Transactional Actors
Communication in Transactions

Janwillem Swalens
Joeri De Koster
Wolfgang De Meuter

Software
Languages.Lab

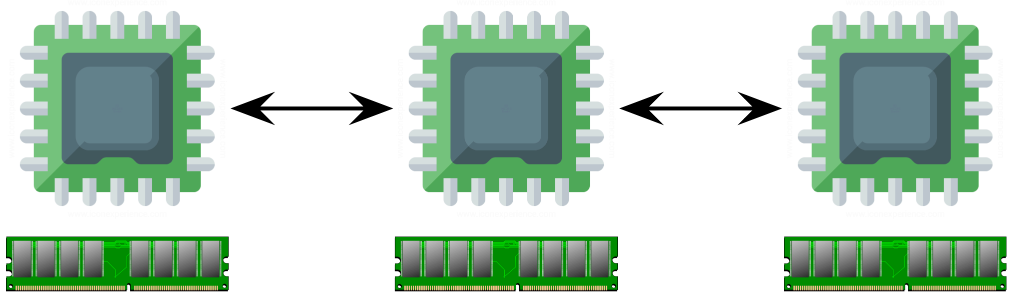
VUB
VRIJE
UNIVERSITEIT
BRUSSEL

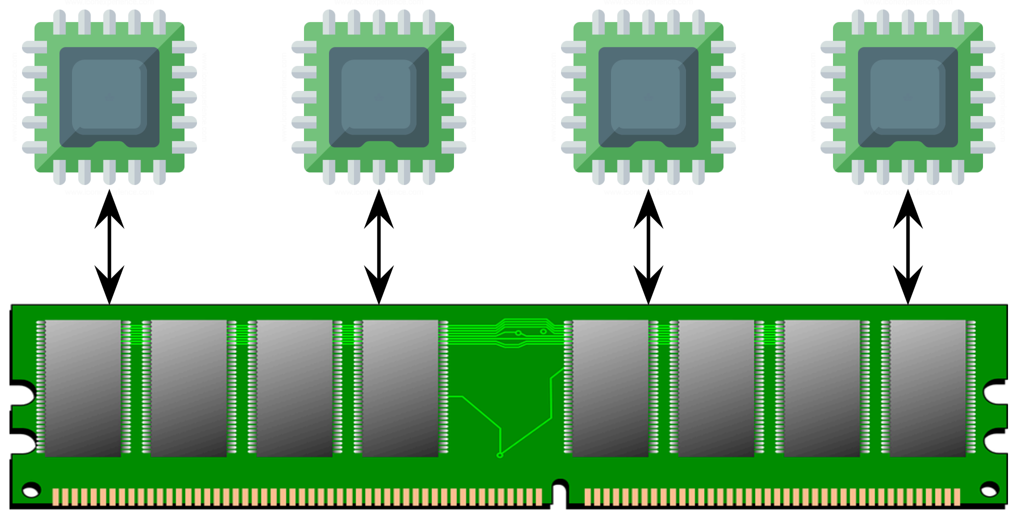
There are many concurrency models

active objects Nested Data Parallelism
Software Transactional Memory
futures dataflow actors locks threads MPI OpenMP Fork/Join MapReduce
transactional events worlds
Concurrent Revisions promises
Communicating Sequential Processes speculative parallelism

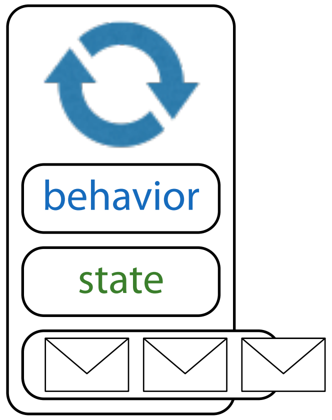
Two categories:

Message passing ↔ Shared memory



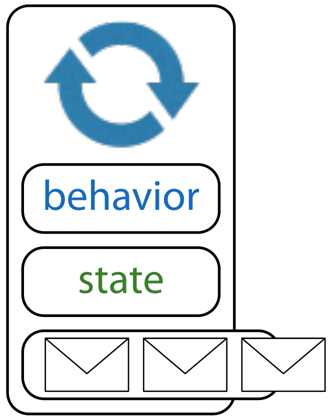


Actors



```
(def airline-behavior
  (behavior [flights]
    [orig dest n]
    (let [flight   (search-flight flights orig dest)
          flight'  (reserve flight n)
          flights' (replace flights flight flight')]
      (become airline-behavior flights'))))

(def air-canada
  (spawn airline-behavior
    {"AC854" {:orig "YVR" :dest "LHR" :seats 211}
     "AC855" {:orig "LHR" :dest "YVR" :seats 211}}))

(send air-canada "LHR" "YVR" 2)
```

