CS 420 Advanced Programming Languages
Fall Semester, 2022
Doc 17 Some Concurrency
Oct 27, 2022

Basic Terms

Asynchronous

- Promise, future, delay

State management, particularly in the face of concurrency

- ref, var, watchers, validators
- Transactions

Communicating Sequential Processes

- agents, core.async

Sample Motivation

(slurp "http://www.eli.sdsu.edu/")

Returns the page at the url

(let [page (slurp "http://www.eli.sdsu.edu/")]
  (display-page-in-GUI page))

What happens if network is slow?

Simple Way to Use threads - future

(future expression)

Runs expression in a separate thread

deref & @ do the same thing

Blocks until thread is done

Returns result of thread

Caches the result

Simple Way to Use threads - future

```
(let [result (future (println "this prints once")
                     (+ 1 1))]
  (println "this prints first")
  (println "this prints third")
  (println "deref: " (deref result))
  (println "deref: " (deref result))
  (println "@: " @result))
```

```
this prints first
this prints once
this prints third
deref:  2
deref:  2
@:  2
```

future

- Runs in a separate thread

deref & @ do the same thing

- Blocks until thread is done
- Returns result of thread
- Caches the result

```
(def web-site-urls
  ["http://www.eli.sdsu.edu/"
   "http://www.google.com/"
   "http://www.yahoo.com/"
   "http://www.bing.com/"
   "http://www.wikipedia.org/"])

(def web-sites-futures (map #(future (slurp %)) web-site-urls))

(def pages (map deref web-sites))
```

Delay

Suspends execution of code until delay is dereferenced

Caches result

Second time dereferenced returns cached result

Thread safe

```
(def wait (delay (println "do it now") (+ 1 2)))

@wait          prints "do it now", returns 3
@wait          returns 3
```

realized?

Returns true if a value has been produced for a promise, delay, future or lazy sequence.

```
(def wait (delay (println "do it now") (+ 1 2)))

(realized? wait)   false
@wait              prints "do it now", returns 3
(realized? wait)   true
@wait              returns 3
```

Example - Notify User of Uploads Once

```
(def gimli-headshots ["serious.jpg" "fun.jpg" "playful.jpg"])

(defn email-user
  [email-address]
  (println "Sending headshot notification to" email-address))

(defn upload-document
  "Needs to be implemented"
  [headshot]
  true)

(let [notify (delay (email-user "and-my-axe@gmail.com"))]
  (doseq [headshot gimli-headshots]
    (future (upload-document headshot)
            (force notify))))
```

Example - Proxy for Expensive Operation

```
(defn fetch-page
  [url]
  {:url url
   :contents (delay (slurp url))})
```

```
(def result (fetch-page "http://www.eli.sdsu.edu/index.html"))
(:contents result)                    #<Delay@2fcc470c: :pending>
(realized? (:contents result))        false
@(:contents result)                   "<!DOCTYPE html>\n<html lang=\"en\">\n …"
```

deref with Timeout

```
(deref (future (Thread/sleep 5000) :done!)
       1000
       :impatient!)
  ;= :impatient!
```

Promise

one-time, single value pipe

```
(def p (promise))
(realized? p)          false
(deliver p 42)         #<core$promise$reify__1707@3f0ba812: 42>
(realized? p)          true
@p                     42
(deliver p 50)         nil
@p                     42
```

Promise

Simple way to send data back from thread

Find a Web Page with a Term

```
(defn find-page-with-term
  [urls term]
  (let [page-with-term (promise)
        search-page #(future (let [page (slurp %)]
                               (when (clojure.string/includes? page term)
                                 (deliver page-with-term %))))]
        (mapv search-page urls)
        page-with-term))
```

```
(def web-site-urls
  ["http://www.eli.sdsu.edu/"
   "https://www.google.com/"
   "https://www.yahoo.com/"
   "https://www.bing.com/"
   "https://www.wikipedia.org/"])
```

```
(let [page (find-page-with-term web-site-urls "Web")]
  (println (realized? page)))
```

```
false
```

```
(def web-site-urls
  ["http://www.eli.sdsu.edu/"
   "https://www.google.com/"
   "https://www.yahoo.com/"
   "https://www.bing.com/"
   "https://www.wikipedia.org/"])
```

