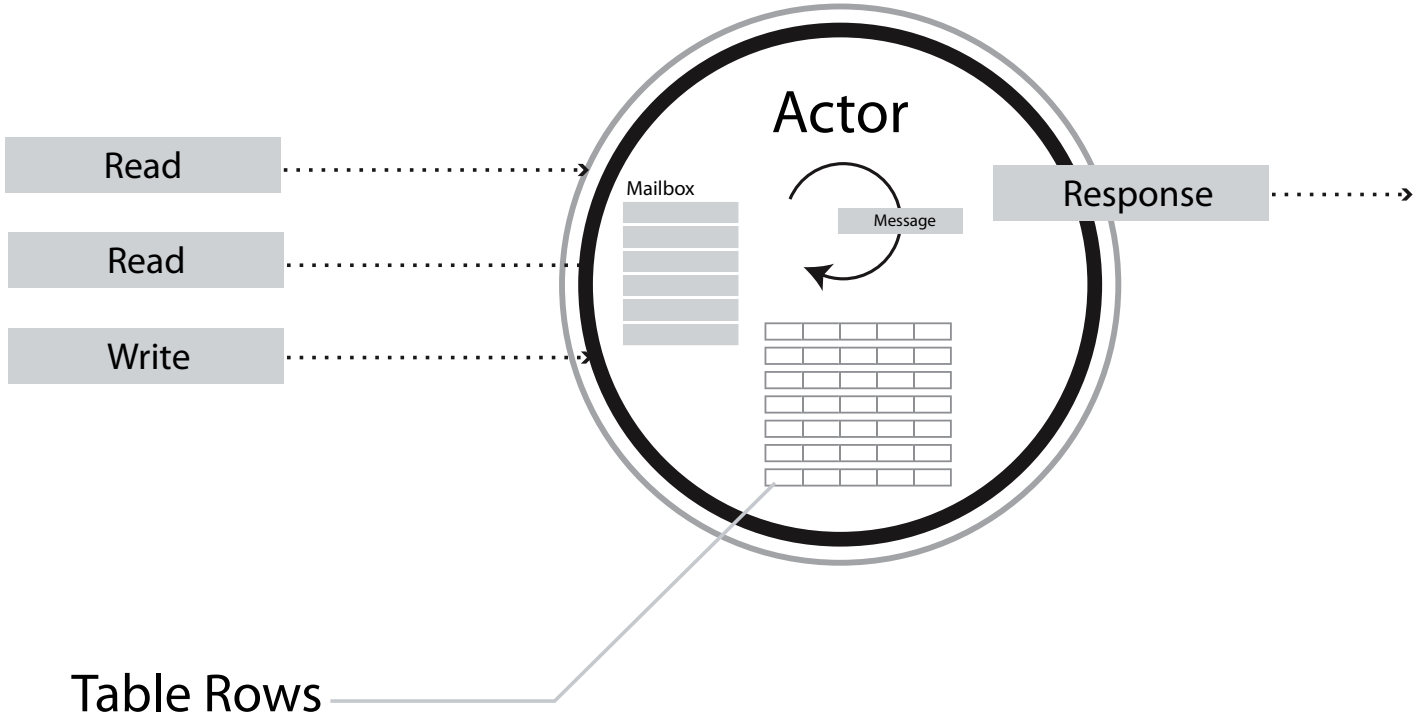
Asynchronously send messages

\+ no guaranteed order

\- no data races, deadlocks or similar