Actors



```
(def airline-behavior
  (behavior [flights]
    [orig dest n]
    (let [flight   (search-flight flights orig dest)
          flight'  (reserve flight n)
          flights' (replace flights flight flight')]
      (become airline-behavior flights'))))
```

turn

```
(def air-canada
  (spawn airline-behavior
    {"AC854" {:orig "YVR" :dest "LHR" :seats 211}
     "AC855" {:orig "LHR" :dest "YVR" :seats 211}}))

(send air-canada "LHR" "YVR" 2)
```

no low-level data races

no deadlocks

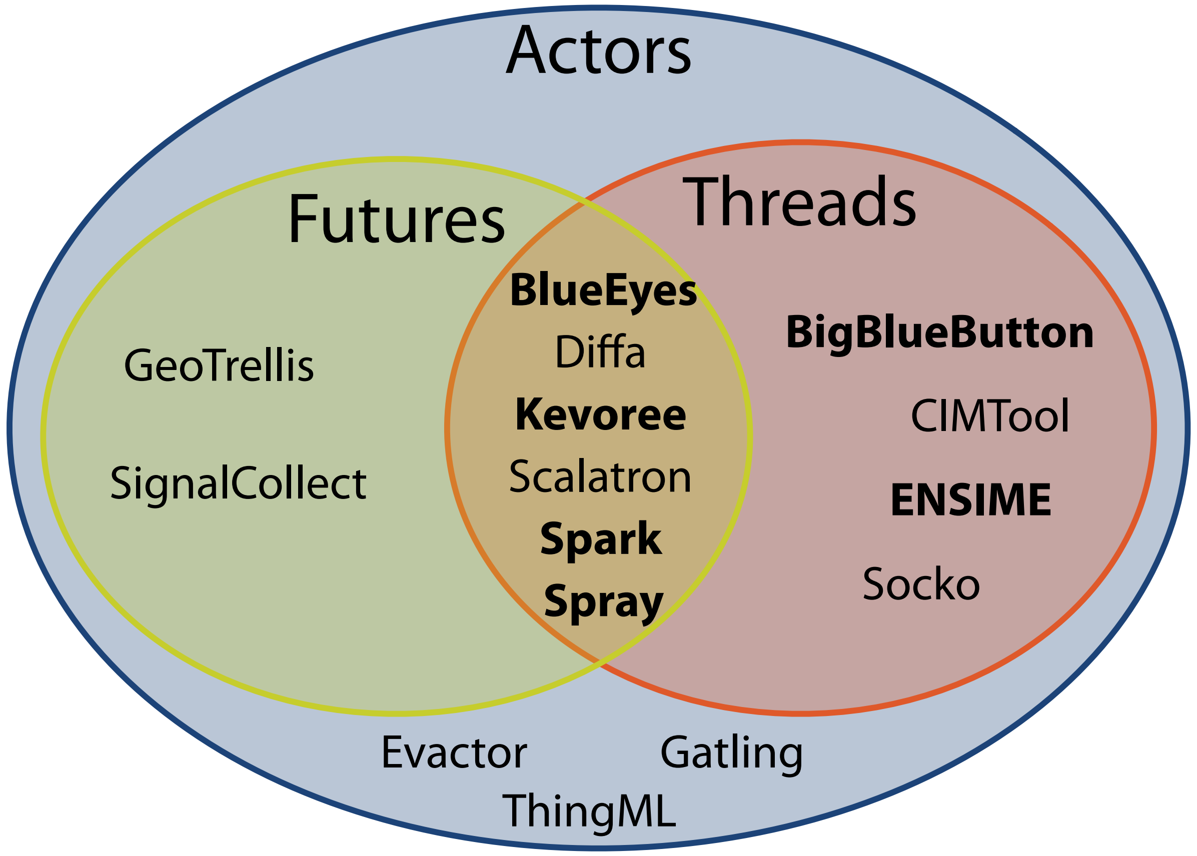
Software Transactional Memory

```
(def flights
  {"AC854" {:orig "YVR" :dest "LHR" :seats (ref 211)}
   "AC855" {:orig "LHR" :dest "YVR" :seats (ref 211)}…})

(dosync
  (let [outbound (get (get flights "AC854") :seats)
        return   (get (get flights "AC855") :seats)]
    (if (and (>= @outbound 2) (>= @return 2))
      (do (ref-set outbound (- @outbound 2))
          (ref-set return   (- @return   2)))
      (println "Not enough seats available"))))
```

trans-
action

serializability

Actors often share memory



Study of 15 Scala programs that use actors:

- 12/15 (80%) combine with another model
- 6/15 (40%) say they circumvent it where it is "not a good fit"

data races and deadlocks possible

Tasharofi, Dinges, and Johnson (2013). *Why Do Scala Developers Mix the Actor Model with Other Concurrency Models?* (ECOOP'13)

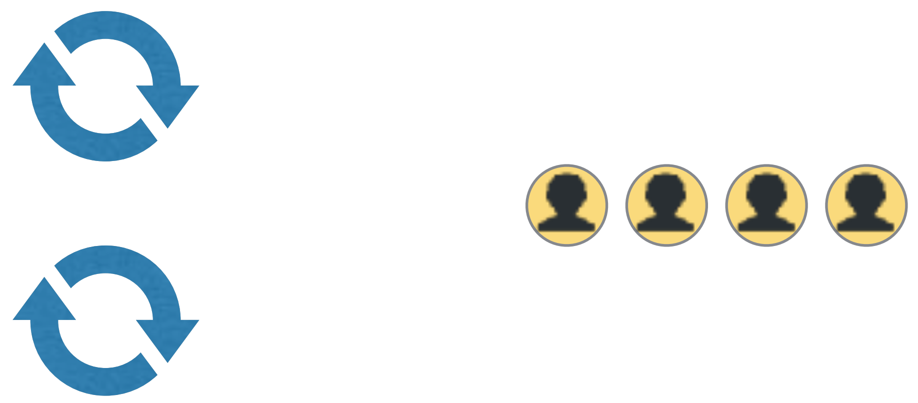
Vacation benchmark

```
(def flights   [(ref {:id "AC855"
                      :price 499
                      :orig "London" :dest "Vancouver" …})
                …])
(def rooms     [(ref {:id 101 …}) …])
(def cars      [(ref {:id "ABC123" …}) …])

(def customers [(ref {:orig "London" :dest "Vancouver"
                      :start "2017-10-22" :end "2017-10-27"
                      :password nil})
                …])
```

```
(defn process-customer [c]
  (dosync
    (reserve-flight (:orig @c) (:dest @c) (:start @c))
    (reserve-flight (:dest @c) (:orig @c) (:end @c))
    (reserve-room   (:dest @c) (:start @c) (:end @c))
    (reserve-car    (:dest @c) (:start @c) (:end @c))
    (ref-set c (assoc @c :password (generate-password)))))
```

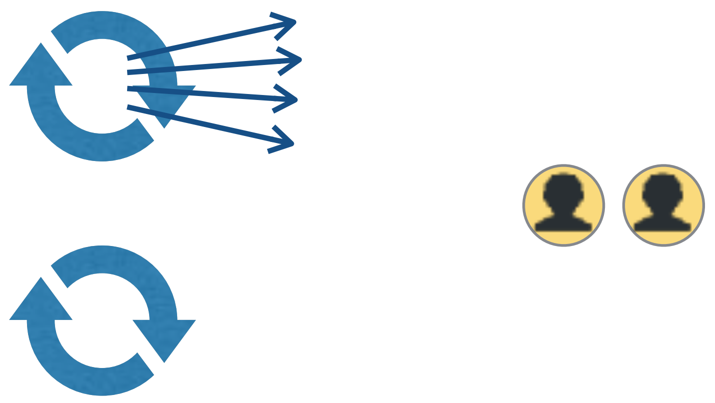
Based on: Minh, Chung, Kozyrakis, Olukotun (2008). *STAMP: Stanford Transactional Applications for Multi-Processing* (IISWC'08)

Customers are processed in parallel



```
(defn process-customer [c]
  (dosync
    (reserve-flight (:orig @c) (:dest @c) (:start @c))
    (reserve-flight (:dest @c) (:orig @c) (:end @c))
    (reserve-room   (:dest @c) (:start @c) (:end @c))
    (reserve-car    (:dest @c) (:start @c) (:end @c))
    (ref-set c (assoc @c :password (generate-password)))))
```

But more fine-grained parallelization is possible



```
(defn process-customer [c]
  (dosync
    (send (rand workers) :flight (:orig @c) …)
    (send (rand workers) :flight (:dest @c) …)
    (send (rand workers) :room   (:dest @c) …)
    (send (rand workers) :car    (:dest @c) …)
    (ref-set c (assoc @c :password (generate-password)))))
```

serializability broken

Observations:

