Chocola: Integrating Futures, Actors, and Transactions

Janwillem Swalens
Joeri De Koster
Wolfgang De Meuter

Software Languages Lab

VUB VRIJE UNIVERSITEIT BRUSSEL

There are many different concurrency models

Fork/Join locks Communicating Sequential Processes

futures promises threads actors active objects

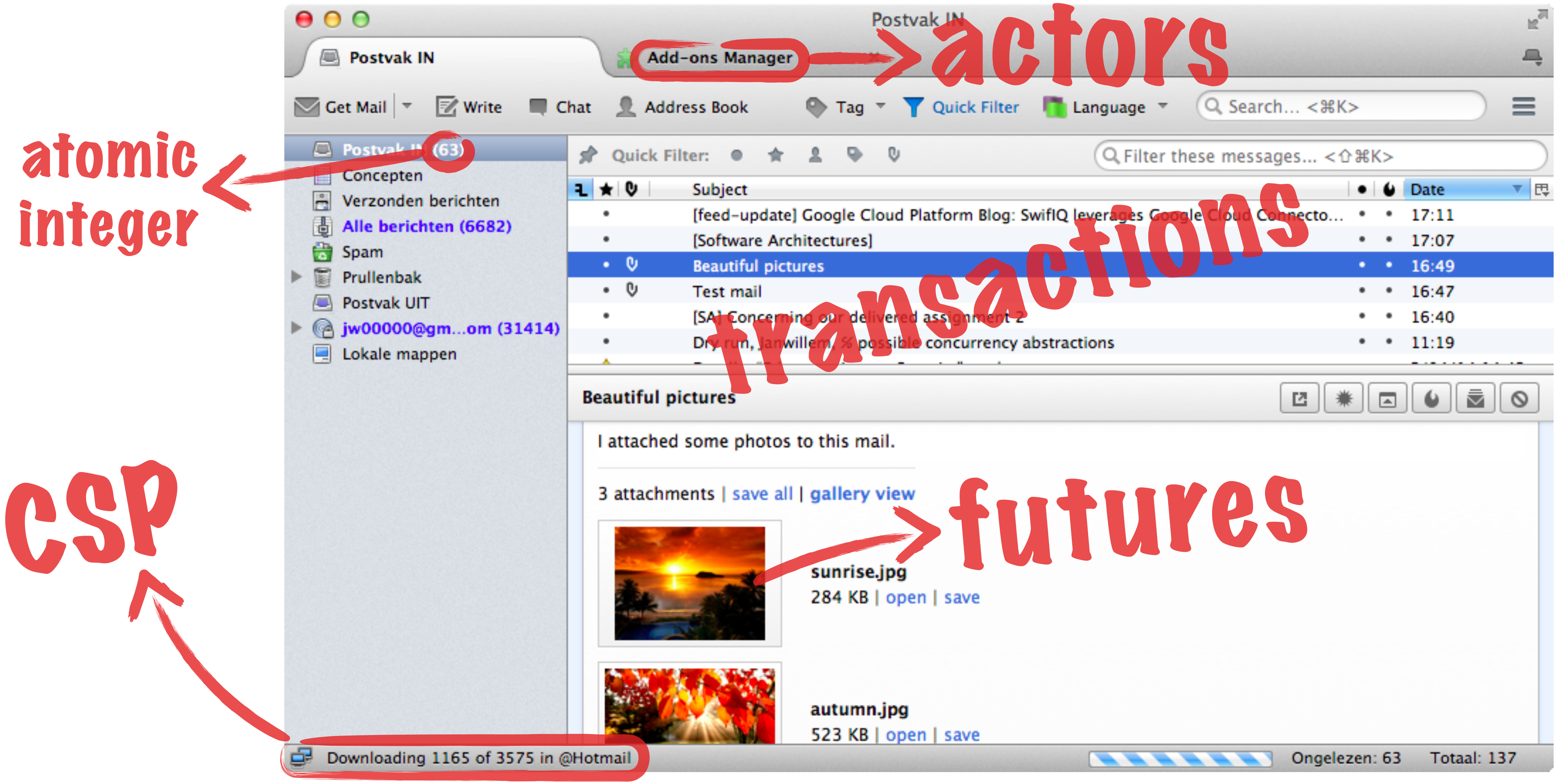
Nested Data Parallelism

dataflow Software Transactional Memory MPI

Concurrent Revisions worlds OpenMP transactional events

speculative parallelism

Programmers combine these in a single application



Subject	Date
[feed-update] Google Cloud Platform Blog: SwiftIQ leverages Google Cloud Connecto…	17:11
[Software Architectures]	17:07
Beautiful pictures	16:49
Test mail	16:47
[SA] Concerning our delivered assignment 2	16:40
Dry run, Janwillem, % possible concurrency abstractions	11:19

Beautiful pictures

I attached some photos to this mail.

