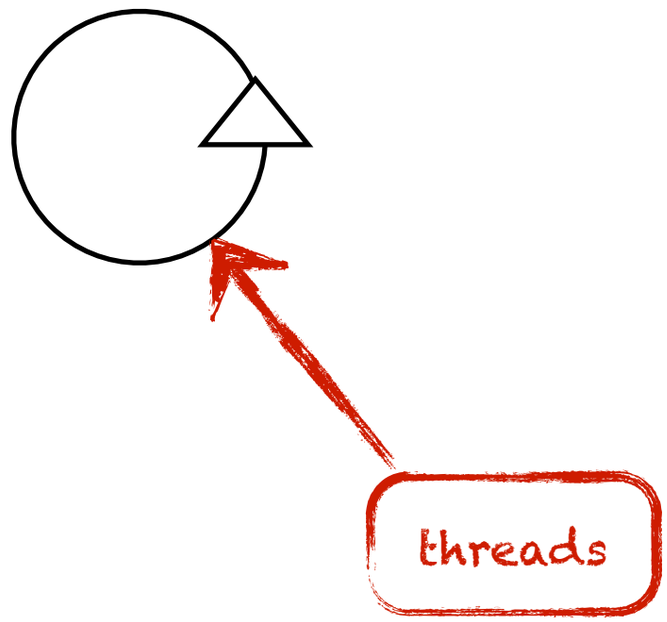
Concurrency in Clojure

@stuarthalloway

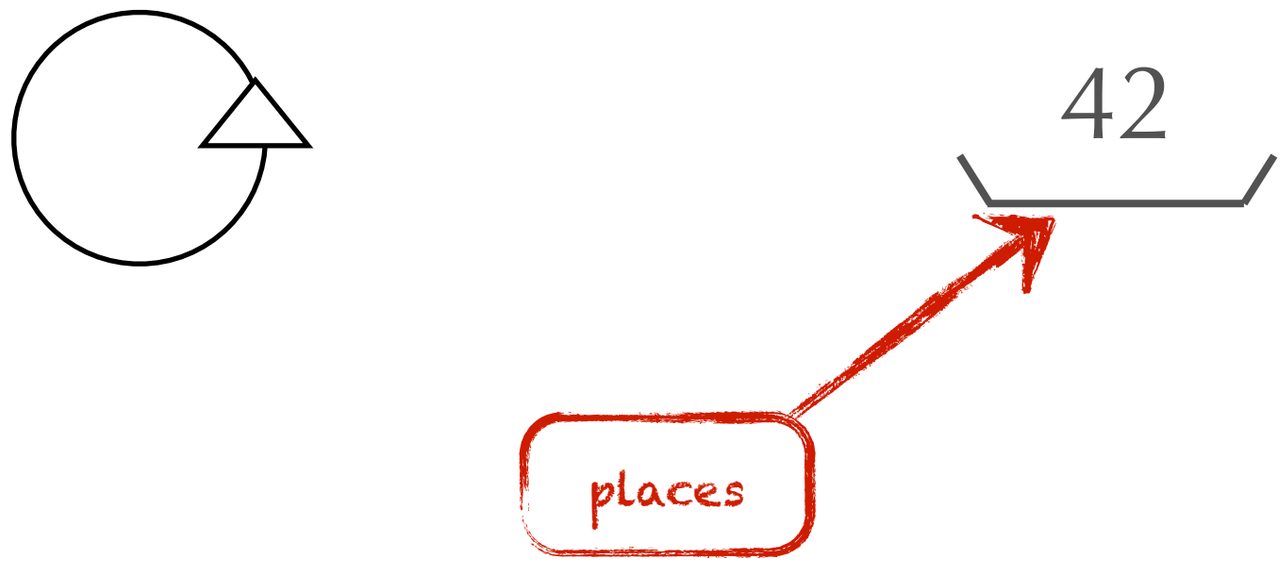
concurrency, coincidence of events or space

parallelism, the execution of operations concurrently by separate parts of a computer

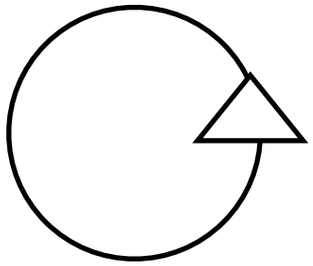
Our Tools

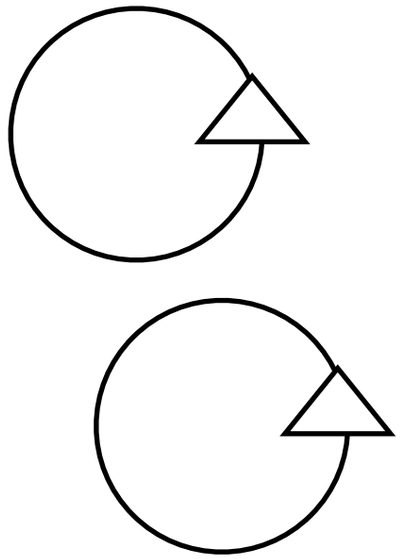


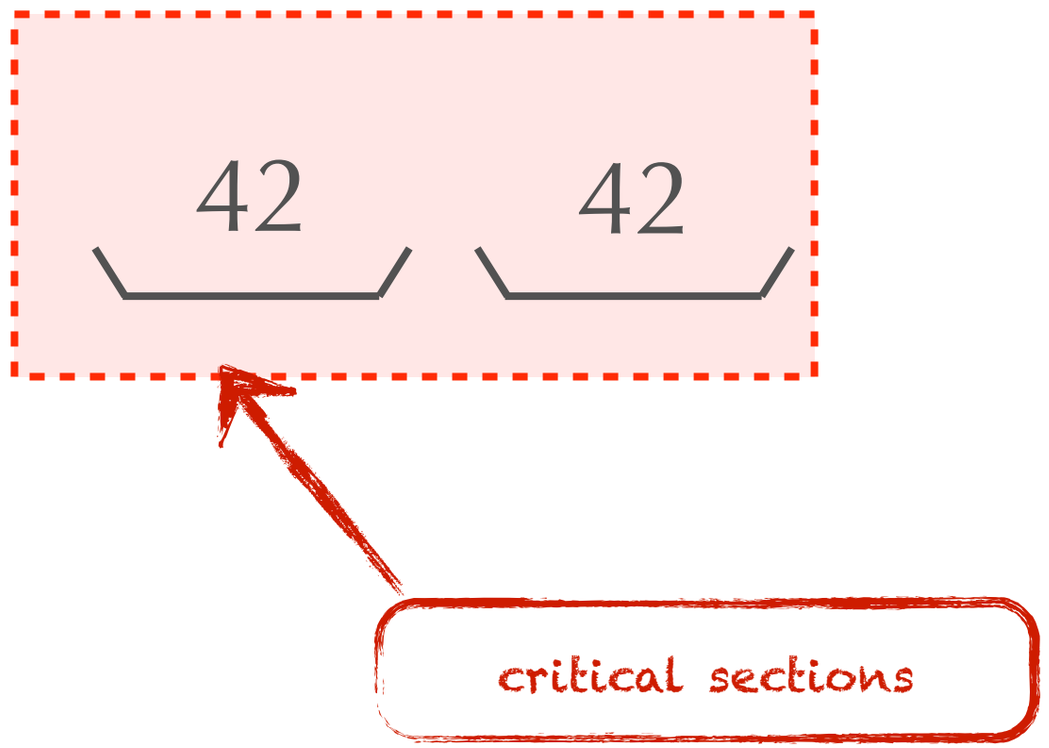
Our Tools



Our Tools







memory, the capacity ... for returning to a previous state when the cause of the transition from that state is removed

record, the fact or condition of having been written down as evidence...
... an authentic or official report

Memory, Records = Places?

Memory is small and expensive

Storage is small and expensive

Machines are *precious, dedicated* resources

Applications are control centers

A Different Approach

New memories use new places

New records use new places

New moments use new places

“In-place” changes encapsulated by constructors

Values

Immutable

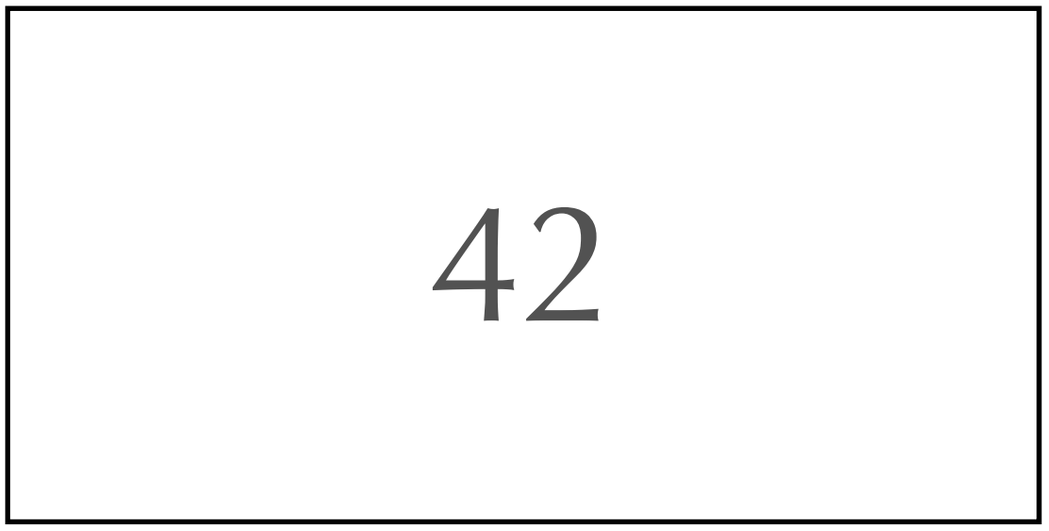
Maybe lazy

Cacheable (forever!)