Actors often share memory
⇒ **races & deadlocks possible**

Transactions contain subtasks that may be parallelized
⇒ **serializability broken**

Actors + Transactions = Problems

Solution: Transactional Actors

	Transaction in...	Actor in ...
...in transaction	**(dosync** (dosync (ref v) (deref r) (ref-set r v))) ✓	**(dosync** (behavior [] [] …) (spawn beh state) (become beh state) (send actor msg)) ?
...in actor	**(behavior [] []** (dosync (ref v) (deref r) (ref-set r v))) ?	**(behavior [] []** (behavior [] [] …) (spawn beh state) (become beh state) (send actor msg)) ✓

Transactional memory in actors

Similar to thread-based systems

```
(behavior [] [c]
  (process-customer c))
```

$\longrightarrow$

```
(defn process-customer [c]
 (dosync
   (reserve-flight (:orig @c) (:dest @c) (:start
   (reserve-flight (:dest @c) (:orig @c) (:end @c
   (reserve-room   (:dest @c) (:start @c) (:end @
   (reserve-car    (:dest @c) (:start @c) (:end @
   (ref-set c (assoc @c :password (generate-passw
```

Actors in a transaction

Difficulty: side effects in transaction

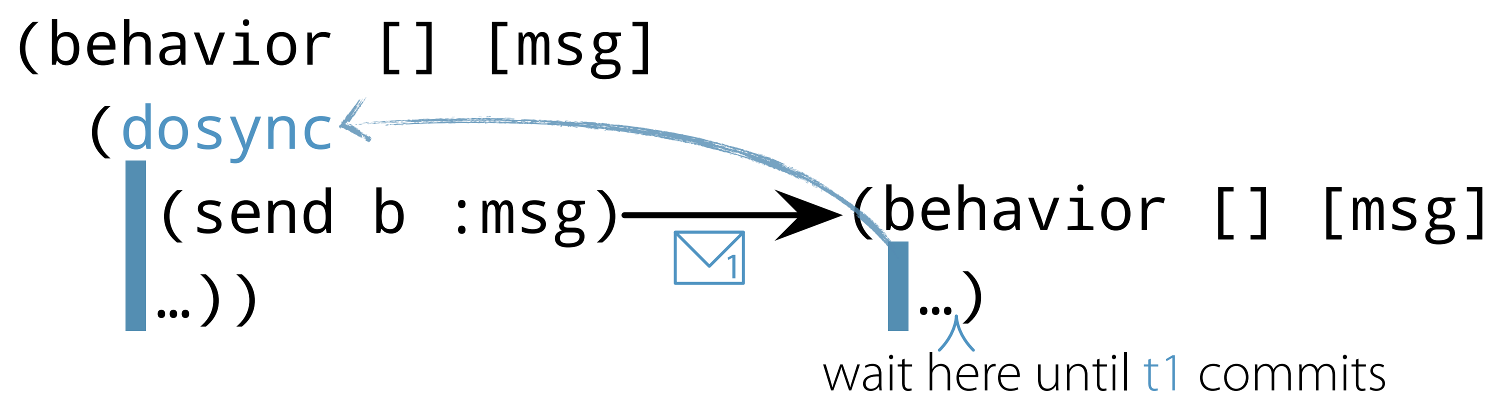
```
(dosync

  (def airline-beh
    (behavior [flights]
      …))


  (spawn airline-beh @flights)
  (become airline-beh @c)


  (send :process-customer @c))
```

separate from transaction, no side-effect ✓

delay side effect until commit (pessimistic) ↷

sent immediately, but **rolled back** on abort (optimistic) ↶

Sending a message in a transaction



```
(behavior [] [msg]
  (dosync
   (send b :msg)
   …))
```

