

# Clojure



# Clojure

- ▶ A lisp

# Clojure

- ▶ A lisp
- ▶ That run on the JVM

# Clojure

- ▶ A lisp
- ▶ That run on the JVM
- ▶ Dynamic

# Clojure

- ▶ A lisp
- ▶ That run on the JVM
- ▶ Dynamic
- ▶ Functional programming

# Clojure syntax

List

```
(1 "two" 3)
```



# Clojure syntax

List

```
(1 "two" 3)
```

Vector

```
[1 "two" 3]
```

# Clojure syntax

List

```
(1 "two" 3)
```

Vector

```
[1 "two" 3]
```

Map

```
{:first-key 1  
 :second-key "two"}
```



# Clojure syntax

List

```
(1 "two" 3)
```

Vector

```
[1 "two" 3]
```

Map

```
{:first-key 1  
 :second-key "two"}
```

Function definition

```
(defn my-function [param1 param2]  
  (str param1 param2))
```

# Clojure syntax

List

```
(1 "two" 3)
```

Vector

```
[1 "two" 3]
```

Map

```
{:first-key 1  
 :second-key "two"}
```

Function definition

```
(defn my-function [param1 param2]  
  (str param1 param2))
```

Function call

```
(my-function 1 "two") ;"1two"
```

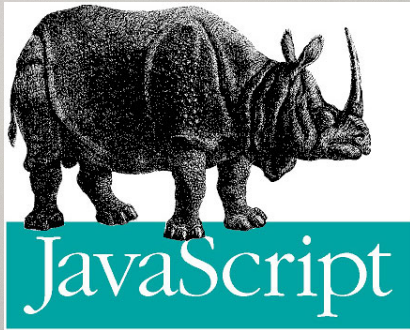
# Clojure syntax

Code is data, data is code

# Data structures

Persistent data structures

# Clojurescript



# core.async

- ▶ Communicating sequential processes (CSP) ... Go



# core.async

- ▶ Communicating sequential processes (CSP) ... Go
- ▶ Asynchronous, reactive, event-driven ... multithreaded

# core.async

- ▶ Communicating sequential processes (CSP) ... Go
- ▶ Asynchronous, reactive, event-driven ... multithreaded
- ▶ Channel  $\simeq$  Queues

# core.async

- ▶ Communicating sequential processes (CSP) ... Go
- ▶ Asynchronous, reactive, event-driven ... multithreaded
- ▶ Channel  $\simeq$  Queues
- ▶ Clojure + Clojurescript

# core.async

```
(go (>! channel {:name "event" :value 3}))
```

```
(go (<! channel))
```

# core.async

## Asynchronous

```
async_call(param, function(result){  
  console.log(result);  
});
```

# core.async

## Asynchronous

```
async_call(param, function(result){  
  console.log(result);  
});
```

```
async_call(param, new Future());
```



# core.async

## Asynchronous

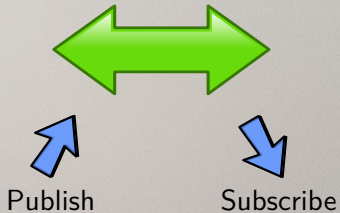
```
async_call(param, function(result){  
  console.log(result);  
});
```

```
async_call(param, new Future());
```

```
async_call(param, channel);
```