Can be arbitrarily large

Share structure

What Can Be a Value?



What Can Be a Value?

42

```
{:first-name "Stu",
 :last-name "Halloway"}
```

What Can Be a Value?

42

```
{:first-name "Stu",
 :last-name "Halloway"}
```

What Can Be a Value?

42

```
{:first-name "Stu",
 :last-name "Halloway"}
```

What Can Be a Value?

42

```
{:first-name "Stu",
 :last-name "Halloway"}
```

Anything?

References

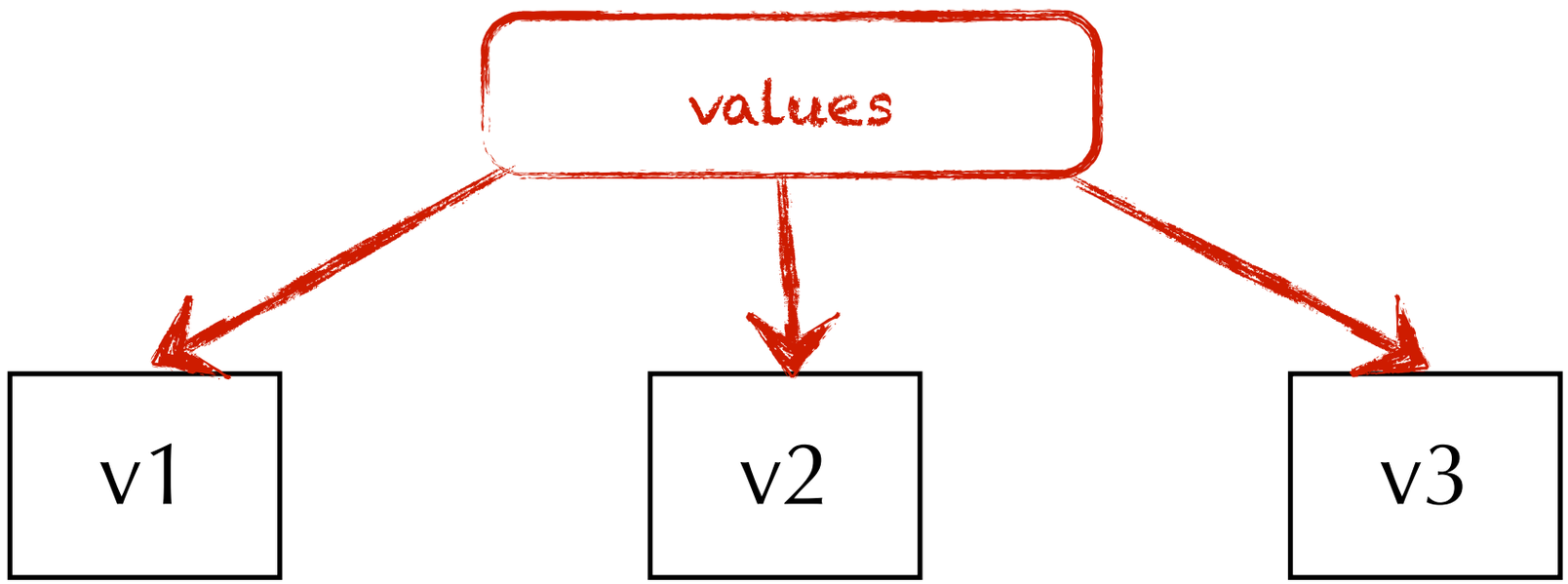
Refer to values (or other references)