References

Time, State, Identity

Time

Relative moments when an event occurs

State

Snapshot of entity’s properties at a moment in time

Identity

Logical entity identified by a common stream of states occurring over time

State & Identity

{:name “Sarah”
 :age 10
 :wears-glasses false}

{:name “Sarah”
 :age 11
 :wears-glasses false}

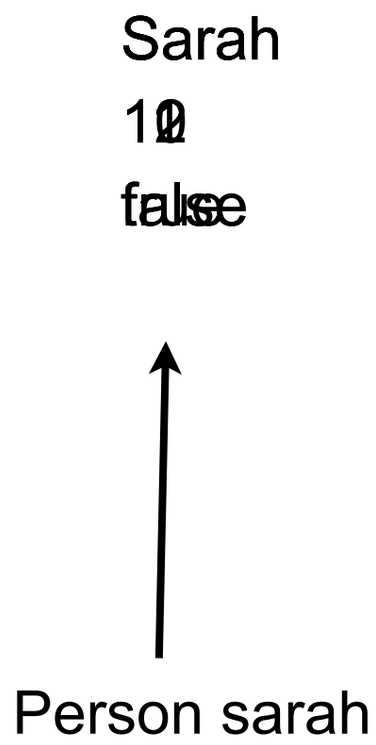
{:name “Sarah”
 :age 12
 :wears-glasses true}

Java

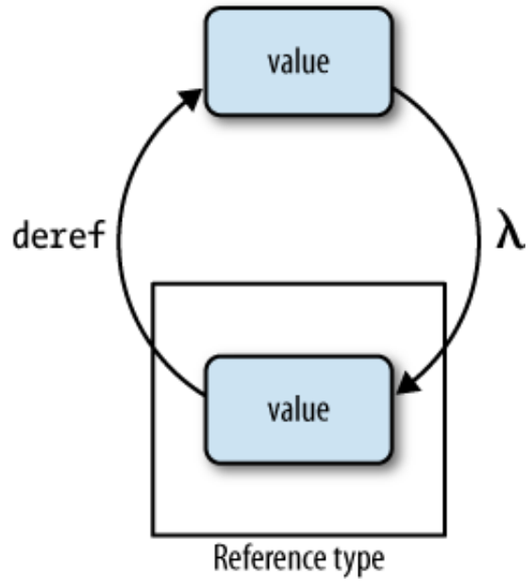
```
class Person {
        public String name;
        public int age;
        public boolean wearsGlasses;

        public Person (String name, int age, boolean wearsGlasses) {
          this.name = name;
          this.age = age;
            this.wearsGlasses = wearsGlasses;
        }
}
```

State & Identity

Complexted in Java



Reference Type Basics



var, ref, atom, agent

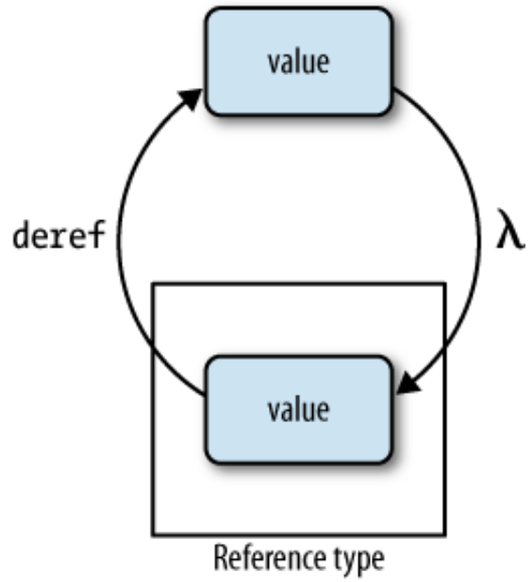
All are pointers

Can change pointer to point to different data

Dereferencing will never block

Each type as different way of setting/changing its value

Reference Type Basics



var, ref, atom, agent

Each type

Can have meta data

Can have watches (observers)

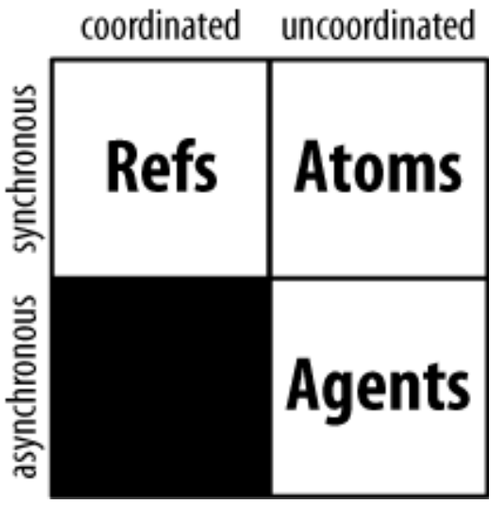
Call specified function when value is change

Can have validator

Enforce constraints on values pointer can point to

Features of each Type

	Ref	Agent	Atom	Var
Coordinated	X			
Asynchronous		X		
Retriable	X		X	
Thread-local				X