3 attachments | save all | gallery view

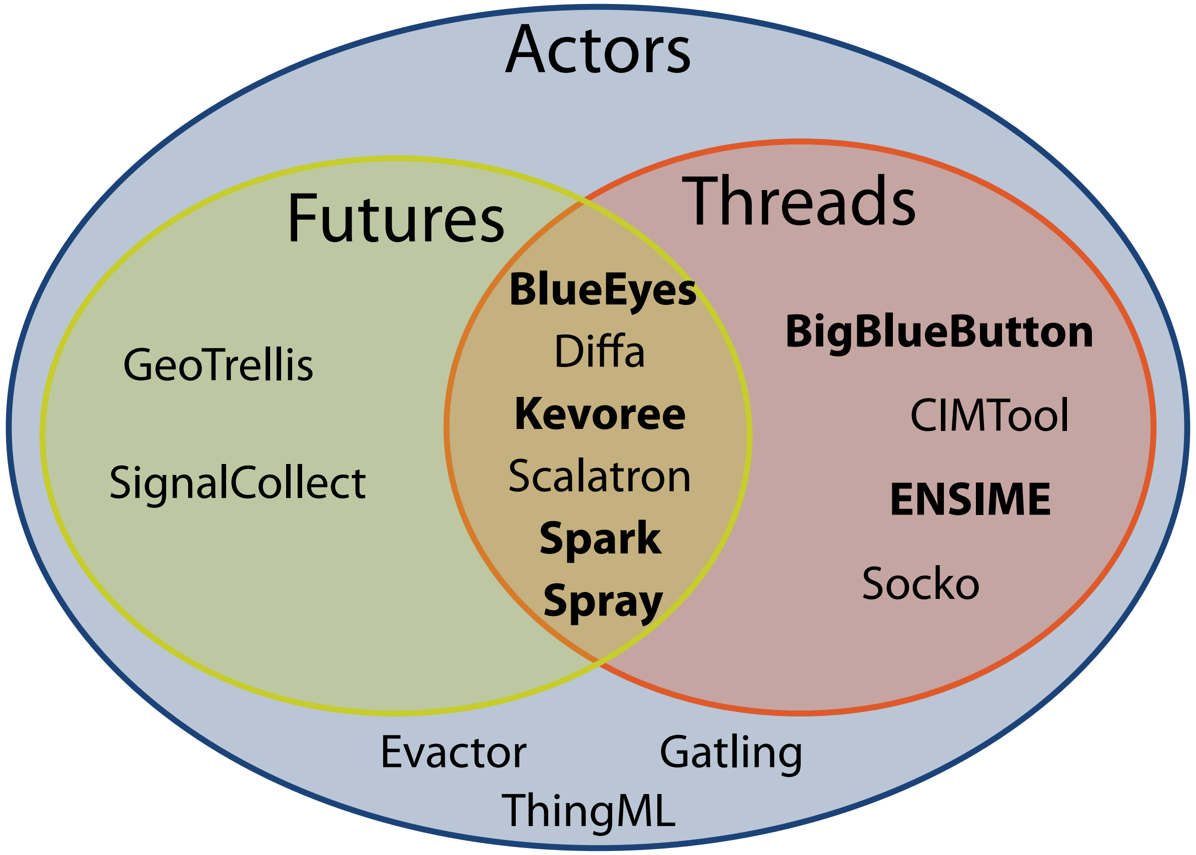
sunrise.jpg
284 KB | open | save

autumn.jpg
523 KB | open | save

Downloading 1165 of 3575 in @Hotmail

Ongelezen: 63 Totaal: 137

Observation 1: programmers combine concurrency models



15 Scala programs with **actors**:

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

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

Observation 2: programming languages support many concurrency models

	Clojure	Scala	Java	Haskell	C++
Deterministic models					
Futures	✓	✓	✓	•	✓
Promises	✓	✓	✓	•	✓
Fork/Join	✓*	✓*	✓		•
Parallel collections	✓*	✓	✓	•	•
Dataflow	•	•	•	•	
Shared-memory models					
Threads	✓*	✓*	✓	✓	✓
Locks	✓*	✓*	✓	✓	✓
Atomic variables	✓	✓*	✓	✓	✓
Transactional memory	✓	•	•	✓	•
Message-passing models					
Actors	•	•	•	•	•
Channels	✓	✓	•	✓	•
Agents	✓				
# supported models	10	8	7	5	5

✓ built in
• library

Clojure has 10 concurrency models built in

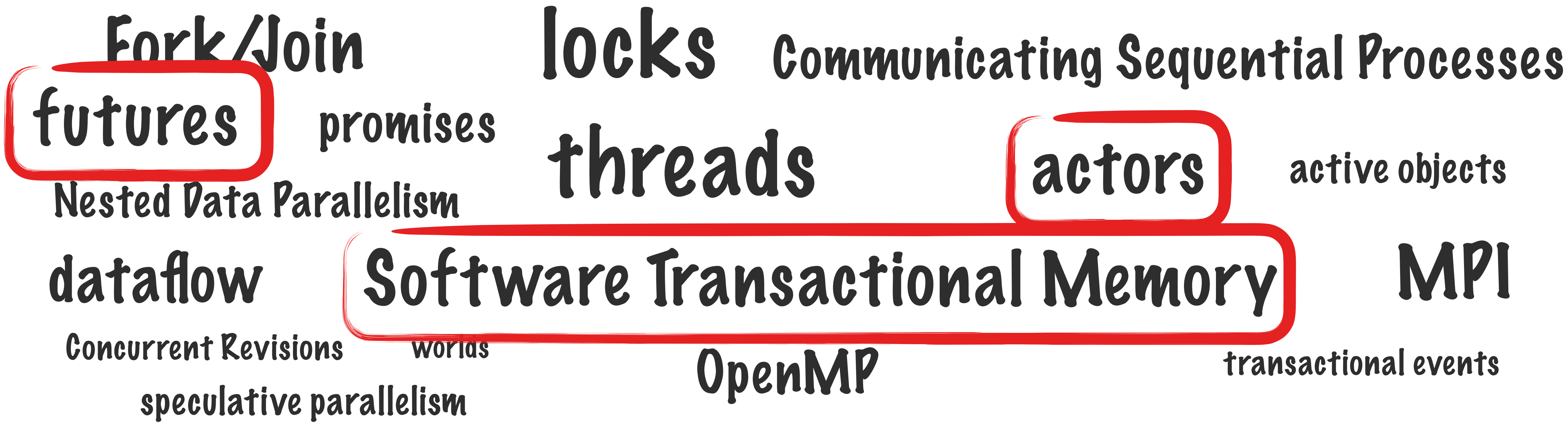
Programmers combine multiple concurrency models

Which problems can this cause?

Are the usual guarantees of concurrency models broken?

Goal

Framework that combines:



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

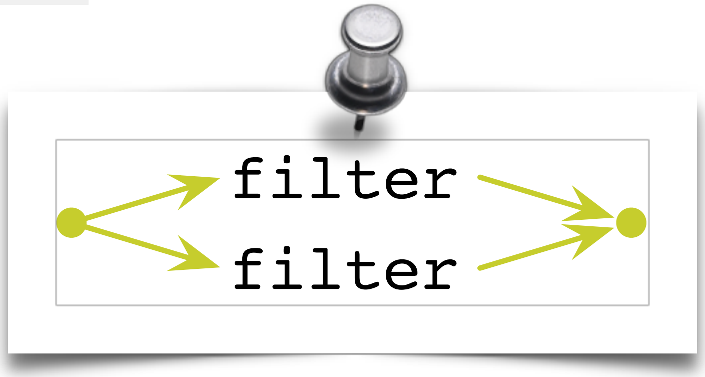
1 Separate models: backward compatibility

2 Combinations: maintain guarantees of all models

If impossible: define a less restrictive guarantee

Futures

```
(defn parallel-filter [f xs]
  (let [[part1 part2] (partition 2 xs)
        future1 (fork (filter f part1))
        future2 (fork (filter f part2))]
    (concat (join future1) (join future2))))
```



Guarantee:

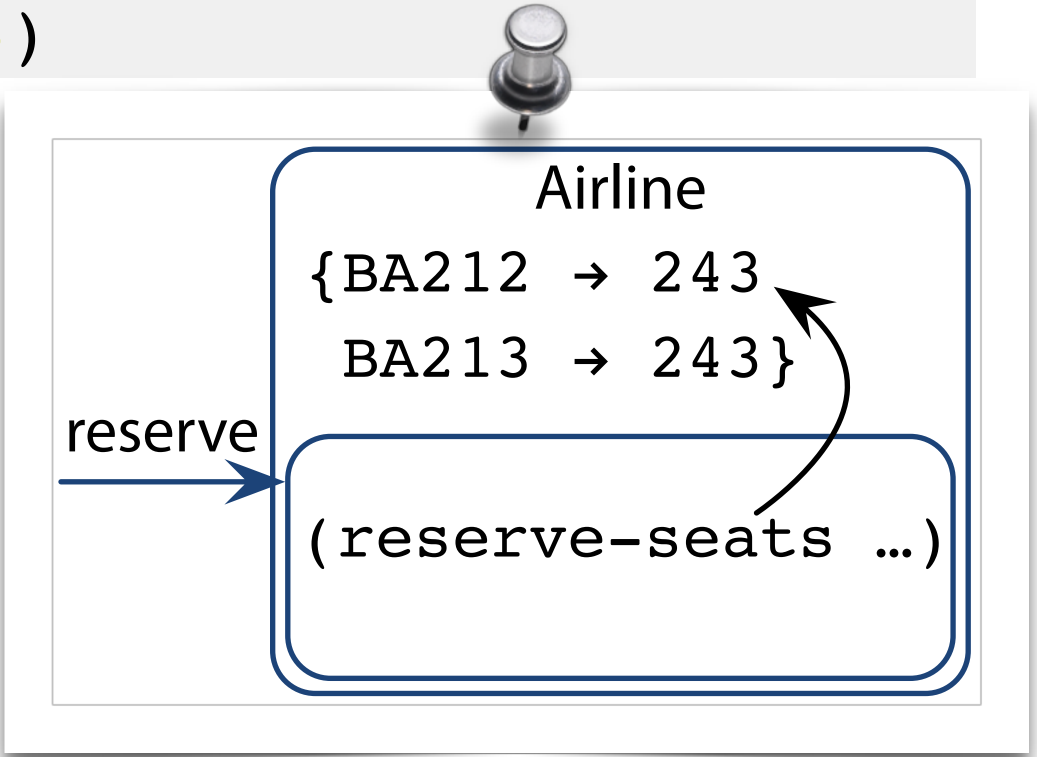
Det **determinacy**

Actors

```
(def flights
  {"BA212" {:from "BOS" :to "LHR" :price 499 :seats 243}
   "BA213" {:from "LHR" :to "BOS" :price 499 :seats 243}})

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

(def british-airways (spawn airline-behavior flights))
(send british-airways "LHR" "BOS" 3)
```



Guarantees:

ITP **isolated turn principle***

DLF **deadlock freedom**

* De Koster, Van Cutsem, De Meuter (2016). *43 Years of Actors: A Taxonomy of Actor Models and Their Key Properties* (AGERE'16)

Transactions

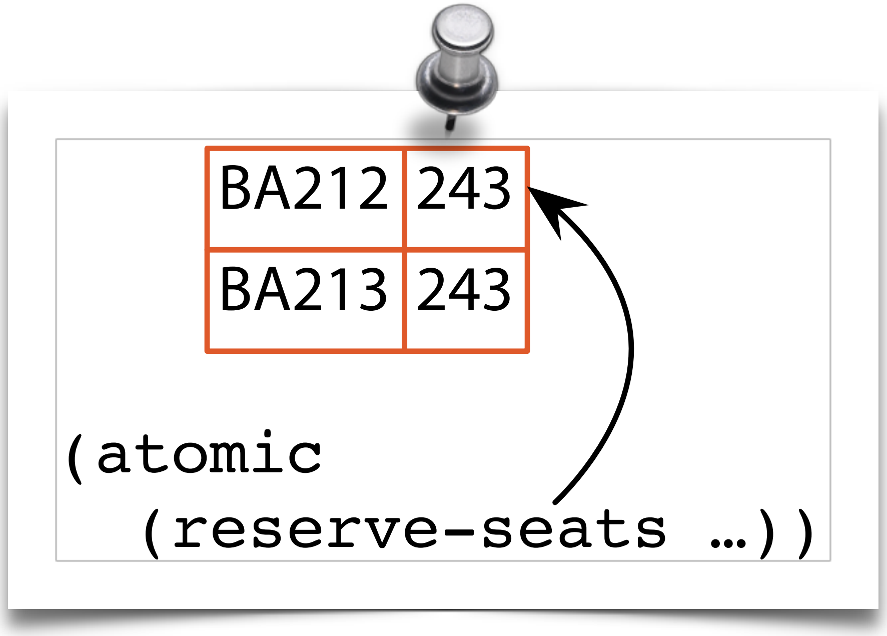
```
(def flights
  {"BA212" (ref {:from "BOS" :to "LHR" … :seats 243})
   "BA213" (ref {:from "LHR" :to "BOS" … :seats 243})})

(defn reserve-seats [flight n]
  (let [flight’ (update (deref flight) :seats - n)]
    (ref-set flight flight’)))

(atomic
  (reserve-seats (get flights "BA213") 3)
  (reserve-seats (get flights "BA212") 3))
```

Guarantees:

- Iso **isolation** (e.g. serializability)
- Pro **progress** (e.g. deadlock freedom)



BA212	243
BA213	243

```
(atomic
  (reserve-seats …))
```

Summary

Futures	Transactions	Actors
Deterministic	*Shared memory*	*Message passing*
`(fork e)`	`(atomic e)`	`(behavior [x] [x] e)`
`(join f)`	`(ref v)`	`(spawn b v)`
	`(deref r)`	`(send a v)`
	`(ref-set r v)`	`(become b v)`
Det Determinacy	Iso Isolation	ITP Isolated turn principle
	Pro Progress	DLF Deadlock freedom

We studied the combinations of futures, transactions, and actors

		inner		
	→ in↓	Future	Transaction	Actor

outer	Future	`(fork`
		`  (fork …)`
		`  (join …))`
			`(fork`
			`  (atomic …))`
				`(fork`
				`  (spawn …)`
				`  (send …)`
				`  (become …))`

	Transaction	`(atomic`
		`  (fork …)`
		`  (join …))`
			`(atomic`
			`  (atomic …)`
			`  (ref …)`
			`  (deref …)`
			`  (ref-set …))`
				`(atomic`
				`  (spawn …)`
				`  (send …)`
				`  (become …))`

	Actor	`(behavior [] []`
		`  (fork …)`
		`  (join …))`
			`(behavior [] []`
			`  (atomic …))`
				`(behavior [] []`
				`  (spawn …)`
				`  (send …)`
				`  (become …))`

“Naive” combinations cause problems

		inner		
	→ in ↓	Future	Transaction	Actor
outer	Future	Nested futures Det	Parallel transactions ~~Det~~ Iso Pro	Communication in future ~~Det~~ ~~ITP~~ DLF
	Transaction	Parallelism in transaction ~~Det~~ ~~Iso~~ Pro	Nested transactions Iso Pro	Communication in transaction ~~Iso~~ Pro ~~ITP~~ DLF
	Actor	Parallelism in actor Det ~~ITP~~ DLF	Shared memory in actor Iso Pro ~~ITP~~ DLF	Actors ITP DLF

“Naive” combinations cause problems

		inner		
	→ in↓	Future	Transaction	Actor
outer	Future	Nested futures Det	Parallel transactions ~~Det~~ Iso Pro	Communication in future ~~Det~~ ~~ITP~~ DLF
	Transaction	Parallelism in transaction ~~Det~~ ~~Iso~~ Pro	Nested transactions Iso Pro	Communication in transaction ~~Iso~~ Pro ~~ITP~~ DLF
	Actor	Parallelism in actor Det ~~ITP~~ DLF	Shared memory in actor Iso Pro ~~ITP~~ DLF	Actors ITP DLF

Swalens, De Koster, De Meuter (2016). *Transactional Tasks: Parallelism in Software Transactions* (ECOOP’16)

Swalens, De Koster, De Meuter (2017). *Transactional Actors: Communication in Transactions?* (SEPS’17)