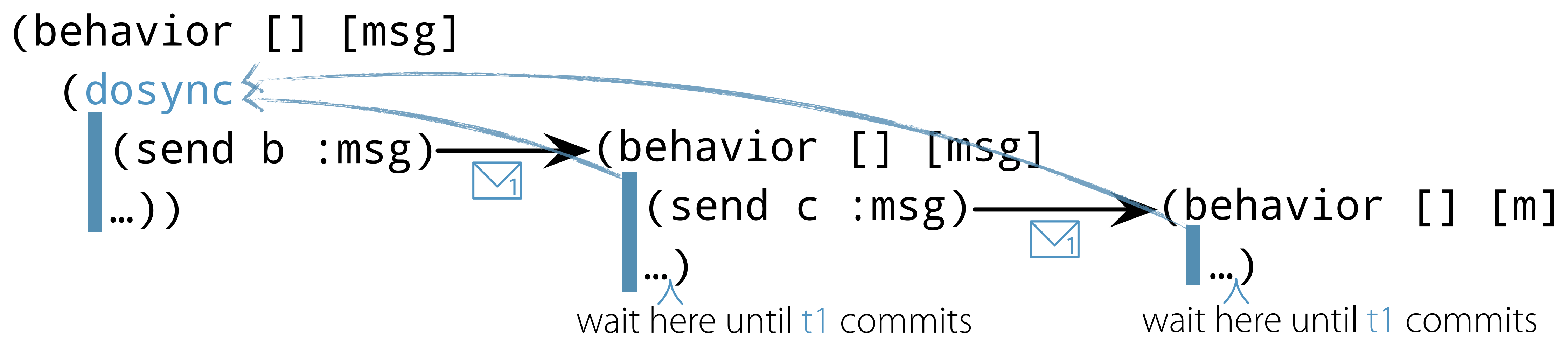
```
(behavior [] [msg]
 …)
```

Message **depends** on the transaction

Receiving turn is **tentative**:

- Side effects (`spawn`, `become`) delayed
- At the end, wait for dependency to commit

Special case: Message in tentative turn



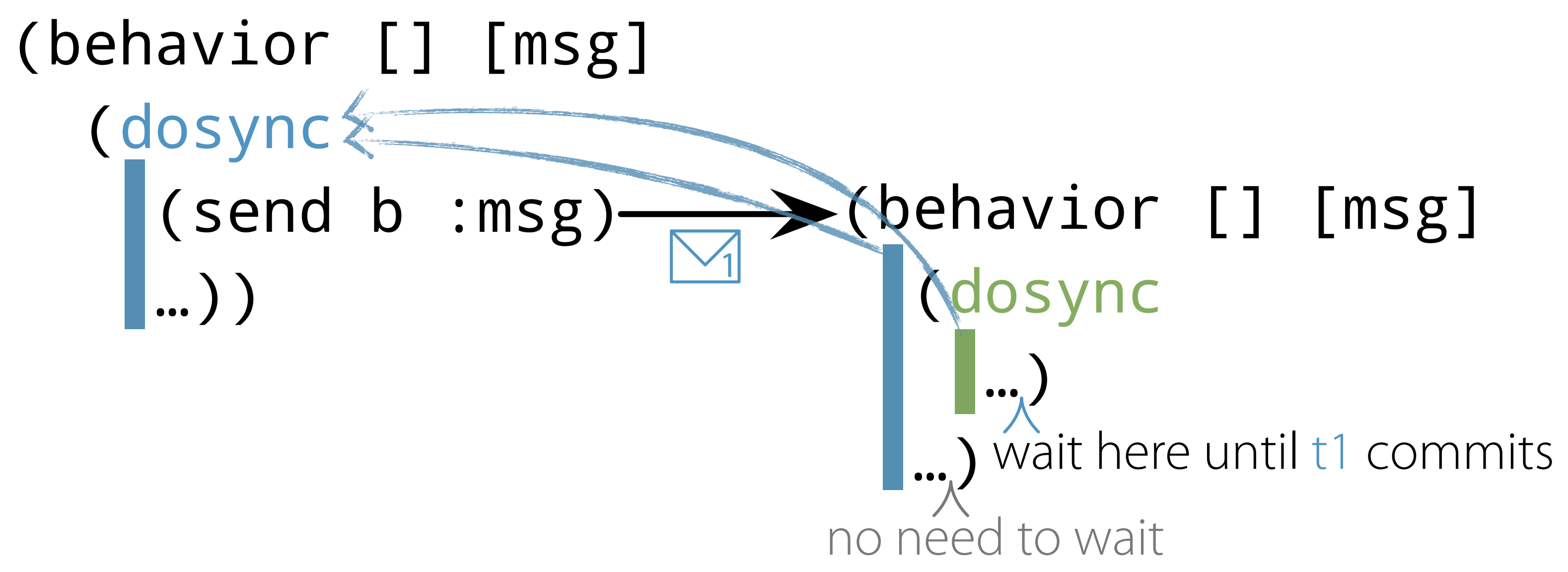
```
(behavior [] [msg]
  (dosync
    (send b :msg)
    …))
```

```
(behavior [] [msg]
  (send c :msg)
  …)
```

```
(behavior [] [m]
  …)
```

Dependency is **forwarded**

Special case: Transaction in tentative turn



```
(behavior [] [msg]
  (dosync
   (send b :msg))
   …))
```

```
(behavior [] [msg]
  (dosync
   …)
  …)
```