Permit atomic, functional succession

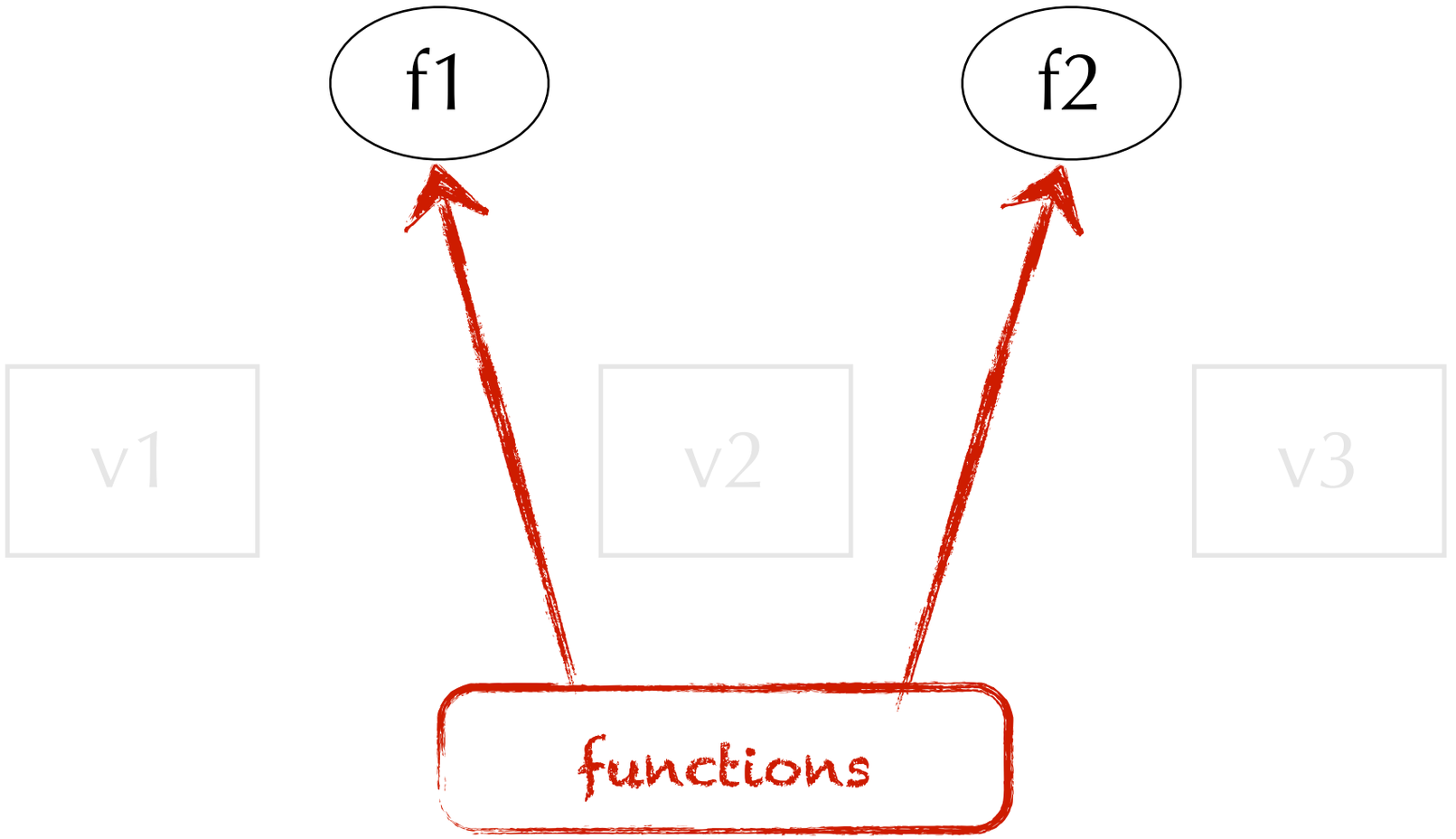
Can model *time* and *identity*

Compatible with a wide variety of update semantics

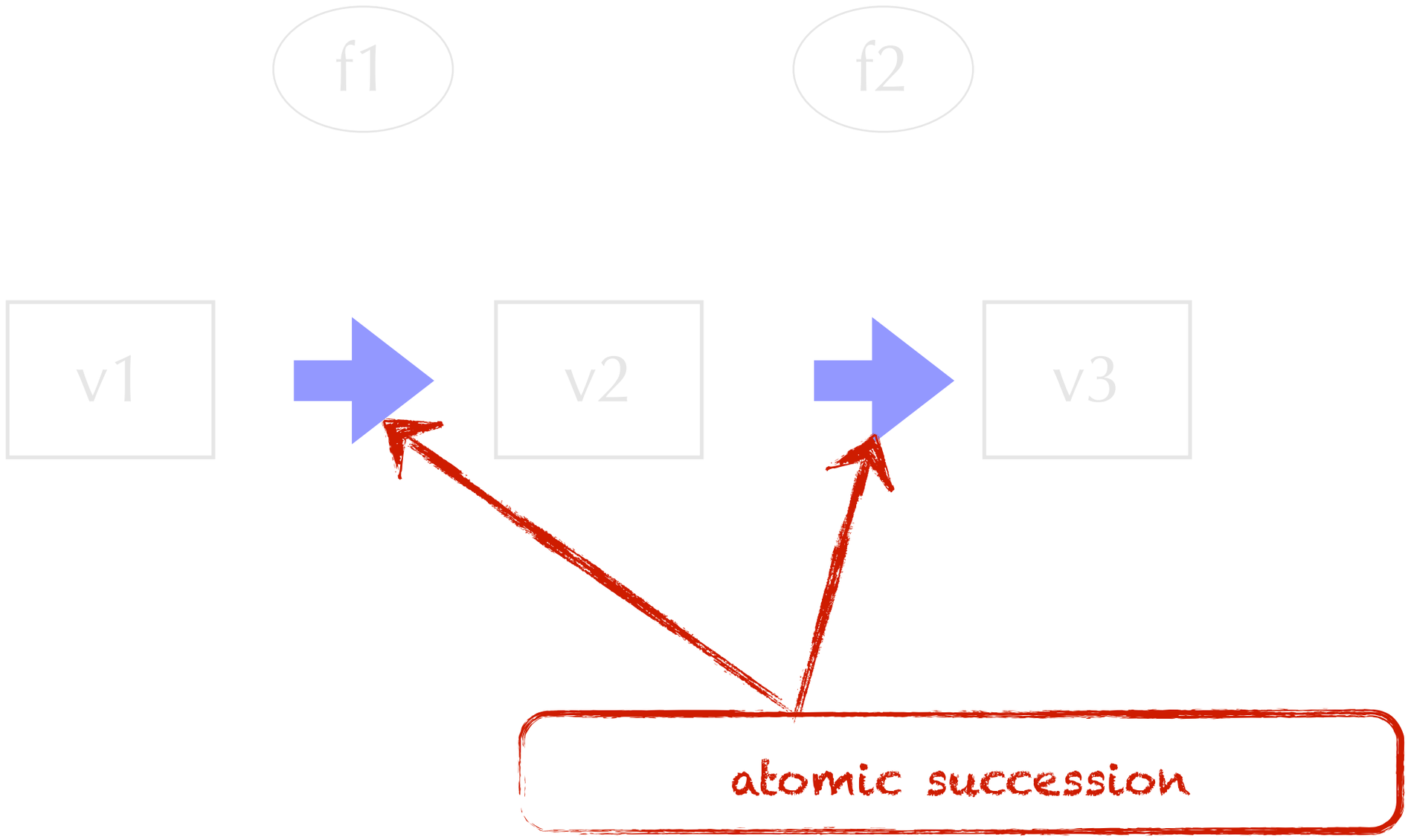
Epochal Time Model



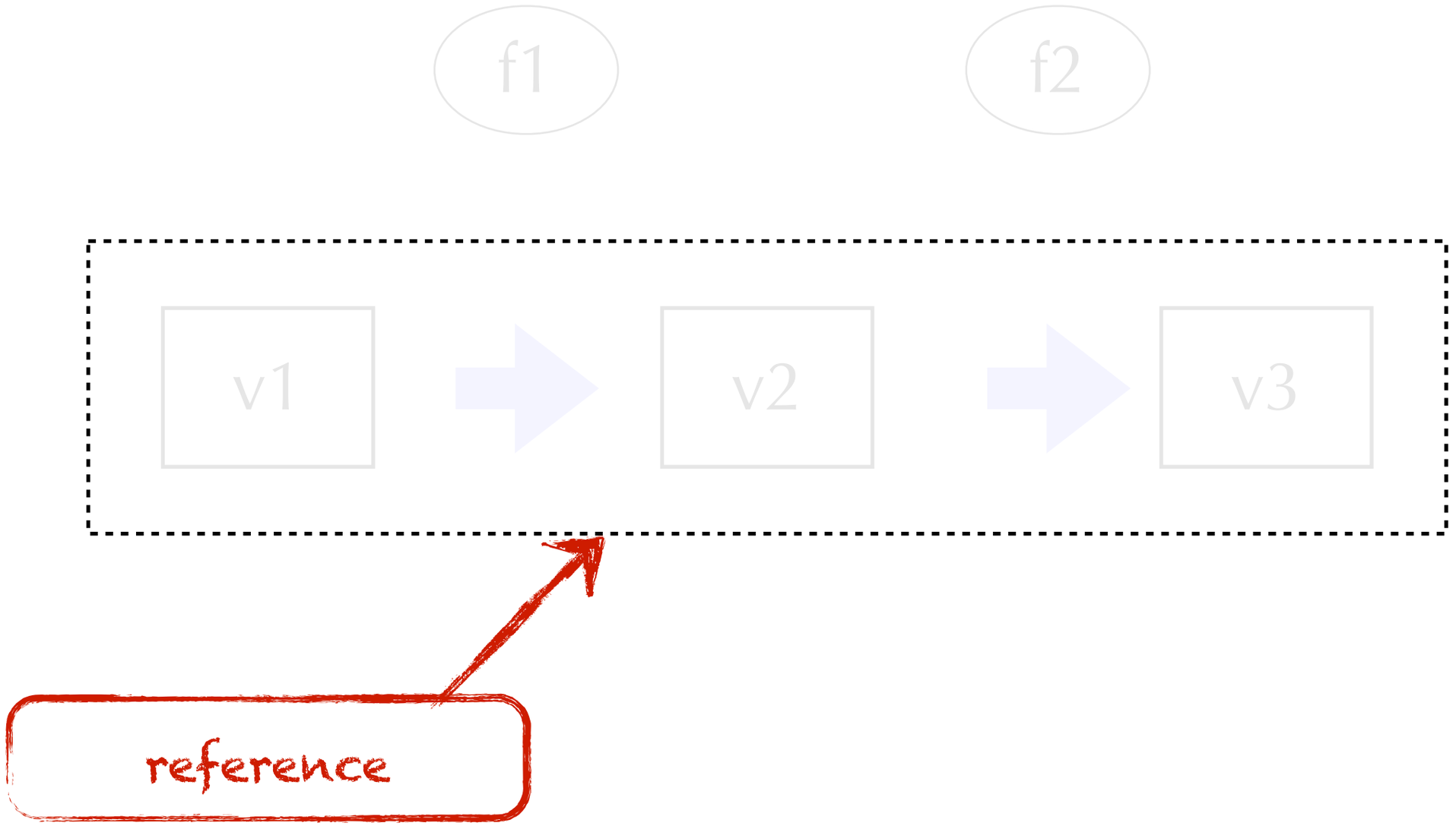
Epochal Time Model



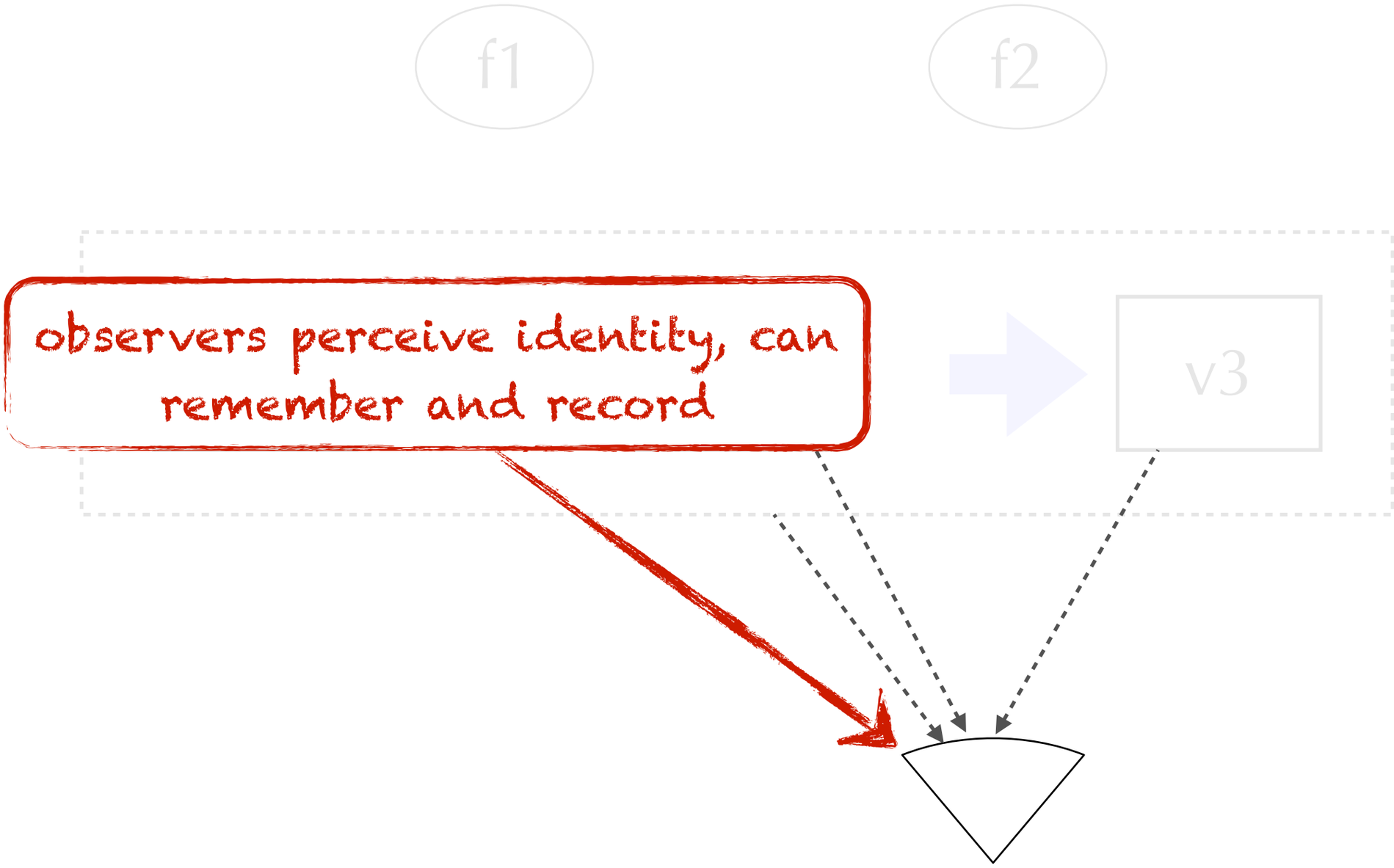
Epochal Time Model



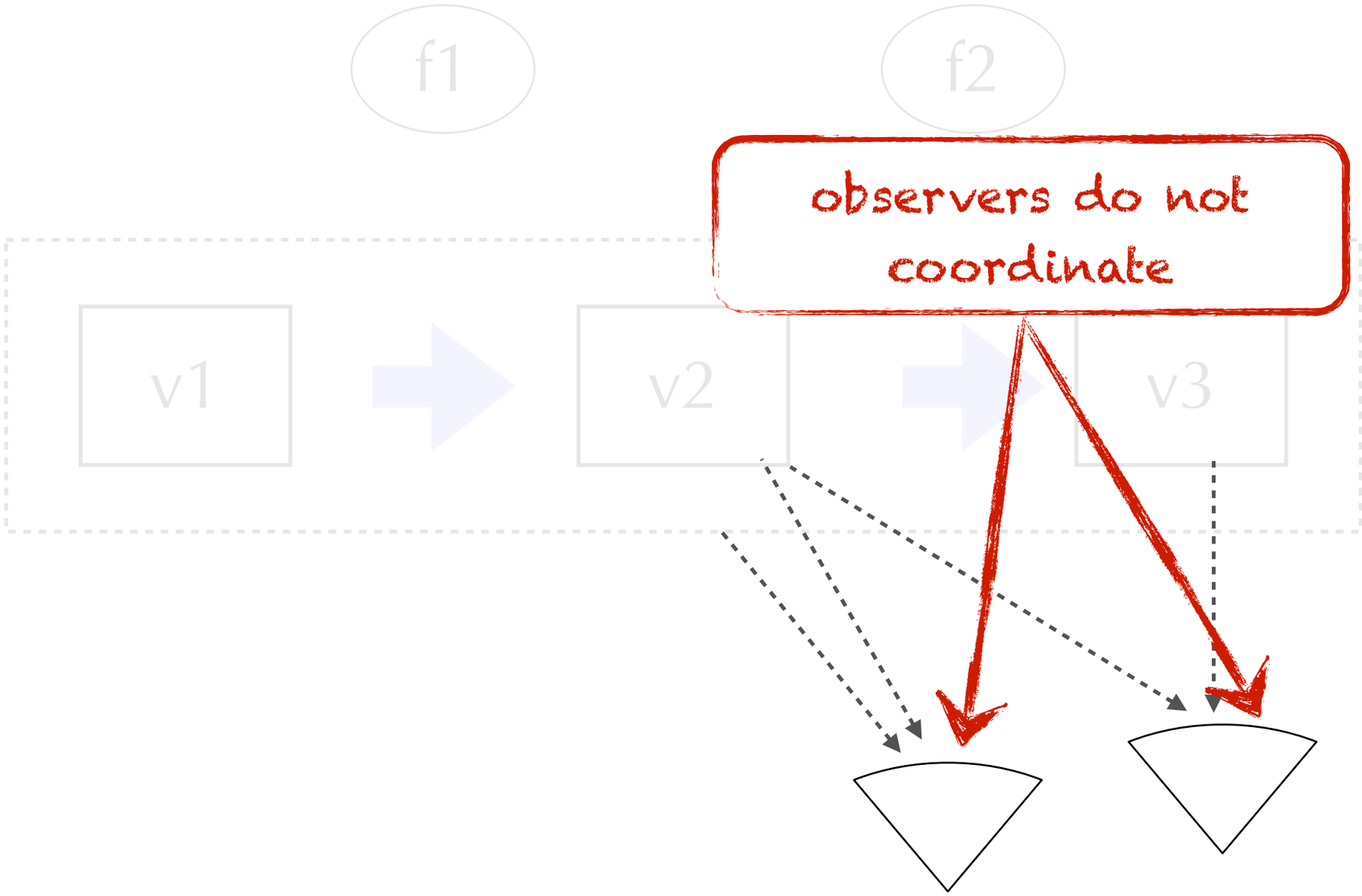
Epochal Time Model



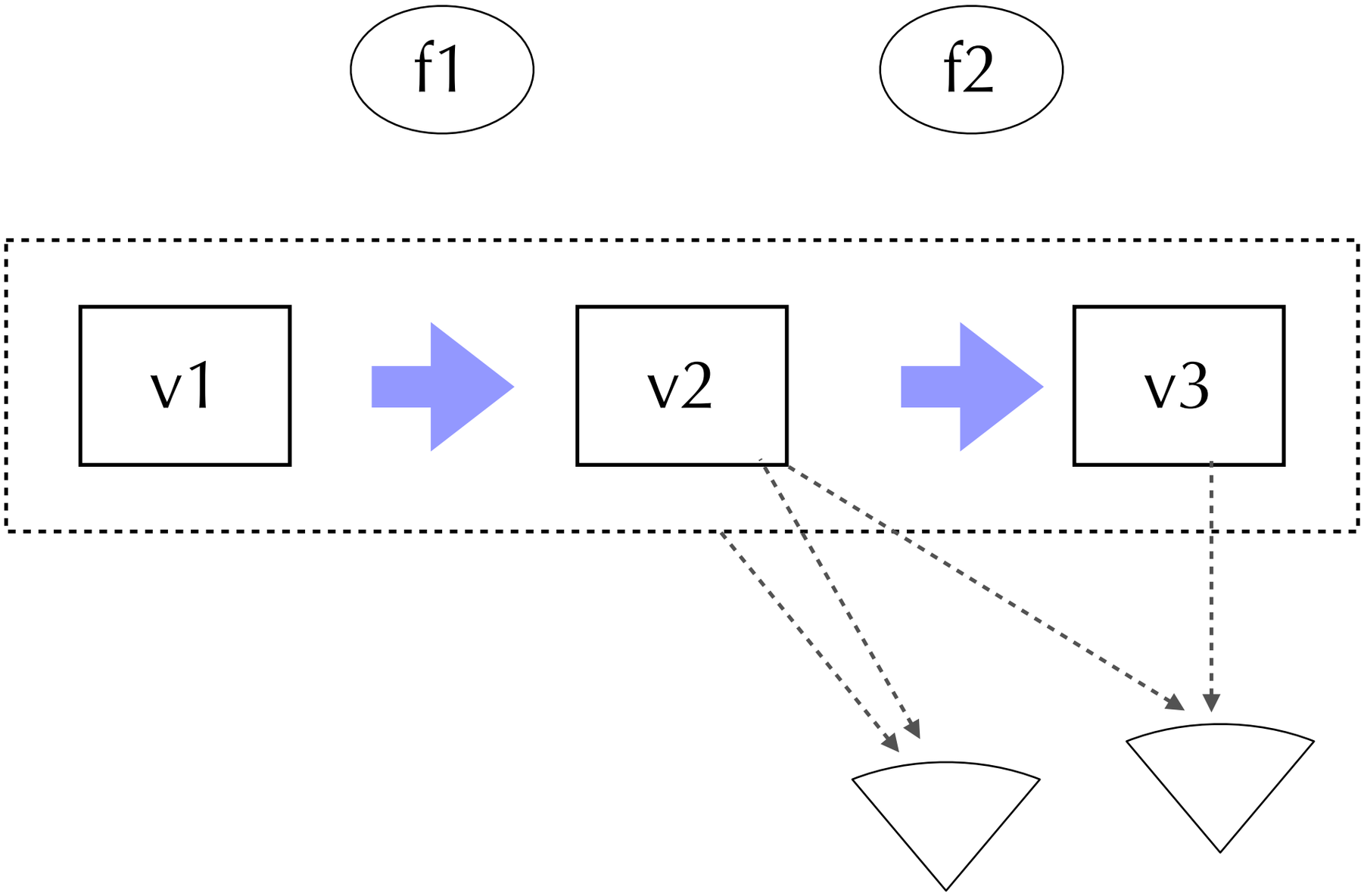
Epochal Time Model



Epochal Time Model



Epochal Time Model



API

```
(def counter  (atom 0))
(swap! counter + 10)
```