Actors & Transactions

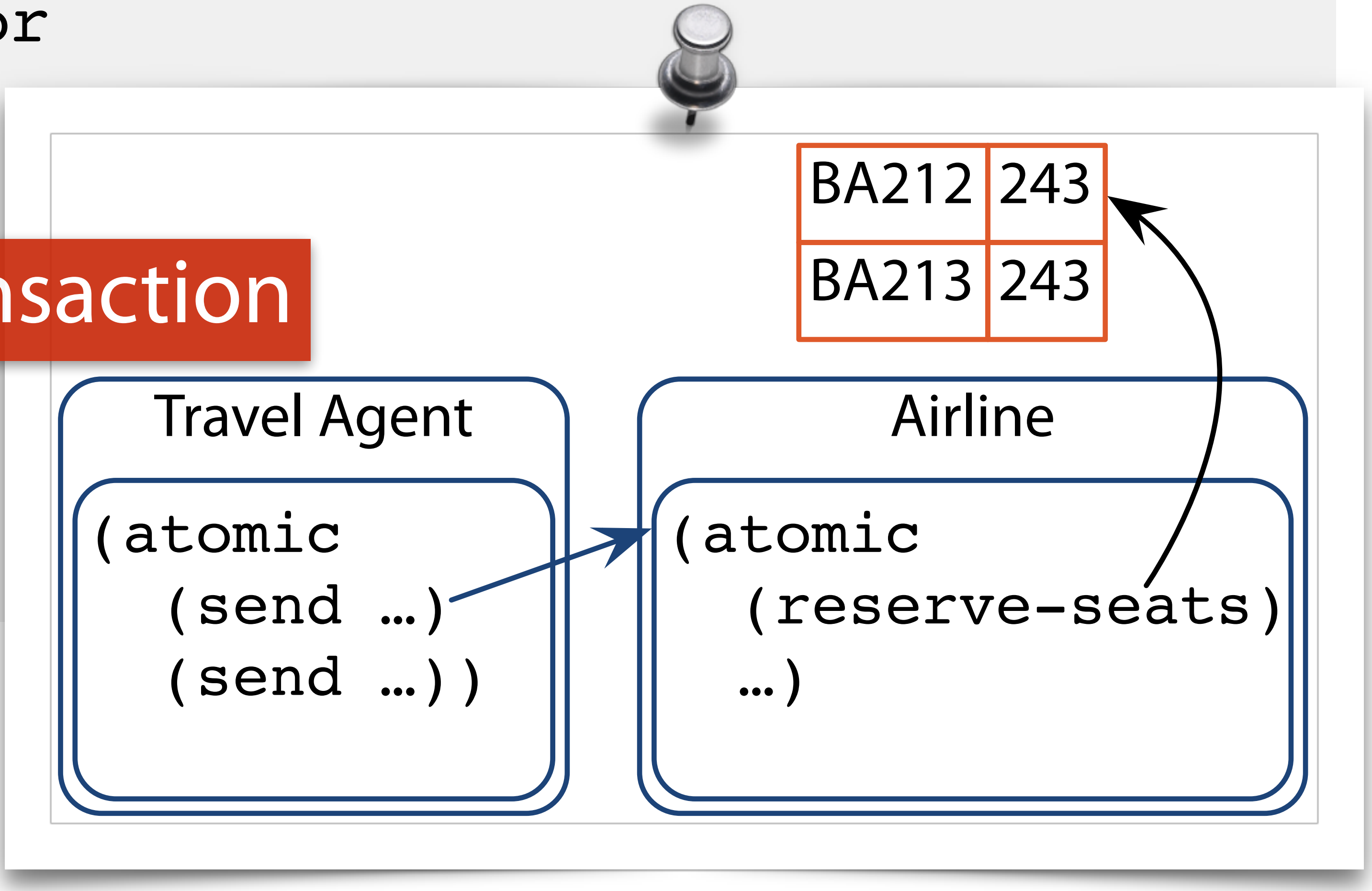
```
(def airline-behavior
  (behavior []
    [orig dest n]
    (atomic
      (let [flight (search-flight flights orig dest)]
        (reserve-seats flight n)))))

(def airline (spawn airline-behavior))

(def travel-agent-behavior
  (behavior []
    [orig dest n]
    (atomic
      (send airline
        orig dest n)
      (send airline
        dest orig n))))
```

transaction in actor

actor in transaction



BA212	243
BA213	243

```
(atomic
  (send …)
  (send …))
```

```
(atomic
  (reserve-seats)
  …)
```

Actors & Transactions

Actor in transaction

```
(atomic
  (send airline o d)
  (send airline d o)
  (ref-set …))
```

⇒ ~~Iso~~

Solution:

Tentative messages, “unsent” if transaction aborts

Iso

Transaction in actor

```
(behavior […]
  […]
  (atomic
    …))
```

⇒ ~~ITP~~

Solution:

Inevitable, so we introduce Low-Level Race Freedom

~~ITP~~ → LLRF

Swalens, De Koster, De Meuter (2017). *Transactional Actors: Communication in Transactions?* (SEPS’17)

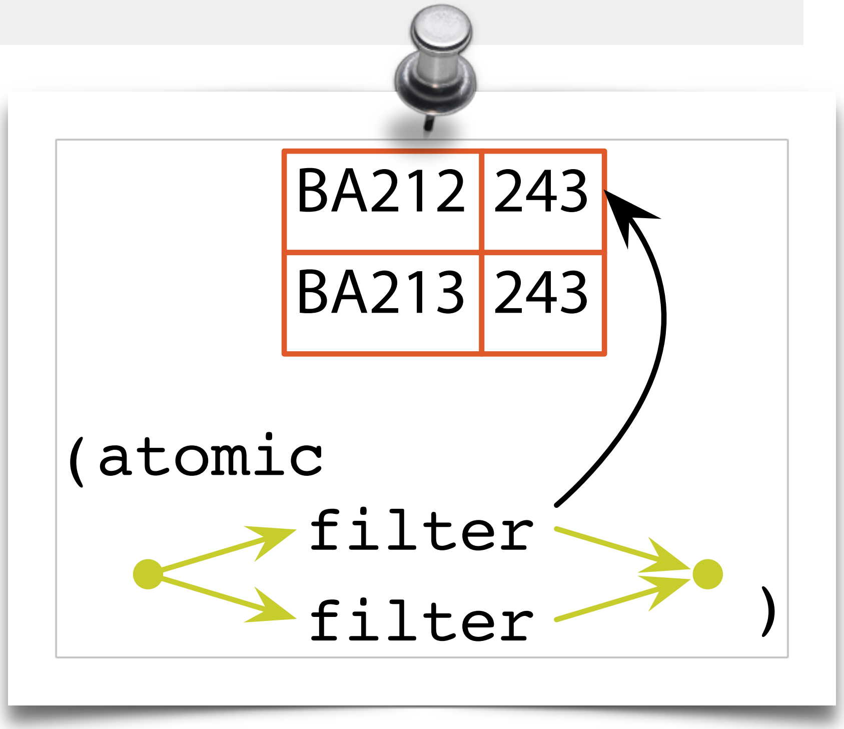
Transactions & Futures

```
(def airline-behavior
  (behavior []
    [orig dest n]
    (atomic
      (let [flight (search-flight flights orig dest)]
        (reserve-seats flight n)))))
```

```
(defn search-flight [flights orig dest]
  (first
    (parallel-filter
      (fn [flight] (and (= (get @flight :from) orig)
                        (= (get @flight :to) dest)))
      (vals flights))))
```

future in transaction

```
(defn parallel-filter [f xs]
  (let [[part1 part2] (partition 2 xs)
        future1 (fork (filter f part1))
        future2 (fork (filter f part2))]
    (concat (join future1) (join future2))))
```