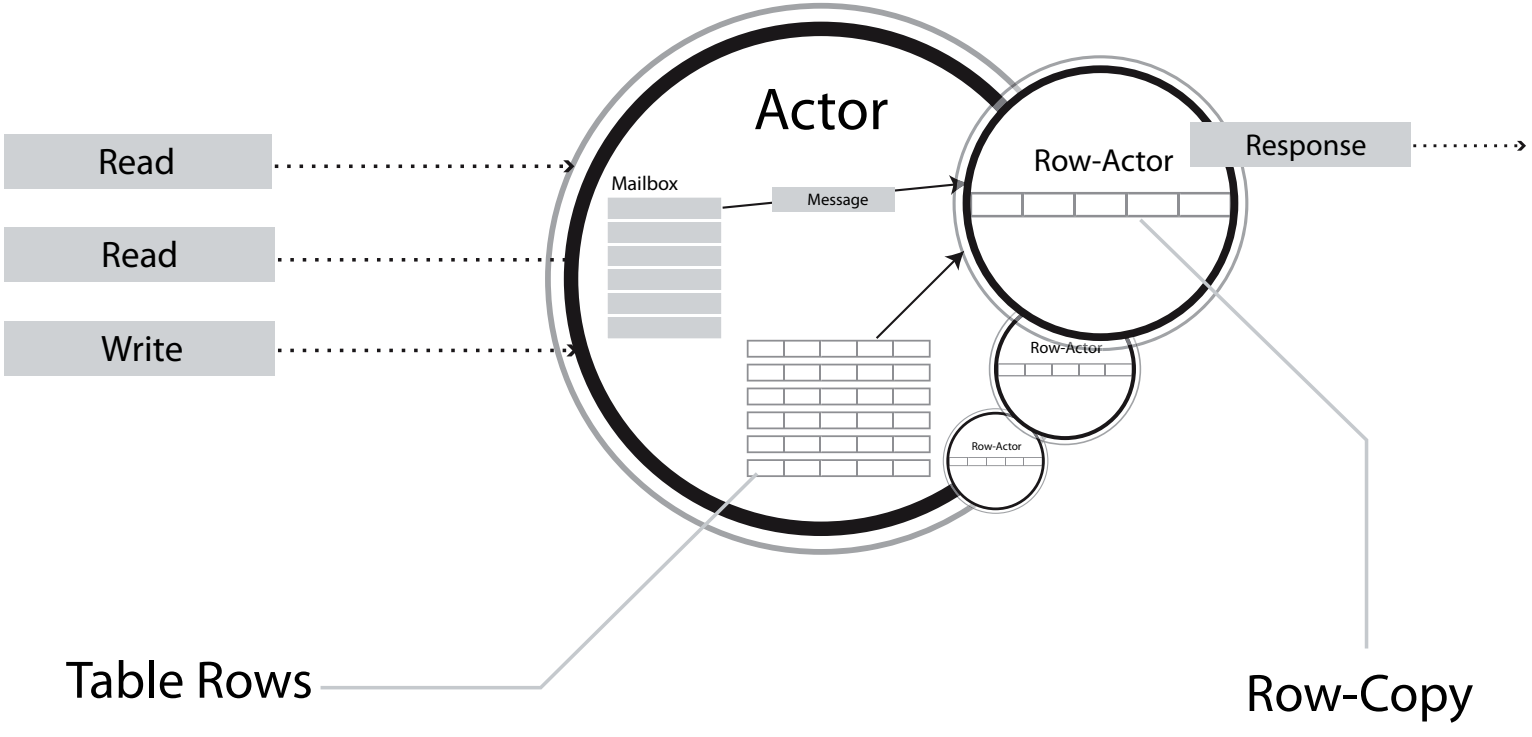
= inherently concurrent

1. The Actor Model (Recap)



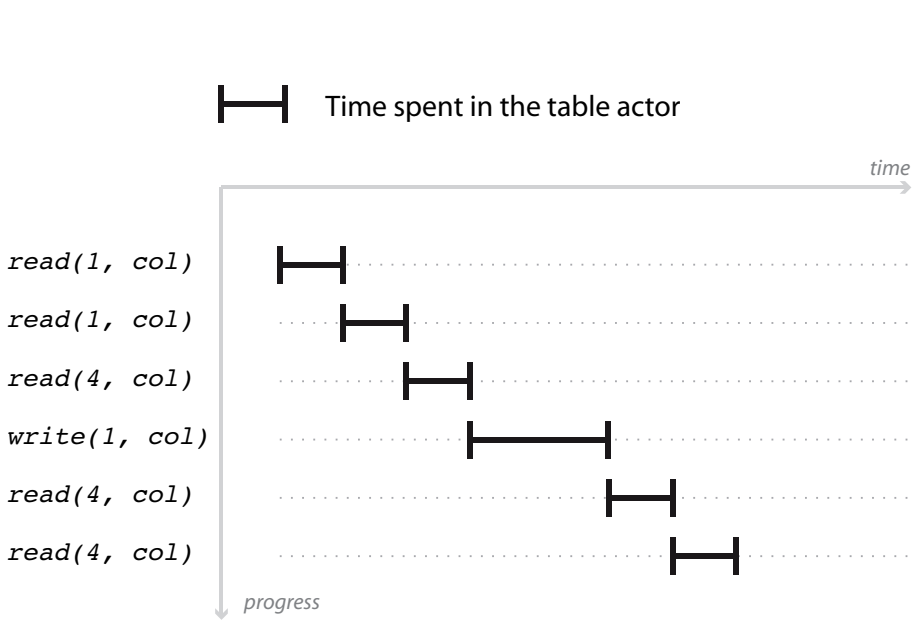
Parallel Message Processing!