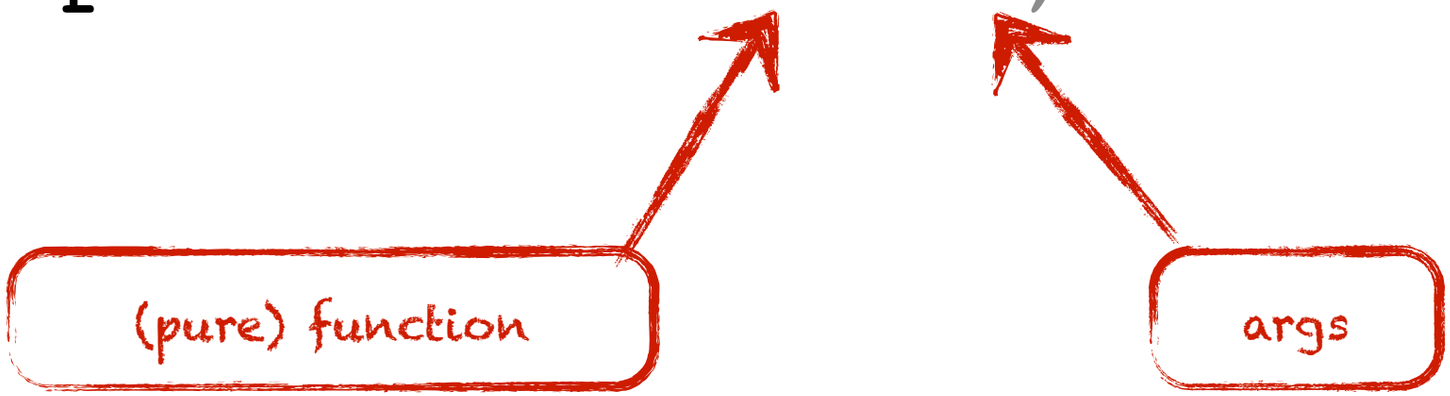
API

reference constructor

```
(def counter   (atom 0))
(swap! counter + 10)
```

API

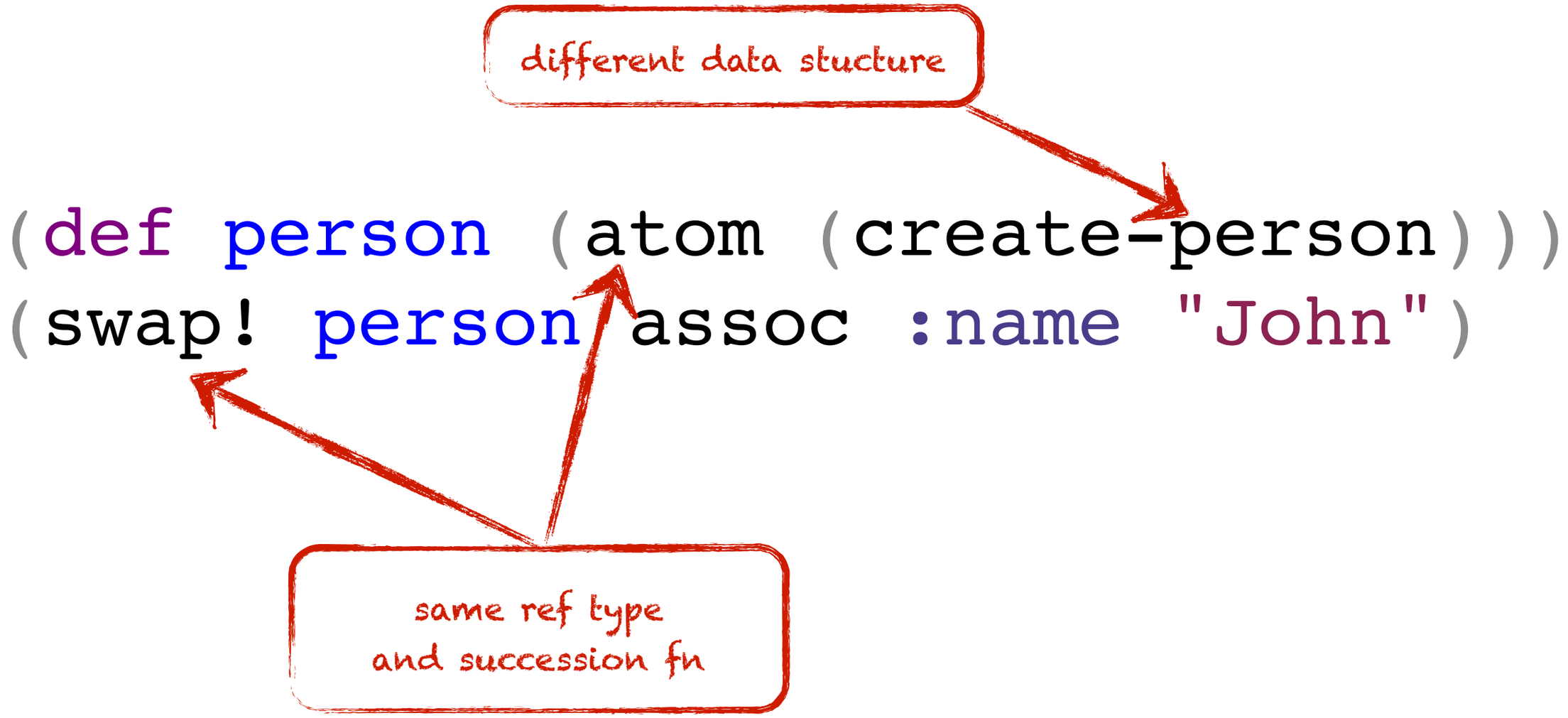
```
(def counter  (atom 0))
(swap! counter + 10)
```



API

```
(def counter  (atom 0))
(swap! counter + 10)
```

Bigger Structures



Varying Semantics

different kind of ref

```
(def number-later (promise))
(deliver number-later 42)
```

different succession

Entire Database

```
(d/create uri)
(def conn (d/connect uri)
(transact conn    data)
```

Entire Database


```
(d/create uri)
(def conn (d/connect uri))