Transaction in tentative turn waits before it commits

⇒ serializability maintained

Properties

Serializability

side effects on actors part of transaction

but: other side effects not allowed in tentative turns

Free from deadlocks

dependencies always from new to old

but: transactions cannot cross turns

Free from low-level races

granularity of turns & transactions

Implementation

Fork of Clojure

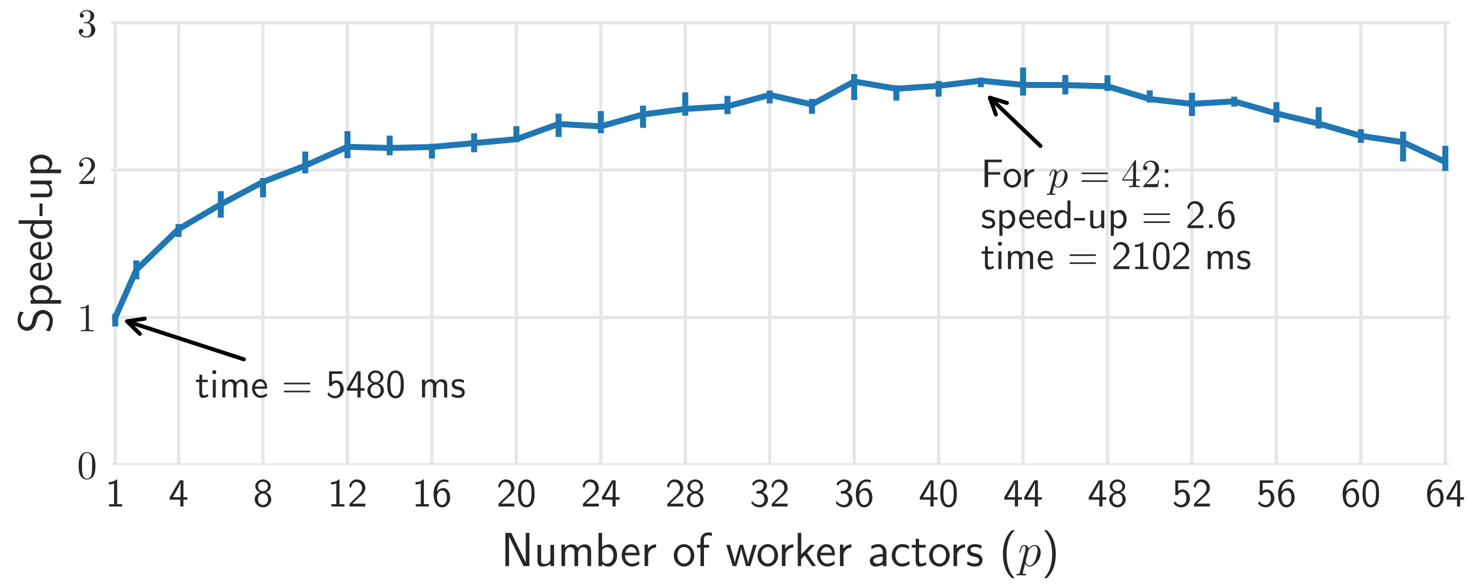
- STM built-in
- Regular actors added
- Transactional Actors as modifications of STM & actors