BA212	243
BA213	243

```
(atomic
     filter
     filter     )
```

Transactions & Futures

Future in transaction

```
(atomic
  (fork (filter f part1))
  (fork (filter f part2)))
```

⇒ ~~Iso~~

Solution:

Futures work on conceptual copy of transactional memory

Their changes are joined into parent

Iso

Transaction in future

```
(fork
  (atomic
    …))
```

⇒ ~~Det~~

Solution:

Inevitable and expected in languages with transactions, so we introduce Intratransaction Determinacy

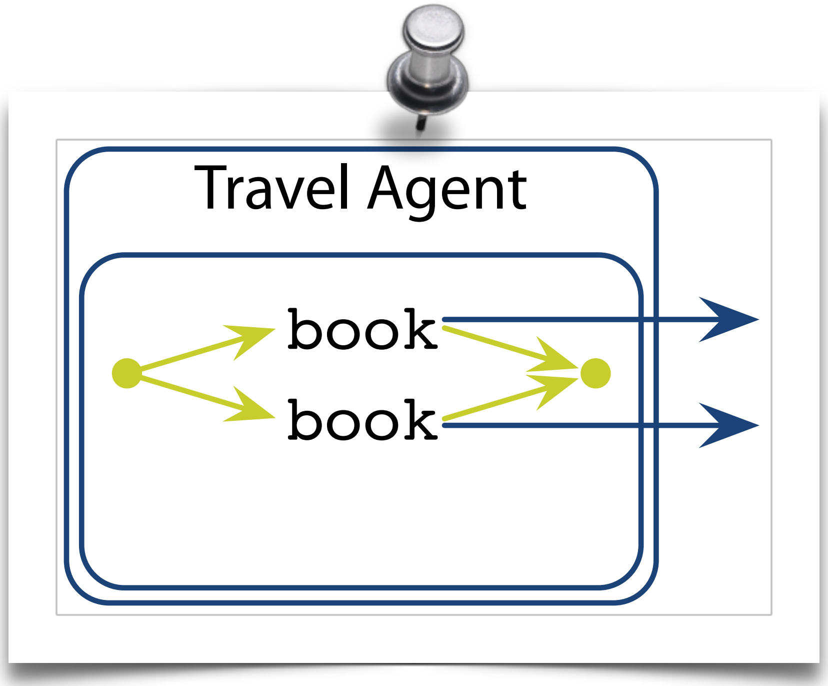
~~Det~~ → ITD

Swalens, De Koster, De Meuter (2016). *Transactional Tasks: Parallelism in Software Transactions* (ECOOP'16)

Futures & Actors

```
(def travel-agent-behavior
  (behavior []
    [orig dest n]
    (fork (book-flight orig dest n))
    (fork (book-flight dest orig n))))
```

future in actor



Futures & Actors

Future in actor

```
(behavior […]
  […]
  (fork (book-flight o d))
  (fork (book-flight d o)))
```

⇒ ~~ITP~~

Solution:

Require all futures to be joined before end of turn

ITP

Actor in future

```
(fork
  (send (filter f part1))
  (send (filter f part2)))
```

⇒ ~~Det~~

Solution:

Inevitable, but expected

Chocola:
c^{h}omposable concurrency language

inner

outer

→ in ↓	Future	Transaction	Actor
Future	Nested futures Det	Parallel transactions ~~Det~~ Iso Pro	Communication in future ~~Det~~ ITP DLF
Transaction	Parallelism in transaction ~~Det~~ → ITD Iso Pro	Nested transactions Iso Pro	Communication in transaction Iso Pro ~~ITP~~ → LLRF DLF
Actor	Parallelism in actor Det ITP DLF	Shared memory in actor Iso Pro ~~ITP~~ → LLRF DLF	Actors ITP DLF

Implementation

Extension of Clojure

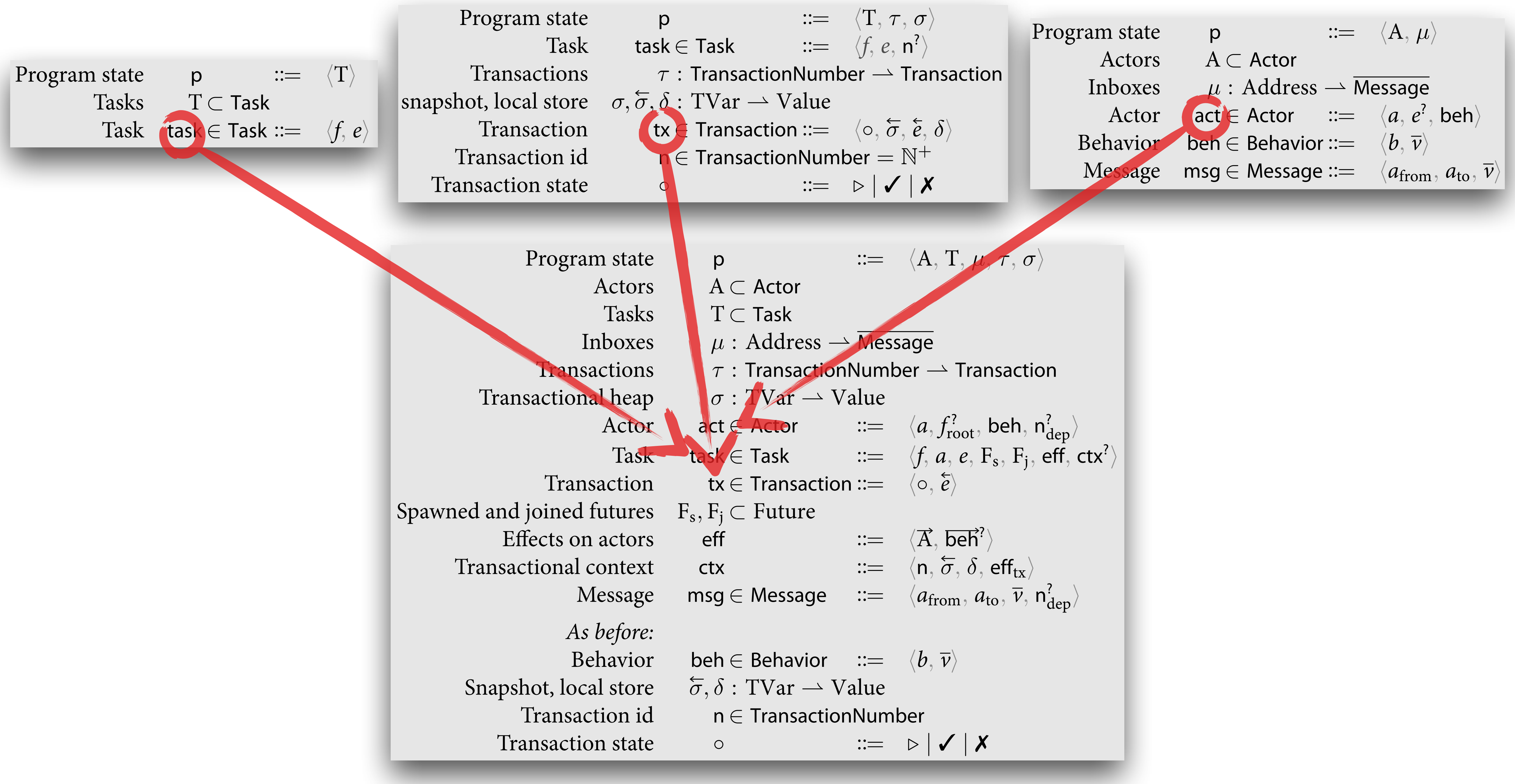
- **Futures & Transactions**: built into Clojure
- **Actors**: simple implementation
- **Combinations**: by modifying the above