(transact conn    data)
```

Entire Database

```
(d/create uri)
(def conn (d/connect uri))
(transact conn    data)
```

agent →

send	processor-derived pool
send-off	IO-derived pool
send-via	user-specified pool

atom ⇄

compare-and-set!	conditional
reset!	boring
swap!	

connection

transact ⇄	ACID
transact-async →	ACID

ref ⇄

alter	
commute	commutative

var ⇄

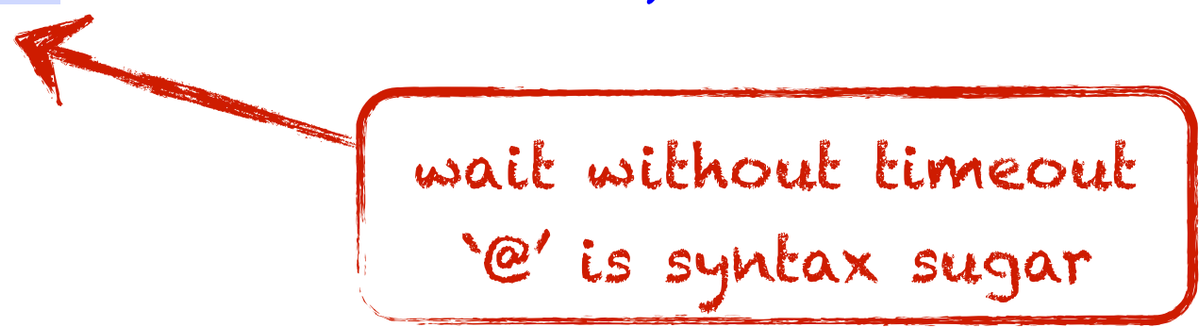
alter-var-root	application config

var binding ⇄

set!	dynamic, binding-local

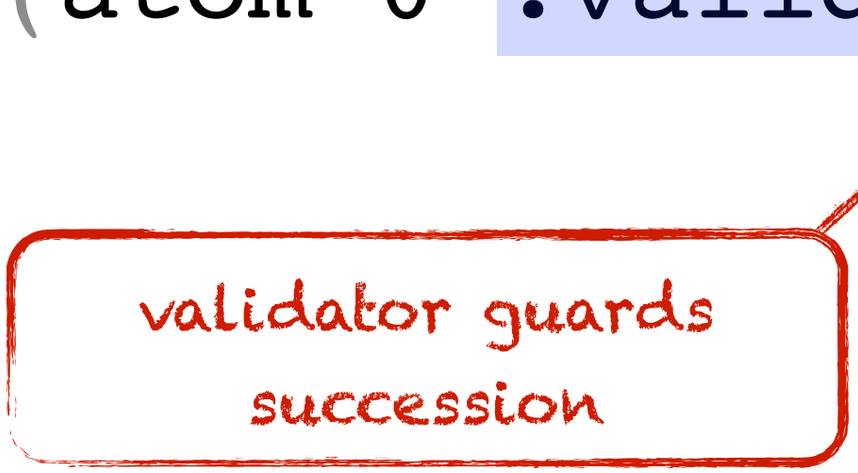
Shared Abilities

```
@some-ref
(deref some-ref)
```