1. Data Partitioning

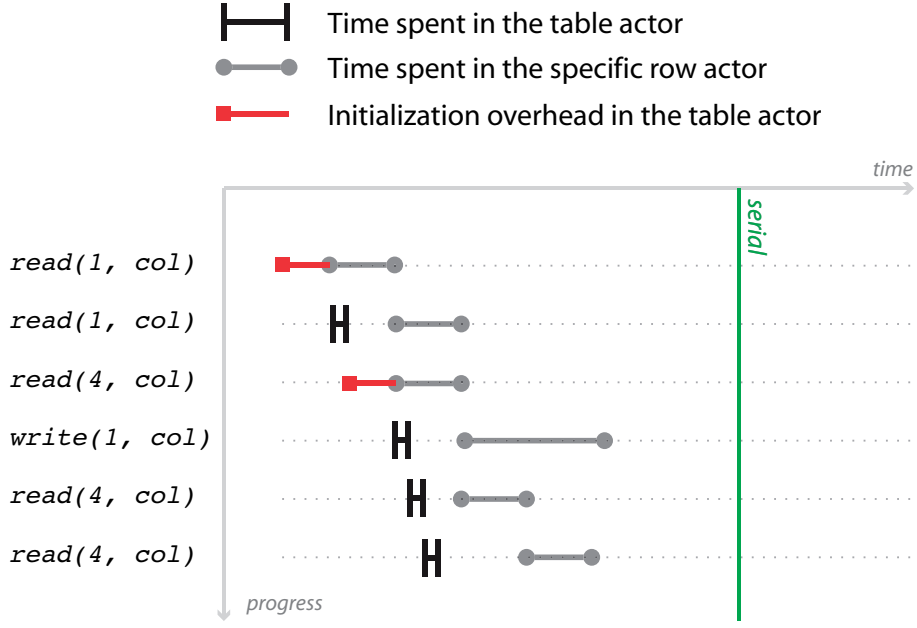


1. The Actor Model

Serial Processing



Data Partitioning

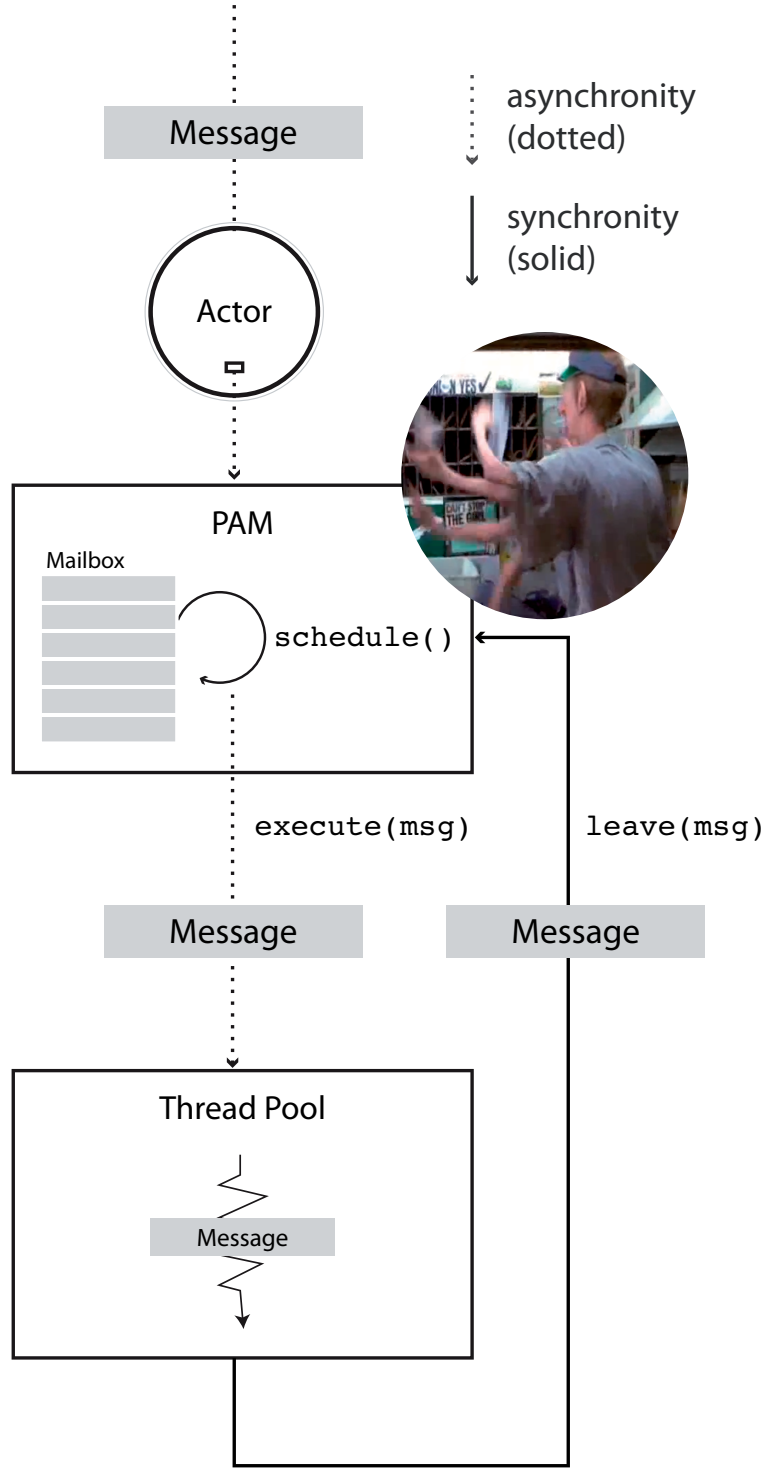


Disentangling task & data coupling

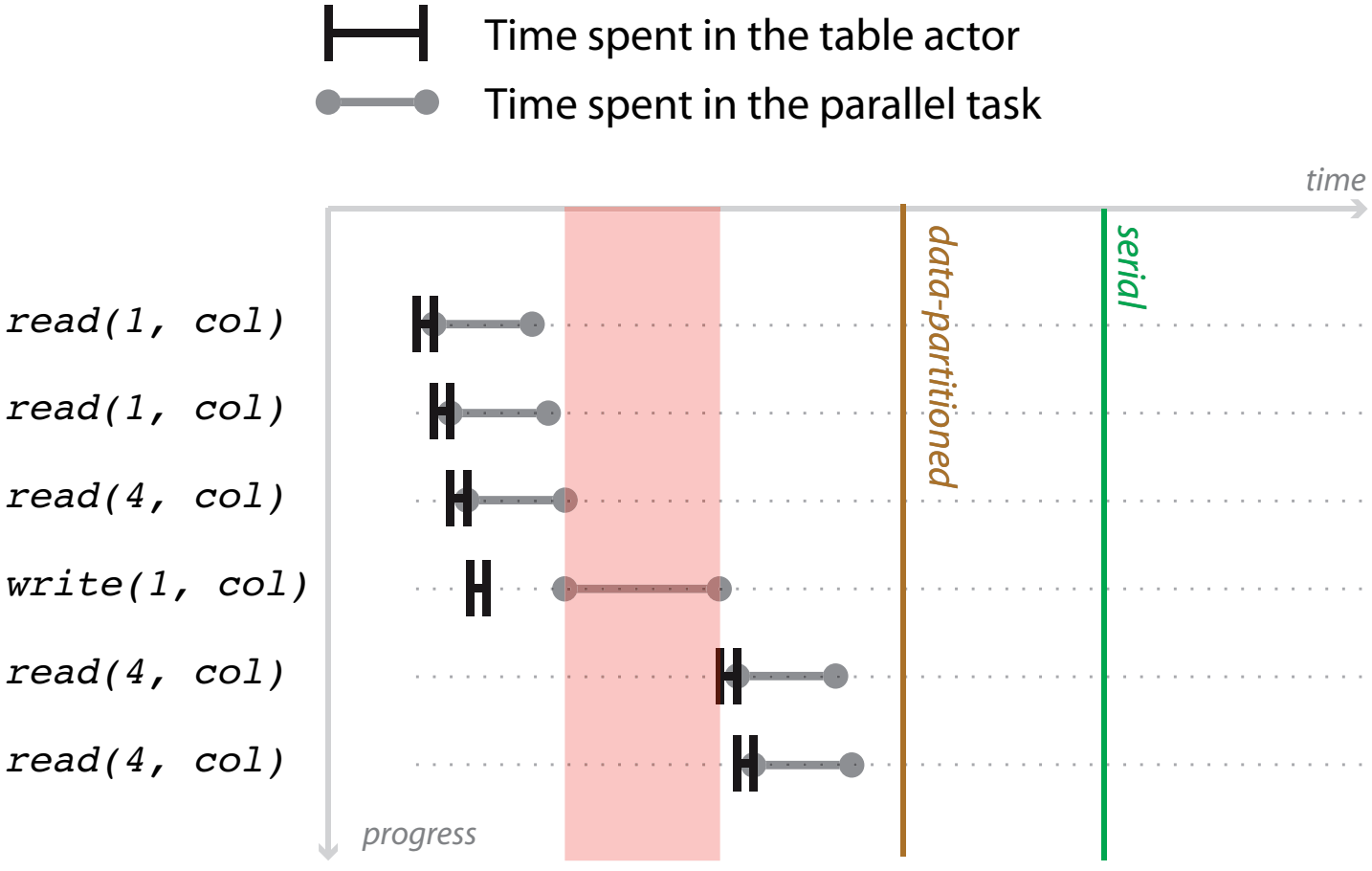
2: PARALLEL ACTOR MONITORS (PAM)

2. Parallel Actor Monitors

- **Parallel Actor Monitors**
  - are **schedulers**
  - implement an **actor-specific scheduling policy**
  - PAM got access to **a thread pool**
  - scheduled messages get **executed in threads**
  - the code has **free access to the actor**‘s state


Message
asynchronity
(dotted)
synchronity
(solid)
Actor
PAM
Mailbox
schedule()
execute(msg)
leave(msg)
Message
Message
Thread Pool
Message

