Clojure core.async Reference

https://clojure.github.io/core.async/

processes/threads

All return a channel immediately to the calling thread. When execution completes, the value of the last expression in <body> (or return value of <f>) will be put on the returned channel.

`(go <body>)`	asynchronously execute <body>
`(go-loop [<bindings>] <body>)`	equivalent to `(go (loop [<bindings>] <body>))`
`(thread <body>)`	run <body> in new thread
`(thread-call <f>)`	call function <f> in a new thread

creating channels

`(chan)`	create an unbuffered channel
`(chan <n>)`	create a channel with buffer size <n>
`(chan <buf>)`	create a channel with a given buffer (see below)

taking a value from a channel

Taking operations will return/pass `nil` if the channel is closed.

`(<! <c>)`	take from <c> in a go block; parks until value available
`(<!! <c>)`	take from <c> out of go block; blocks until value available
`(take! <c> <f>)`	take from <c> asynchronously; pass value to <f>

putting a value on a channel

Putting operations will return/pass `true` if port is not closed. You cannot put `nil` on a channel.

`(>! <c> <v>)`	put <v> on <c> in a go block; parks until buffer space free
`(>!! <c> <v>)`	put <v> on <c> out of go block; blocks until buffer space free
`(put! <c> <v> <f>)`	put <v> on <c> asynchronously; pass true to <f> if <c> is not closed

creating buffers

`(buffer <n>)`	create a fixed buffer of size <n>
`(dropping-buffer <n>)`	create a buffer of size <n> that drops new value when full
`(sliding-buffer <n>)`	create a buffer of size <n> that drops oldest value when full

alternative operators

`(alts! [<port1> ... <portn>])`	operate on 1 available port in go block; parks until available
`(alts! <ports> <option value>)`	like above but with options
when <port> is	operation is
<c>	(<! <c>)
[<c> <v>]	(>! <c> <v>)
when <option value> is	
:default <v>	returns <v> immediately if nothing available
:priority true	if 2 or more available ports, resolve tie with order in <ports> vector
`(alts!! [<port1> ... <portn>])`	operate on 1 available <port> out of go block; blocks until available
when <port> is	operation is
<c>	(<!! <c>)
[<c> <v>]	(>!! <c> <v>)

PurelyFunctional.tv