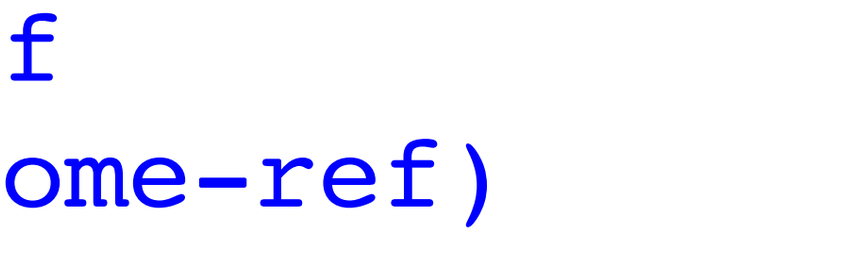
Shared Abilities

```
@some-ref
(deref some-ref)

(atom 0 :validator integer?)
```



Shared Abilities

```
@some-ref
(deref some-ref)

(atom 0 :validator integer?)
(add-watch ref :a/name watcher)
(remove-watch ref :a/name)
```



Pending References

Represent work that may not be done yet

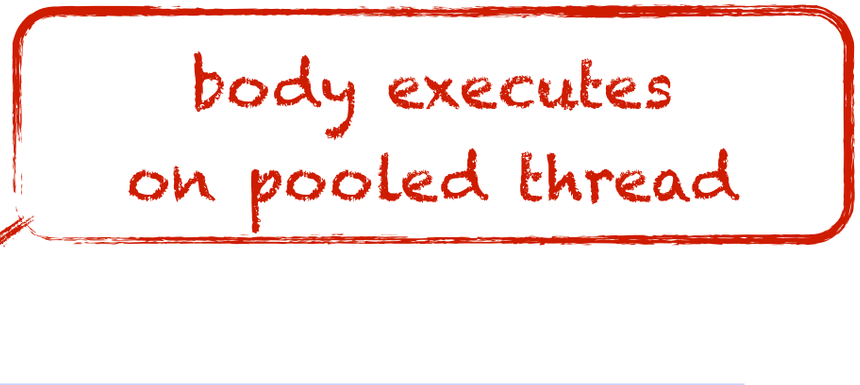
Will only ever refer to one thing

Not identities

Future

```
(def result (some-long-task))

(deref result 1000 or-else)

@result
(deref result)
```

Future




```
(def result (some-long-task))
```

```
(deref result 1000 or-else)

@result
(deref result)
```

Future

```
(def result (some-long-task))

(deref result 1000 or-else)

@result
(deref result)
```

Future

```
(def result (some-long-task))
```

```
(deref result 1000 or-else)
```

```
@result
(deref result)
```

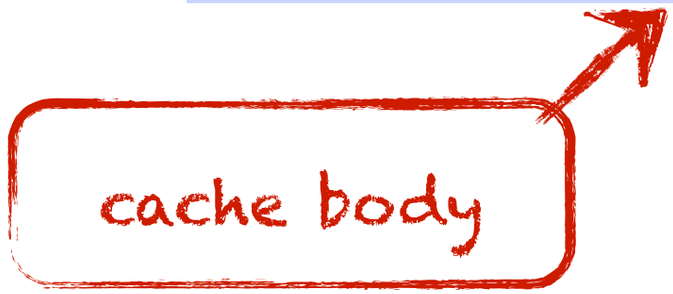
Delay

```
(def result (dont-need-yet))
```

```
@result
(deref result)
```

Delay

```
(def result (dont-need-yet))
```

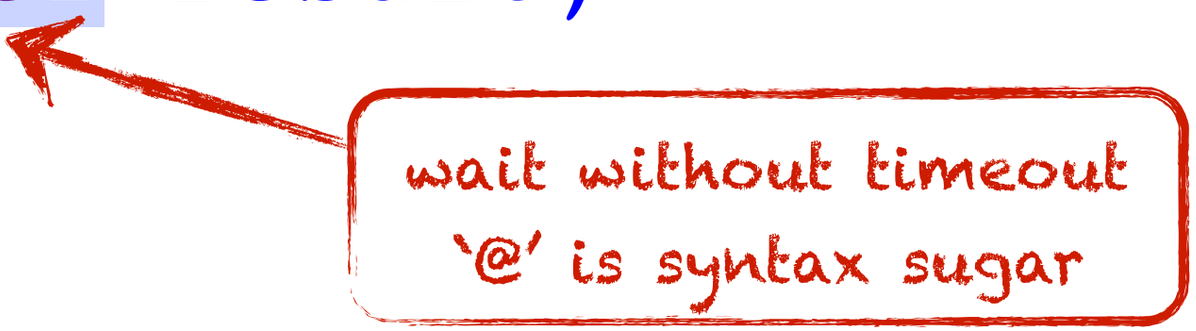



```
@result
(deref result)
```

Delay

```
(def result (dont-need-yet))
```

```
@result
(deref result)
```



Promise

```
(def result (promise))
```

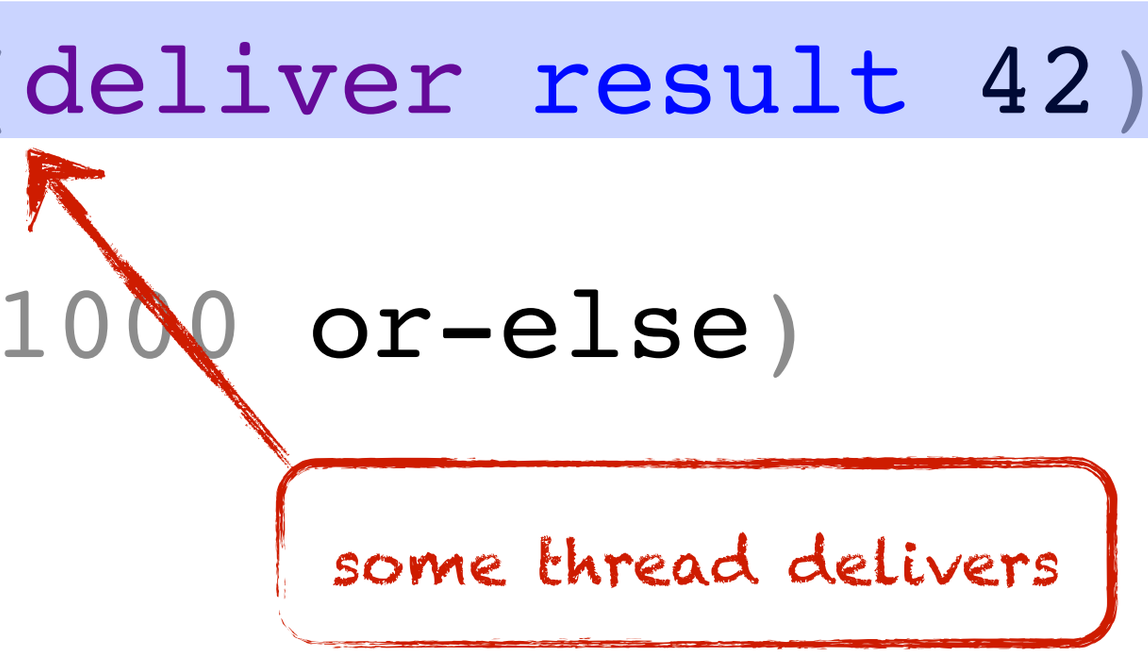
no-arg constructor

```
(deliver result 42)
```

```
(deref result 1000 or-else)

@result
(deref result)
```

Promise

```
(def result (promise))

(deliver result 42)

(deref result 1000 or-else)

@result
(deref result)
```