Synchronous - block until operation completes

Asynchronous - Non blocking, operation can compete on separate thread

Coordinated - Supports transactions

Thread-local - Changes made are local to current thread

Creating & Referencing Each Type

```
(def ref-example (ref 10))
@ref-example
(deref ref-example)

(def agent-example (agent 10))
@agent-example
(deref agent-example)

(def atom-example (atom 10))
@atom-example
(deref atom-example)

(def var-example 10)
var-example
```

Note the difference

Watches

```
(defn cat-watch
  [key pointer old new]
  (println "Watcher" key pointer old new))

(def cat 4)

(add-watch (var cat) :cat cat-watch)

(def cat 10)

(remove-watch (var cat) :cat)

(def cat 20)
```

Output in Console

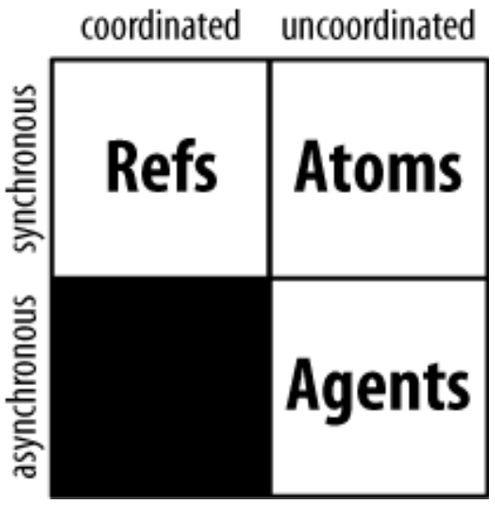
Watcher :cat #'user/ca

Validator

```
(def cat 4)

(set-validator! (var cat) #(> 10 %))

(def cat 9)

(def cat 20)                          ;;exception
```

Features of each Type

	Ref	Agent	Atom	Var
Coordinated	X			
Asynchronous		X		
Retriable	X		X	
Thread-local				X



Synchronous - block until operation completes

Asynchronous - Non blocking, operation can compete on separate thread

Coordinated - Supports transactions

Thread-local - Changes made are local to current thread

Atoms

Changes are
- Synchronous
- Uncoordinated
- Atomic

Synchronous
- Code waits until change done

Uncoordinated
- No transaction support

Atomic
- Threads only see old or new value
- Never see partially changed data

Atoms - Methods for change

swap!

Applies function to current state for new state

reset!

Changes state to given value

compare-and-set!

Changes state to given value only if current value is what you think it is

reset!

```
(def a (atom 0))

@a                  0

(reset! a 5)        5

@a                  5
```

swap!

```
(def a (atom 0))

@a                  0

(swap! a inc)       1

@a                  1
```

swap!

```
(def sarah (atom {:name "Sarah" :age 10 :wears-glasses? false}))

(swap! sarah update-in [:age] + 3)          {:name "Sarah", :age 13,
                                             :wears-glasses? false}

@sarah                                      {:name "Sarah", :age 13,
                                             :wears-glasses? false}
```

swap! is Atomic

```
(swap! sarah (comp #(update-in % [:age] inc)
                   #(assoc % :wears-glasses? true)))
```

Compound operation on sarah

What happens if other thread reads sarah during swap!