http://chocola.soft.brussels

https://github.com/jswalens/chocolalib

Formalization of operational semantics

Uniform formalization of three separate models



Program state	p	::=	$\langle \mathrm{T} \rangle$
Tasks	$\mathrm{T} \subset$ Task		
Task	task $\in$ Task	::=	$\langle f, e \rangle$

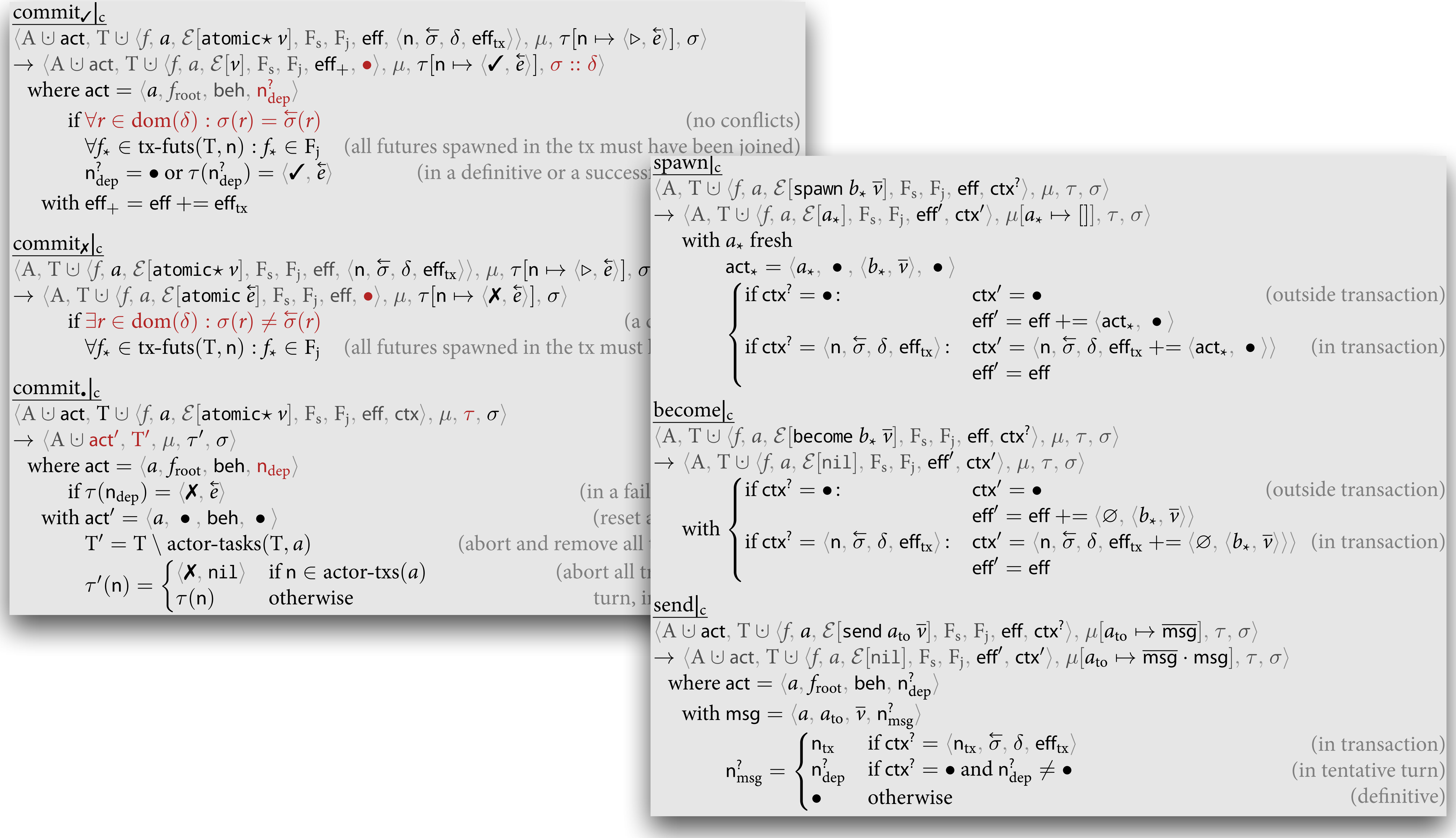
Program state	p	::=	$\langle \mathrm{T}, \tau, \sigma \rangle$
Task	task $\in$ Task	::=	$\langle f, e, \mathrm{n}^{?} \rangle$
Transactions	τ : TransactionNumber $\rightharpoonup$ Transaction		
snapshot, local store	$\sigma, \overleftarrow{\sigma}, \delta$: TVar $\rightharpoonup$ Value		
Transaction	tx $\in$ Transaction	::=	$\langle \circ, \overleftarrow{\sigma}, \overleftarrow{e}, \delta \rangle$
Transaction id	n $\in$ TransactionNumber $= \mathbb{N}^{+}$		
Transaction state	$\circ$	::=	$\triangleright \mid$ ✓ $\mid$ ✗

Program state	p	::=	$\langle \mathrm{A}, \mu \rangle$
Actors	$\mathrm{A} \subset$ Actor		
Inboxes	μ : Address $\rightharpoonup \overline{\text{Message}}$		
Actor	act $\in$ Actor	::=	$\langle a, e^{?}, \text{beh} \rangle$
Behavior	beh $\in$ Behavior	::=	$\langle b, \overline{v} \rangle$
Message	msg $\in$ Message	::=	$\langle a_{\text{from}}, a_{\text{to}}, \overline{v} \rangle$