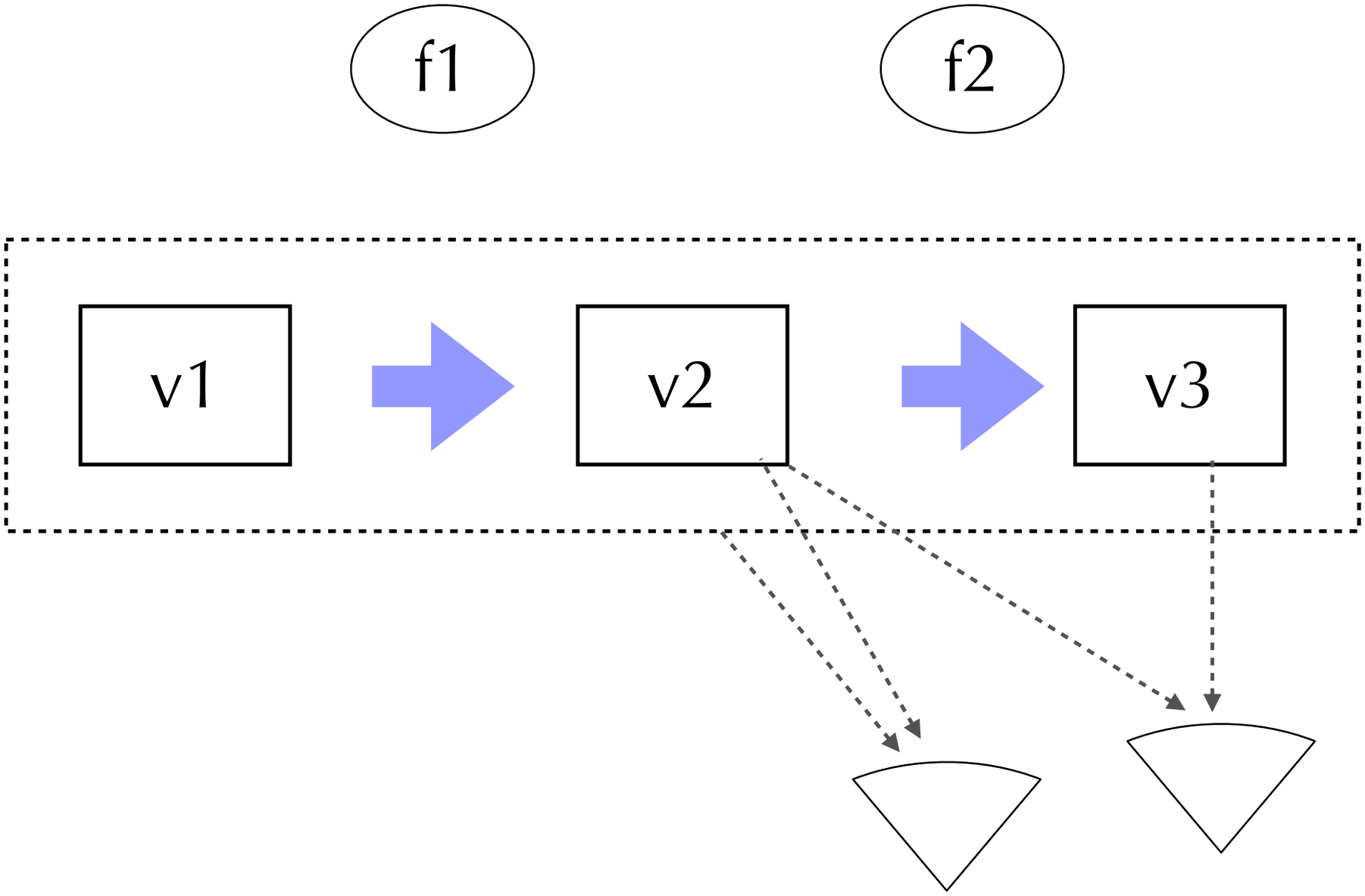
Promise

```
(def result (promise))

                (deliver result 42)

(deref result 1000 or-else)

@result
(deref result)
```

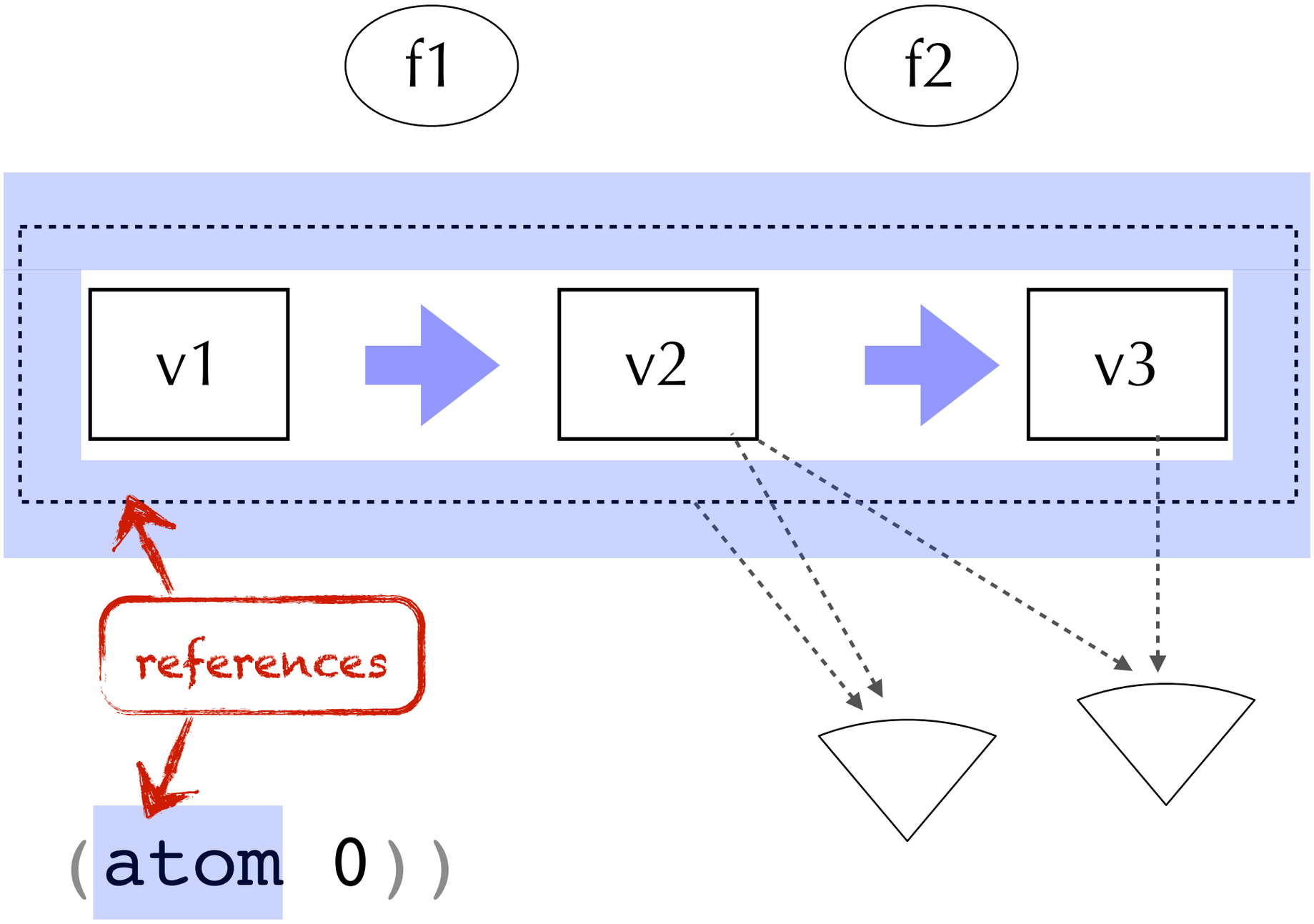
Promise

```
(def result (promise))

                (deliver result 42)

(deref result 1000 or-else)

@result
(deref result)
```

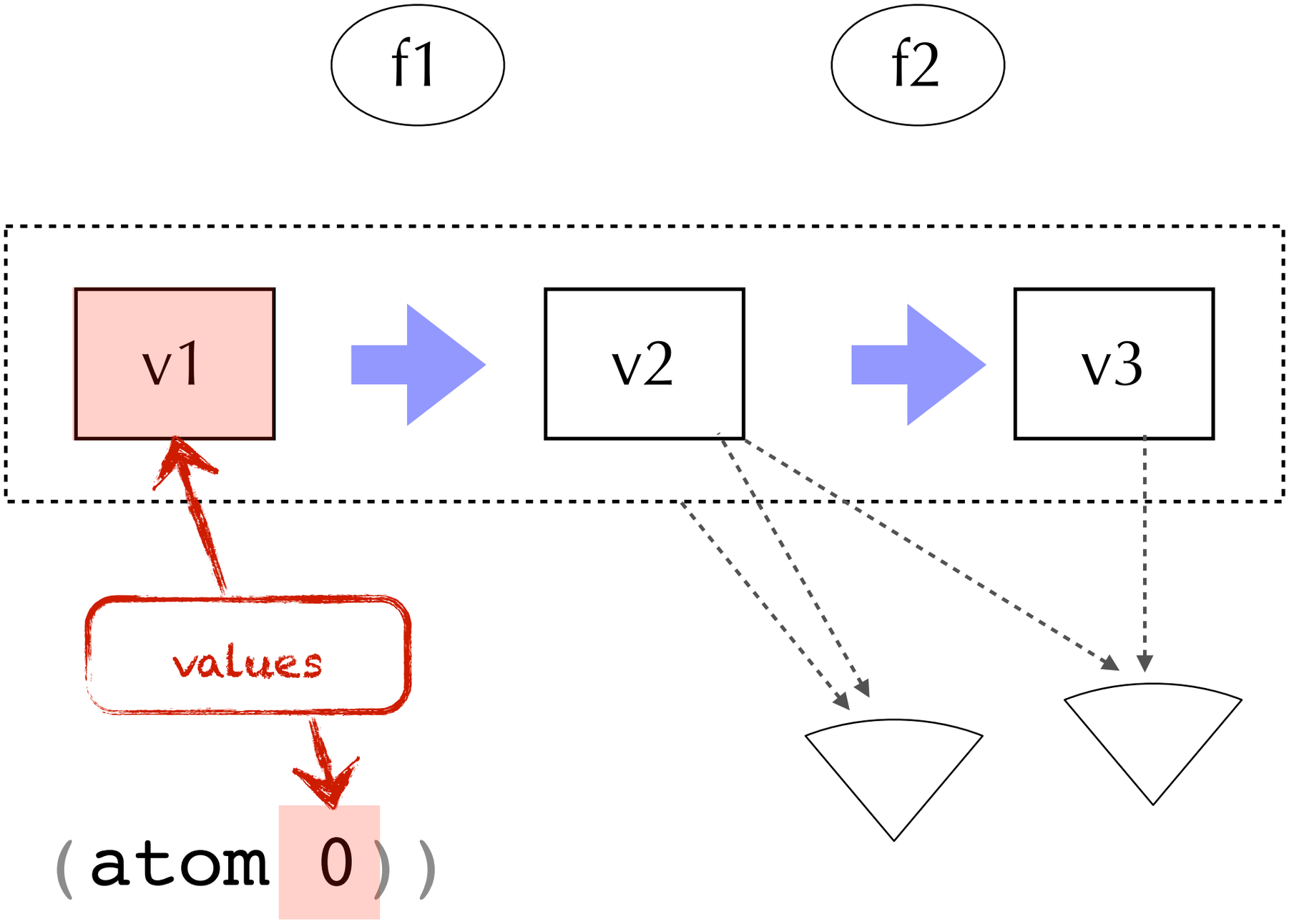
Where Are We?