2. Parallel Actor Monitors



AFM + AM = UM

3: THE UNIFIED MODEL

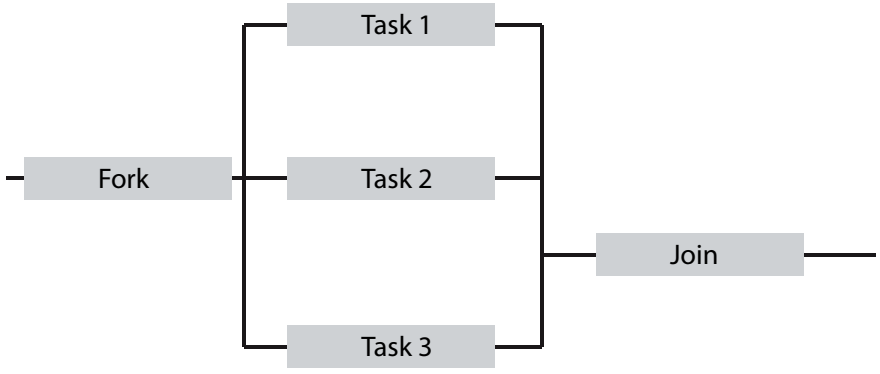
3. The Unified Model

- **The Unified Model**
  - combines the AFM,
  - the Actor Model
  - and Data Driven futures

3. Async-Finish Model (AFM)

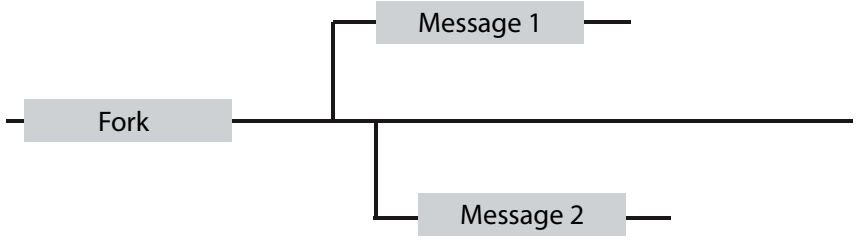
Message

```
finish {
   async {
      Task 1
   }
   async {
      Task 2
   }
   async {
      Task 3
   }
}
```



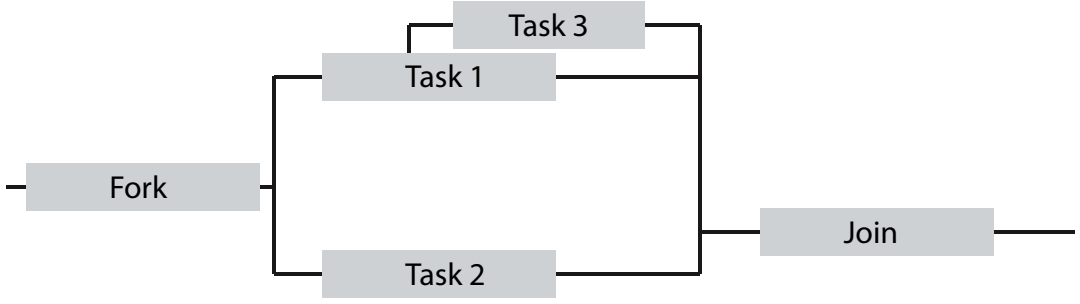
Escaping!

```
async {
   Message 1
}
async {
   Message 2
}
```