Program state	p	::=	$\langle \mathrm{A}, \mathrm{T}, \mu, \tau, \sigma \rangle$
Actors	$\mathrm{A} \subset$ Actor		
Tasks	$\mathrm{T} \subset$ Task		
Inboxes	μ : Address $\rightharpoonup \overline{\text{Message}}$		
Transactions	τ : TransactionNumber $\rightharpoonup$ Transaction		
Transactional heap	σ : TVar $\rightharpoonup$ Value		
Actor	act $\in$ Actor	::=	$\langle a, f^{?}_{\text{root}}, \text{beh}, \mathrm{n}^{?}_{\text{dep}} \rangle$
Task	task $\in$ Task	::=	$\langle f, a, e, \mathrm{F_s}, \mathrm{F_j}, \text{eff}, \text{ctx}^{?} \rangle$
Transaction	tx $\in$ Transaction	::=	$\langle \circ, \overleftarrow{e} \rangle$
Spawned and joined futures	$\mathrm{F_s}, \mathrm{F_j} \subset$ Future		
Effects on actors	eff	::=	$\langle \overrightarrow{\mathrm{A}}, \overline{\overrightarrow{\text{beh}}}^{?} \rangle$
Transactional context	ctx	::=	$\langle \mathrm{n}, \overleftarrow{\sigma}, \delta, \text{eff}_{\text{tx}} \rangle$
Message	msg $\in$ Message	::=	$\langle a_{\text{from}}, a_{\text{to}}, \overline{v}, \mathrm{n}^{?}_{\text{dep}} \rangle$
As before:			
Behavior	beh $\in$ Behavior	::=	$\langle b, \overline{v} \rangle$
Snapshot, local store	$\overleftarrow{\sigma}, \delta$: TVar $\rightharpoonup$ Value		
Transaction id	n $\in$ TransactionNumber		
Transaction state	$\circ$	::=	$\triangleright \mid$ ✓ $\mid$ ✗

Formalization

same constructs, but take context into account



$\underline{\text{commit}_{\checkmark}|_{c}}$

$\langle A \uplus \mathsf{act}, T \uplus \langle f, a, \mathcal{E}[\mathtt{atomic}{\star}\ v], F_s, F_j, \mathsf{eff}, \langle \mathsf{n}, \overleftarrow{\sigma}, \delta, \mathsf{eff}_{tx} \rangle\rangle, \mu, \tau[\mathsf{n} \mapsto \langle \triangleright, \overleftarrow{e} \rangle], \sigma \rangle$

$\rightarrow \langle A \uplus \mathsf{act}, T \uplus \langle f, a, \mathcal{E}[v], F_s, F_j, \mathsf{eff}_{+}, \bullet \rangle, \mu, \tau[\mathsf{n} \mapsto \langle \checkmark, \overleftarrow{e} \rangle], \sigma :: \delta \rangle$

where $\mathsf{act} = \langle a, f_{root}, \mathsf{beh}, \mathsf{n}^{?}_{dep} \rangle$

if $\forall r \in \mathrm{dom}(\delta) : \sigma(r) = \overleftarrow{\sigma}(r)$ (no conflicts)

$\forall f_{\star} \in \text{tx-futs}(T, \mathsf{n}) : f_{\star} \in F_j$ (all futures spawned in the tx must have been joined)

$\mathsf{n}^{?}_{dep} = \bullet$ or $\tau(\mathsf{n}^{?}_{dep}) = \langle \checkmark, \overleftarrow{e} \rangle$ (in a definitive or a successf

with $\mathsf{eff}_{+} = \mathsf{eff} \mathrel{+}= \mathsf{eff}_{tx}$

$\underline{\text{commit}_{\text{✗}}|_{c}}$

$\langle A, T \uplus \langle f, a, \mathcal{E}[\mathtt{atomic}{\star}\ v], F_s, F_j, \mathsf{eff}, \langle \mathsf{n}, \overleftarrow{\sigma}, \delta, \mathsf{eff}_{tx} \rangle\rangle, \mu, \tau[\mathsf{n} \mapsto \langle \triangleright, \overleftarrow{e} \rangle], \sigma$

$\rightarrow \langle A, T \uplus \langle f, a, \mathcal{E}[\mathtt{atomic}\ \overleftarrow{e}], F_s, F_j, \mathsf{eff}, \bullet \rangle, \mu, \tau[\mathsf{n} \mapsto \langle \text{✗}, \overleftarrow{e} \rangle], \sigma \rangle$

if $\exists r \in \mathrm{dom}(\delta) : \sigma(r) \neq \overleftarrow{\sigma}(r)$ (a c

$\forall f_{\star} \in \text{tx-futs}(T, \mathsf{n}) : f_{\star} \in F_j$ (all futures spawned in the tx must h

$\underline{\text{commit}_{\bullet}|_{c}}$

$\langle A \uplus \mathsf{act}, T \uplus \langle f, a, \mathcal{E}[\mathtt{atomic}{\star}\ v], F_s, F_j, \mathsf{eff}, \mathsf{ctx} \rangle, \mu, \tau, \sigma \rangle$

$\rightarrow \langle A \uplus \mathsf{act}', T', \mu, \tau', \sigma \rangle$

where $\mathsf{act} = \langle a, f_{root}, \mathsf{beh}, \mathsf{n}_{dep} \rangle$

if $\tau(\mathsf{n}_{dep}) = \langle \text{✗}, \overleftarrow{e} \rangle$ (in a fail

with $\mathsf{act}' = \langle a, \bullet, \mathsf{beh}, \bullet \rangle$ (reset a

$T' = T \setminus \text{actor-tasks}(T, a)$ (abort and remove all t

$\tau'(\mathsf{n}) = \begin{cases} \langle \text{✗}, \mathtt{nil} \rangle & \text{if } \mathsf{n} \in \text{actor-txs}(a) \\ \tau(\mathsf{n}) & \text{otherwise} \end{cases}$ (abort all tr turn, i

$\underline{\text{spawn}|_{c}}$

$\langle A, T \uplus \langle f, a, \mathcal{E}[\mathtt{spawn}\ b_{\star}\ \overline{v}], F_s, F_j, \mathsf{eff}, \mathsf{ctx}^{?} \rangle, \mu, \tau, \sigma \rangle$

$\rightarrow \langle A, T \uplus \langle f, a, \mathcal{E}[a_{\star}], F_s, F_j, \mathsf{eff}', \mathsf{ctx}' \rangle, \mu[a_{\star} \mapsto []], \tau, \sigma \rangle$

with $a_{\star}$ fresh

$\mathsf{act}_{\star} = \langle a_{\star}, \bullet, \langle b_{\star}, \overline{v} \rangle, \bullet \rangle$

$$\begin{cases} \text{if } \mathsf{ctx}^{?} = \bullet: & \mathsf{ctx}' = \bullet & \text{(outside transaction)} \\ & \mathsf{eff}' = \mathsf{eff} \mathrel{+}= \langle \mathsf{act}_{\star}, \bullet \rangle & \\ \text{if } \mathsf{ctx}^{?} = \langle \mathsf{n}, \overleftarrow{\sigma}, \delta, \mathsf{eff}_{tx} \rangle: & \mathsf{ctx}' = \langle \mathsf{n}, \overleftarrow{\sigma}, \delta, \mathsf{eff}_{tx} \mathrel{+}= \langle \mathsf{act}_{\star}, \bullet \rangle \rangle & \text{(in transaction)} \\ & \mathsf{eff}' = \mathsf{eff} & \end{cases}$$