It gets the old value

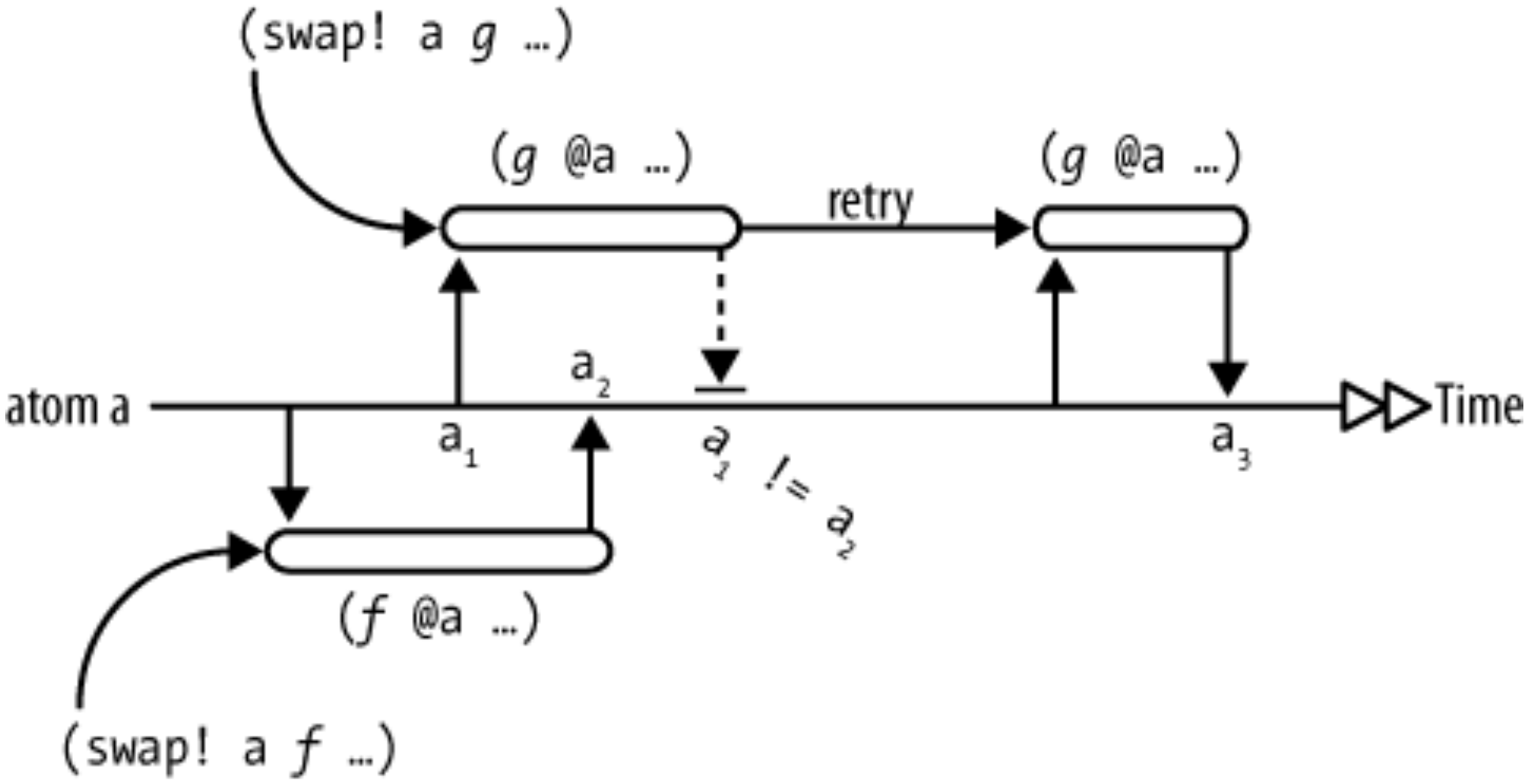
swap! is Atomic

```
(swap! sarah (comp #(update-in % [:age] inc)
                   #(assoc % :wears-glasses? true)))
```

What happens if other thread modifies sarah during swap!

It retries until it can read the new value

Then modifies sarah


(swap! a g ...)
(g @a ...)
retry
(g @a ...)
atom a
a1
a2
a2 != a2
a3
Time
(f @a ...)
(swap! a f ...)

Recall - find-page-with-term

```
(defn find-page-with-term
  [urls term]
  (let [page-with-term (promise)
        search-page #(future (let [page (slurp %)]
                               (when (clojure.string/includes? page term)
                                 (deliver page-with-term %))))]
        (mapv search-page urls)
        page-with-term))
```

Finding all pages containing a Term

Write to a vector when find a page with a term

Need to make sure that only one thread writes at a time

Need a mutex

```
(defn find-page-with-term
  [urls term]
  (let [pages-with-term (atom [ ])
        search-page #(future (let [url %
                                   page (slurp ulr)]
                               (when (clojure.string/includes? page term)
                                 (swap! pages-with-term conj url))))]
    (mapv search-page urls)
    pages-with-term))
```

Ref

Coordinated reference type

Multiple values can be changed

Changes are atomic

No Race conditions

No deadlocks

No manual locks, monitors etc

Software Transactional Memory

Ref changes are done in a transaction

No changes are visible out side transaction until transaction is completed

Exceptions abort the transaction

If

Transaction A and B modify one or more of the same refs

Transaction A starts before B, but ends between B's start and end

Then

Transaction B will retry with the new values of the refs

Starting a Transaction

(dosync form1 form2 ... formN)

Altering a ref

(alter ref fun & args)

Applys the fun to the ref to get new value

(ref-set ref val)

Sets the ref to val

Example

```
(def sam-account (ref 10))
(def pete-account (ref 20))

(set-validator! sam-account #(< 0 %))
(set-validator! pete-account #(< 0 %))

(defn sam-pay-pete
  [amount]
  (dosync
    (alter pete-account + amount)
    (alter sam-account - amount)))
```

```
(sam-pay-pete 8)
@sam-account          2
@pete-account         28
(sam-pay-pete 8)      Exception
@sam-account          2
@pete-account         28
```