3. Data-Driven Futures

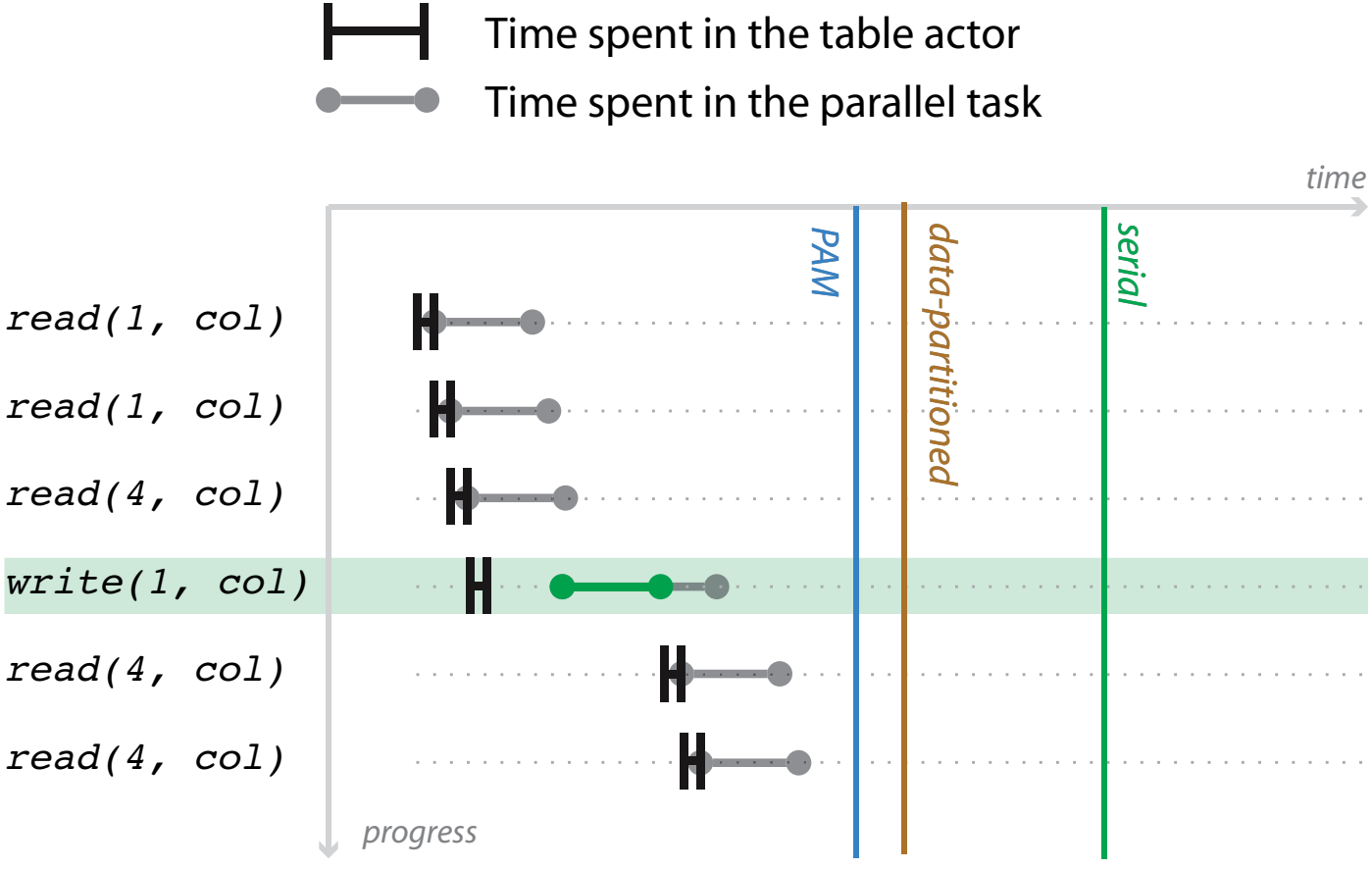
```
finish {

   value future = ddf()

   async {
      Task 1 compute the value of the future
      future.put(result)
   }
   async {
      Task 2
   }
   asyncAwait( future ) {
      Task 3
   }
}
```



3. The Unified Model

- **AFM (fork-join)**
  - Simple creation and coordination of tasks
  - Intra-task (message) parallelsim (nesting)
- **Escaping**
  - Parallel message processing
- **DDF**
  - Coordination
- **Actor Model**