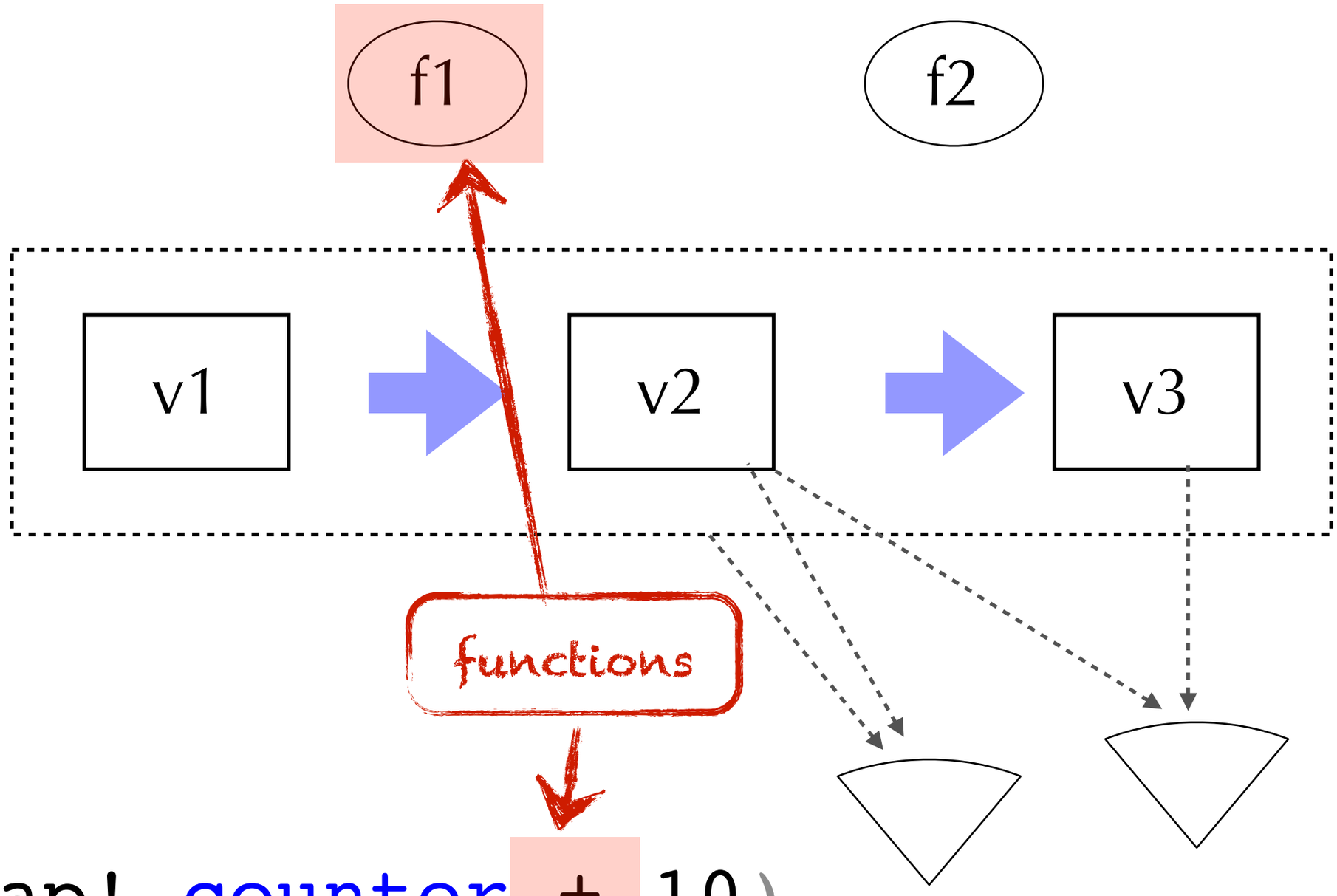
Where Are We?



Where Are We?

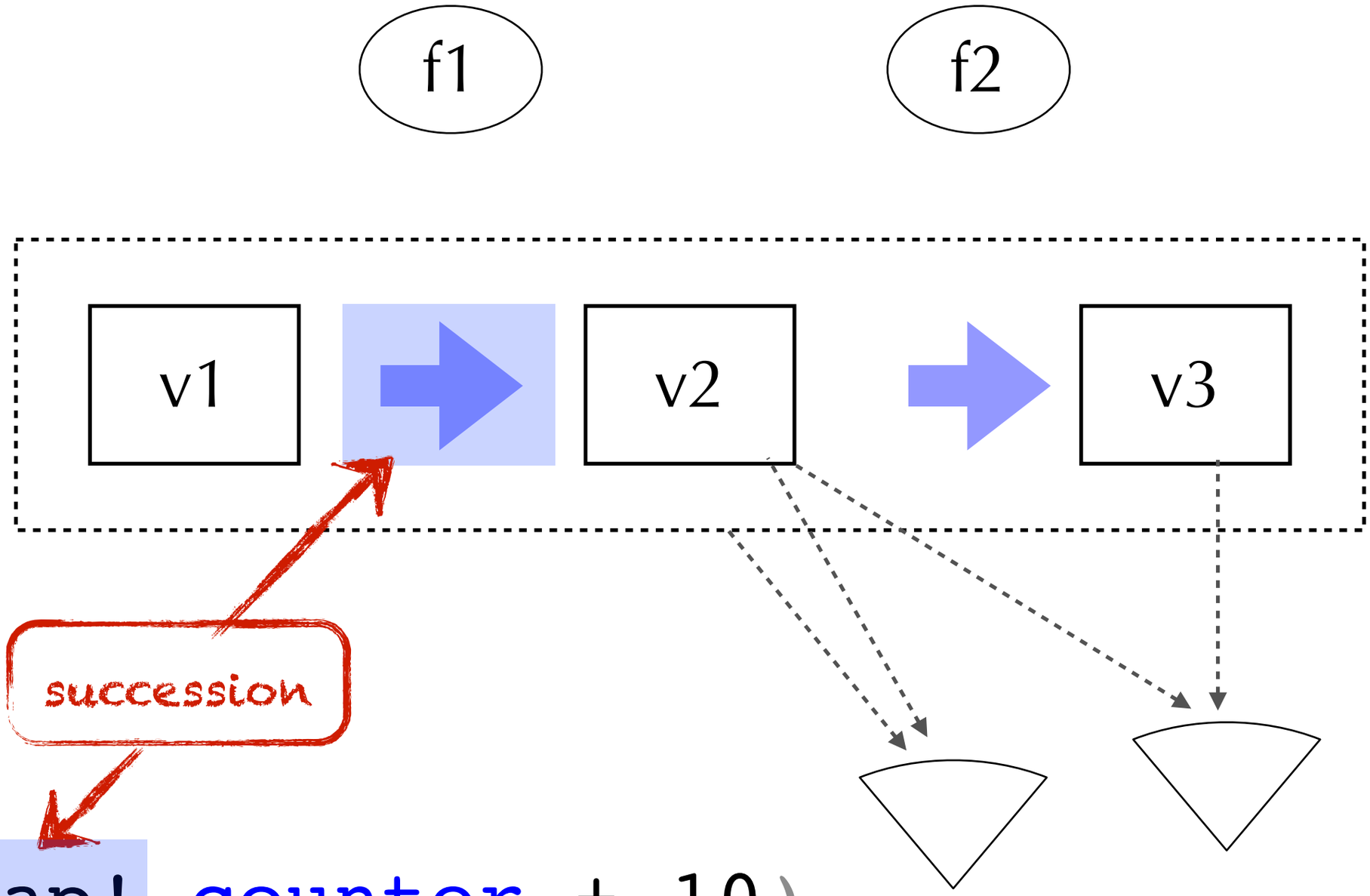


Where Are We?




```
(swap! counter + 10)
```

Where Are We?




```
(swap! counter + 10)
```

References

Definitions are from the Oxford English Dictionary.

Musicbrainz, http://musicbrainz.org/.

Wikipedia, http://www.wikipedia.org/.

@stuarthalloway

https://github.com/stuarthalloway/presentations/wiki