Details in paper

https://github.com/jswalens/transactional-actors

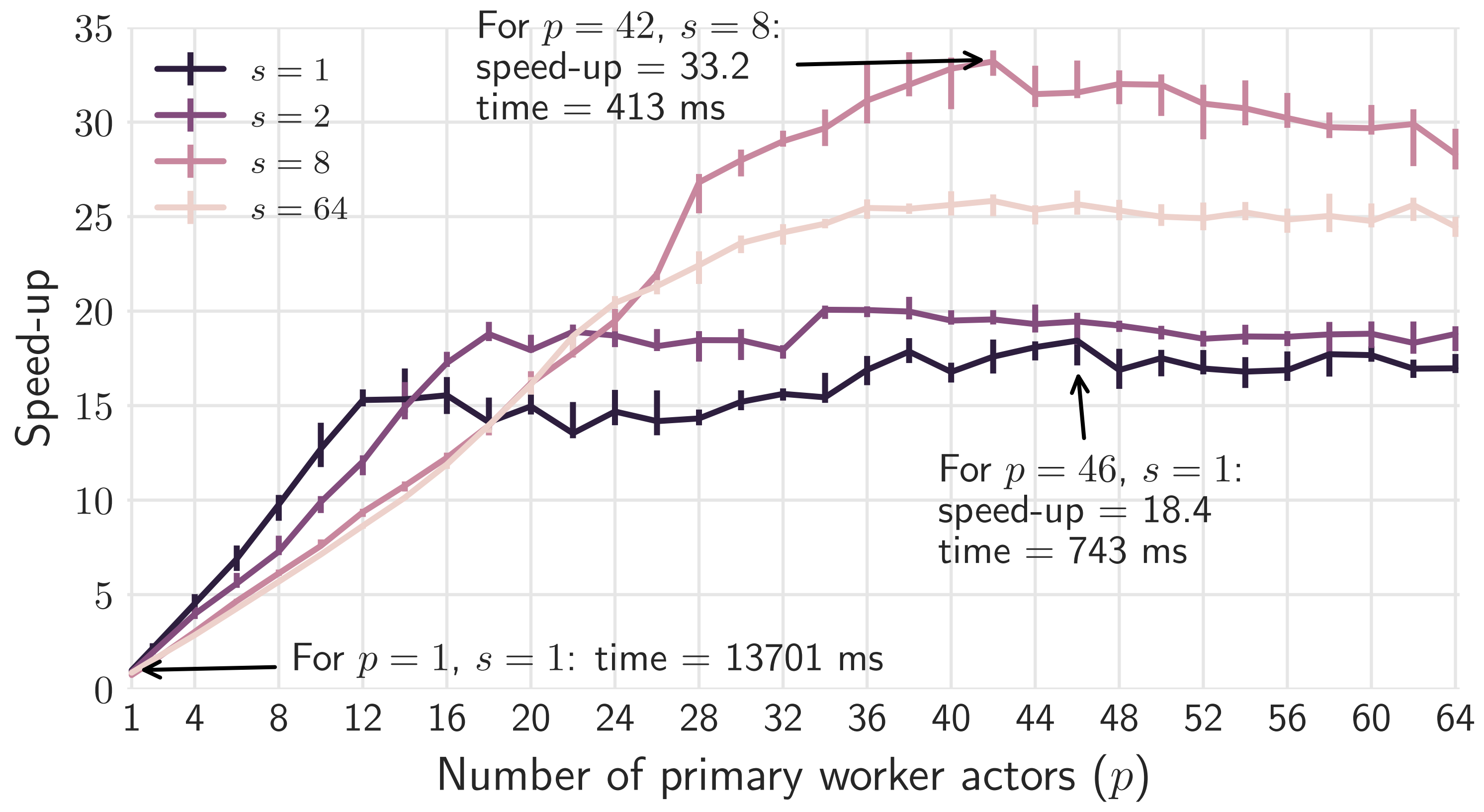
Evaluation: Vacation benchmark

Original



Speed-up limited: 2.6 because of conflicts

Transactional Actors



Better speed-up due to:

- finer-grained parallelism
- fewer/cheaper conflicts

For 1 thread: much slower

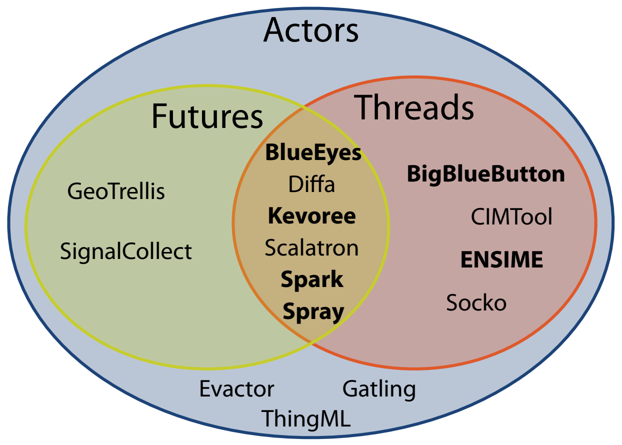
Limitations & Future Work

- Implement optimizations
- Evaluate:
  - **More benchmark applications** (suggestions?)
  - **Comparison with related work** (performance & software quality atttributes)
- Formalize of semantics and properties

Summary

Problem:

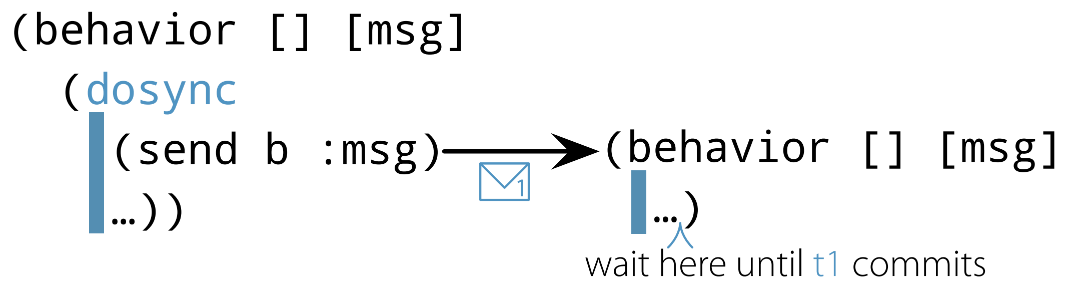
Shared memory & message passing often combined, but breaks properties