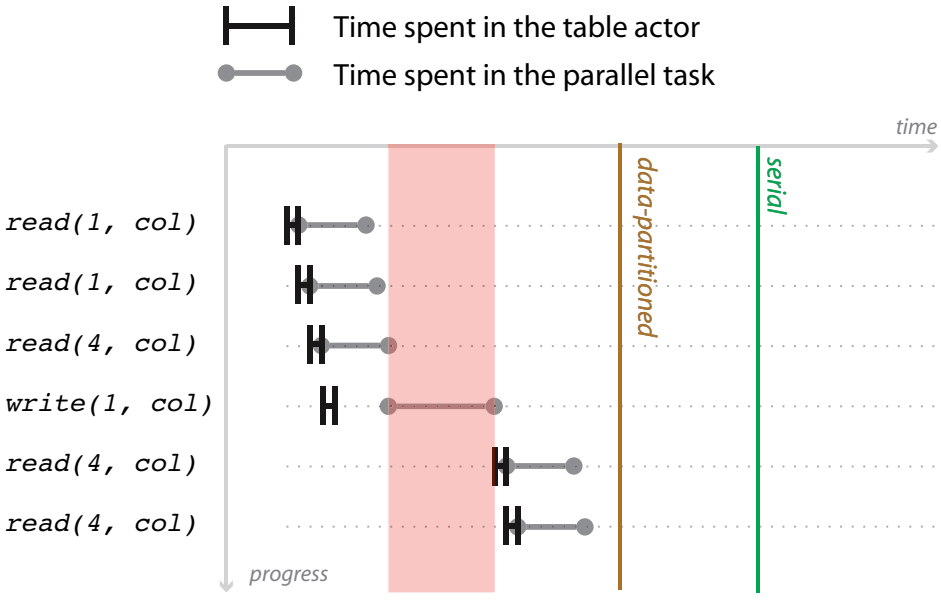
3. Unified Model

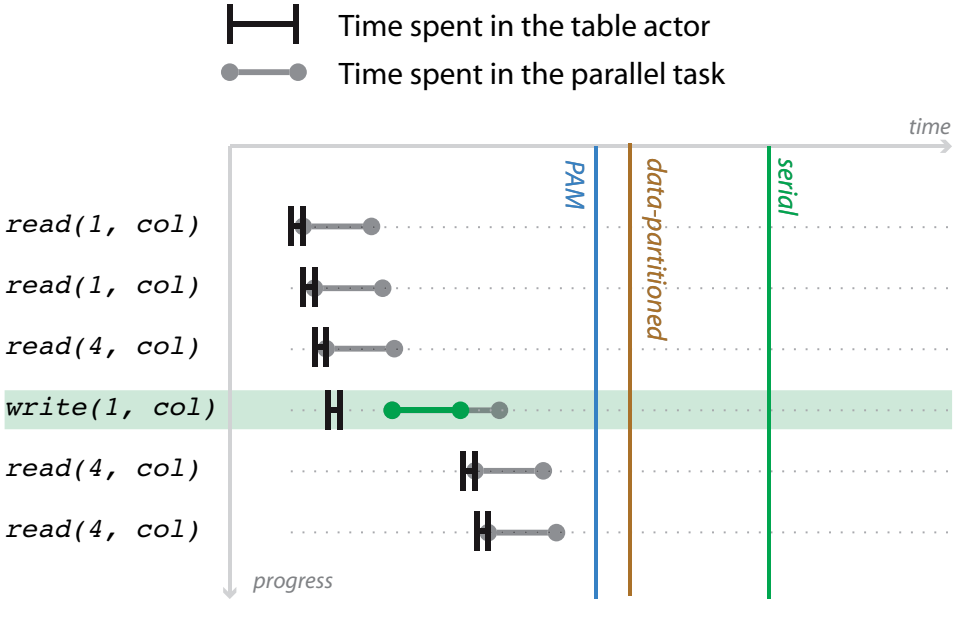


Task Parallelism vs. Intra-Task Parallelism

4: COMPARISON & CONCLUSION

4. Comparison & Conclusion





4. Comparison & Conclusion

- Both approaches **exploit parallelism**
- but introduce **unmonitored data races.**

New Paradigm & Actuality

5: REFLECTION

5. Reflection

- Gained **experience in writing**
- doing **research**
- had **issues finding other** approaches
- just **surveyed the provided materials**

All clear?

QUESTIONS