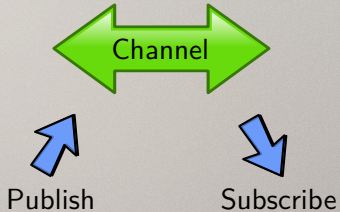
# core.async

Pub/Sub



# core.async

Pub/Sub



# core.async

Composable

► map

# core.async

Composable

- ▶ map
- ▶ filter

# core.async

## Composable

- ▶ map
- ▶ filter
- ▶ merge



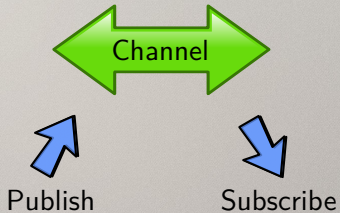
# core.async

## Composable

- ▶ map
- ▶ filter
- ▶ merge
- ▶ choose

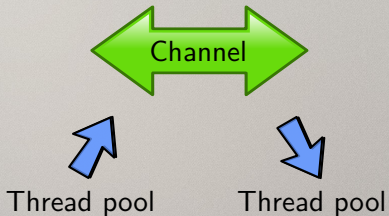
# core.async

Pub/Sub



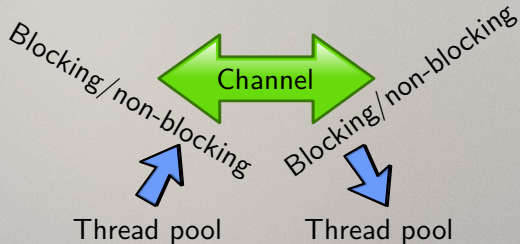
# core.async

Pub/Sub



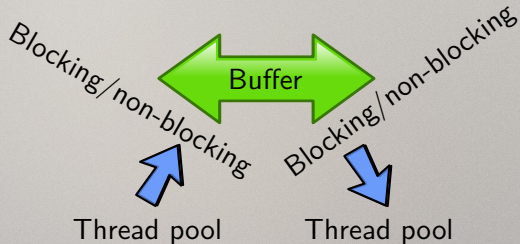
# core.async

Pub/Sub



# core.async

Pub/Sub



# core.typed

## Static typing

- Early error catching



# core.typed

## Static typing

- ▶ Early error catching
- ▶ Documentation by types

# core.typed

## Static typing

- ▶ Early error catching
- ▶ Documentation by types
- ▶ Faster

# core.typed

## Static typing

- ▶ Early error catching
- ▶ Documentation by types
- ▶ Faster

# core.typed

## Static typing

- ▶ Early error catching
- ▶ Documentation by types
- ▶ Faster

## Dynamic typing

- ▶ Duck typing  $\Rightarrow$  better code reuse

# core.typed

## Static typing

- ▶ Early error catching
- ▶ Documentation by types
- ▶ Faster

## Dynamic typing

- ▶ Duck typing  $\Rightarrow$  better code reuse
- ▶ Live programming

# core.typed

## Static typing

- ▶ Early error catching
- ▶ Documentation by types
- ▶ Faster

## Dynamic typing

- ▶ Duck typing  $\Rightarrow$  better code reuse
- ▶ Live programming
- ▶ Compilation speed / compiler complexity



# core.typed

- ▶ Optional type system

# core.typed

- Optional type system

No more beans

```
(HMap :mandatory { :a Number } :optional { :b Symbol })
```

# core.typed

- Optional type system

No more beans

```
(HMap :mandatory { :a Number } :optional { :b Symbol })
```

Union types

```
(Fn [(U String Number) -> (Value 3)])
```

# Immutable

► JBoss

# Immutable

- ▶ JBoss
- ▶ Standard deployment

# Immutable

- ▶ JBoss
- ▶ Standard deployment
- ▶ Logs



# Immutable

- ▶ JBoss
- ▶ Standard deployment
- ▶ Logs
- ▶ Jobs / Daemons

# Immutable

- ▶ JBoss
- ▶ Standard deployment
- ▶ Logs
- ▶ Jobs / Daemons
- ▶ Messaging (HornetQ)

# Immutant

- ▶ JBoss
- ▶ Standard deployment
- ▶ Logs
- ▶ Jobs / Daemons
- ▶ Messaging (HornetQ)
- ▶ Caching (Infinispan)

# Immutant

- ▶ JBoss
- ▶ Standard deployment
- ▶ Logs
- ▶ Jobs / Daemons
- ▶ Messaging (HornetQ)
- ▶ Caching (Infinispan)
- ▶ Distributed transactions

# Immutable

- ▶ JBoss
- ▶ Standard deployment
- ▶ Logs
- ▶ Jobs / Daemons
- ▶ Messaging (HornetQ)
- ▶ Caching (Infinispan)
- ▶ Distributed transactions
- ▶ Clustering

# Immutable

- ▶ JBoss
- ▶ Standard deployment
- ▶ Logs
- ▶ Jobs / Daemons
- ▶ Messaging (HornetQ)
- ▶ Caching (Infinispan)
- ▶ Distributed transactions
- ▶ Clustering
- ▶ REPL



# Wrap up

- ▶ Lisp is great

# Wrap up

- ▶ Lisp is great
- ▶ Asynchronous / multithread  $\Rightarrow$  `core.async`

# Wrap up

- ▶ Lisp is great
- ▶ Asynchronous / multithread  $\Rightarrow$  `core.async`
- ▶ Type checking  $\Rightarrow$  `core.typed`

# Wrap up

- ▶ Lisp is great
- ▶ Asynchronous / multithread  $\Rightarrow$  `core.async`
- ▶ Type checking  $\Rightarrow$  `core.typed`
- ▶ Application server  $\Rightarrow$  Immutant