Solution:

Transactional Actors

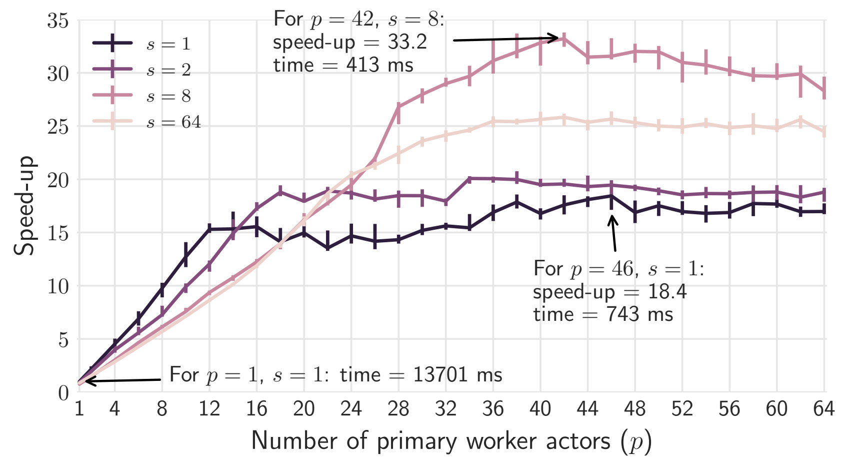
- Messages have a dependency
- Transaction aborts
  ⇒ all its effects are rolled back



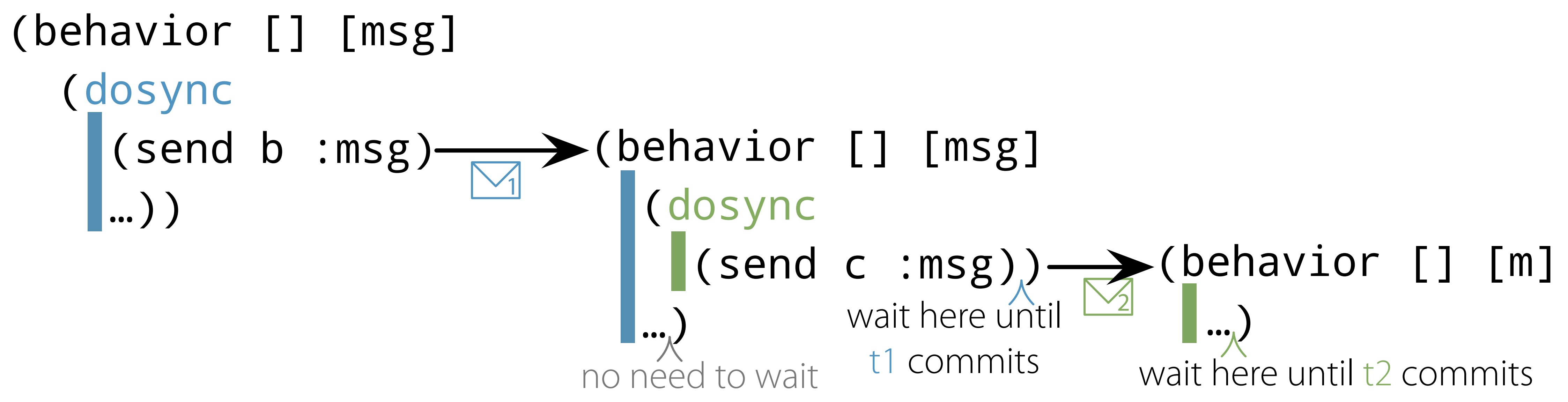
```
(behavior [] [msg]
  (dosync
   (send b :msg)──────▶(behavior [] [msg]
   …))                  …)
```

Benefits:

- serializable, deadlock free, race free
- finer-grained parallelism
  ⇒ higher speed-up



Message in transaction in tentative turn



```
(behavior [] [msg]
  (dosync
   (send b :msg)
   …))
```

```
(behavior [] [msg]
 (dosync
  (send c :msg))
 …)
```

```
(behavior [] [m]
 …)
```