$\underline{\text{become}|_{c}}$

$\langle A, T \uplus \langle f, a, \mathcal{E}[\mathtt{become}\ b_{\star}\ \overline{v}], F_s, F_j, \mathsf{eff}, \mathsf{ctx}^{?} \rangle, \mu, \tau, \sigma \rangle$

$\rightarrow \langle A, T \uplus \langle f, a, \mathcal{E}[\mathtt{nil}], F_s, F_j, \mathsf{eff}', \mathsf{ctx}' \rangle, \mu, \tau, \sigma \rangle$

$$\text{with} \begin{cases} \text{if } \mathsf{ctx}^{?} = \bullet: & \mathsf{ctx}' = \bullet & \text{(outside transaction)} \\ & \mathsf{eff}' = \mathsf{eff} \mathrel{+}= \langle \varnothing, \langle b_{\star}, \overline{v} \rangle \rangle & \\ \text{if } \mathsf{ctx}^{?} = \langle \mathsf{n}, \overleftarrow{\sigma}, \delta, \mathsf{eff}_{tx} \rangle: & \mathsf{ctx}' = \langle \mathsf{n}, \overleftarrow{\sigma}, \delta, \mathsf{eff}_{tx} \mathrel{+}= \langle \varnothing, \langle b_{\star}, \overline{v} \rangle \rangle \rangle & \text{(in transaction)} \\ & \mathsf{eff}' = \mathsf{eff} & \end{cases}$$

$\underline{\text{send}|_{c}}$

$\langle A \uplus \mathsf{act}, T \uplus \langle f, a, \mathcal{E}[\mathtt{send}\ a_{to}\ \overline{v}], F_s, F_j, \mathsf{eff}, \mathsf{ctx}^{?} \rangle, \mu[a_{to} \mapsto \overline{\mathsf{msg}}], \tau, \sigma \rangle$

$\rightarrow \langle A \uplus \mathsf{act}, T \uplus \langle f, a, \mathcal{E}[\mathtt{nil}], F_s, F_j, \mathsf{eff}', \mathsf{ctx}' \rangle, \mu[a_{to} \mapsto \overline{\mathsf{msg}} \cdot \mathsf{msg}], \tau, \sigma \rangle$

where $\mathsf{act} = \langle a, f_{root}, \mathsf{beh}, \mathsf{n}^{?}_{dep} \rangle$

with $\mathsf{msg} = \langle a, a_{to}, \overline{v}, \mathsf{n}^{?}_{msg} \rangle$

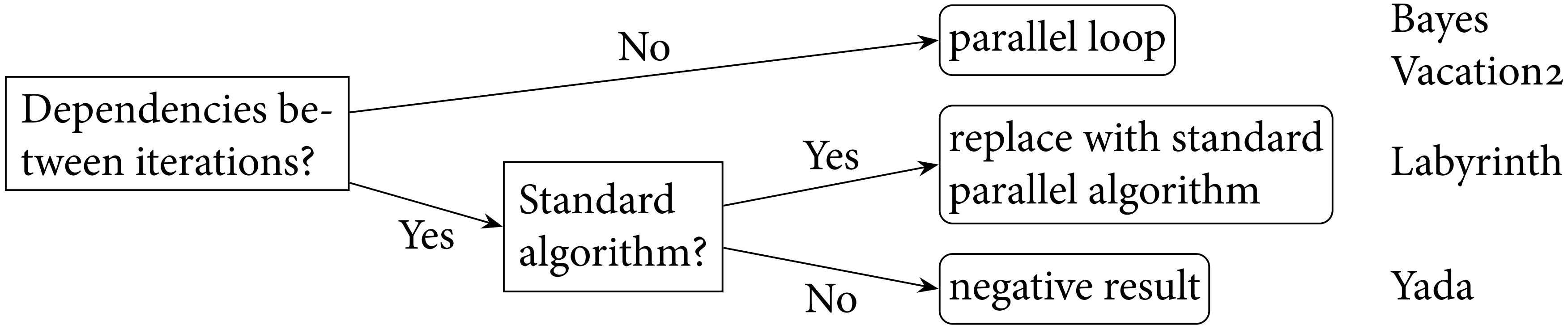
$$\mathsf{n}^{?}_{msg} = \begin{cases} \mathsf{n}_{tx} & \text{if } \mathsf{ctx}^{?} = \langle \mathsf{n}_{tx}, \overleftarrow{\sigma}, \delta, \mathsf{eff}_{tx} \rangle & \text{(in transaction)} \\ \mathsf{n}^{?}_{dep} & \text{if } \mathsf{ctx}^{?} = \bullet \text{ and } \mathsf{n}^{?}_{dep} \neq \bullet & \text{(in tentative turn)} \\ \bullet & \text{otherwise} & \text{(definitive)} \end{cases}$$

Evaluation approach

① selection of benchmarks

Application	Transaction length (mean # of instructions per tx)	Average time in transaction
Labyrinth	219,571	100%
Bayes	60,584	83%
Yada	9,795	100%
Vacation-high	3,223	86%
Genome	1,717	97%
Intruder	330	33%
Kmeans-high	117	7%
SSCA2	50	17%

② parallelization



③ evaluation criteria

performance: speed-up

developer effort: lines changed + qualitative assessment

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

Evaluation results

	Speed-up original		Speed-up Chocola	Lines of code added	
Labyrinth	1.3	↗	2.3	+11%	8 cores
Bayes	2.8	↗	3.5	+1	
Vacation2	2.6	↗	33.2	+8%	64 cores
Yada	futures/actors not applicable				

Better performance for little effort

https://github.com/jswalens/{labyrinth,bayes,yada,vacation2}

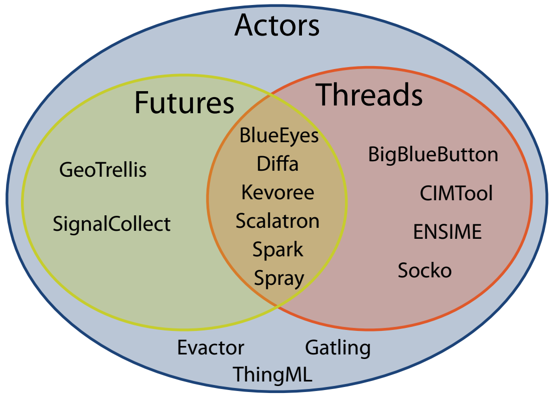
Limitations & Future work

- Formal proofs of guarantees
- Applicability & more benchmarks
- Comparison of implementation techniques

Conclusion

Concurrency models are combined

Naive combinations violate guarantees



We studied the combinations of futures, transactions, and actors

→in↓	Future	Transaction	Actor
Future	Nested futures Det	Parallel transactions ~~Det~~ Iso Pro	Communication in future ~~Det~~ ITP DLF
Transaction	Parallelism in transaction ~~Det~~ → ITD Iso Pro	Nested transactions Iso Pro	Communication in transaction Iso Pro ~~ITP~~ → LLRF DLF
Actor	Parallelism in actor Det ITP DLF	Shared memory in actor Iso Pro ~~ITP~~ → LLRF DLF	Actors ITP DLF

⇒ Chocola: maintain guarantees wherever possible

http://chocola.soft.brussels