

Design

v1.0.0

```
interface Stage:
```

```
* Handle(Item) Item: Handles the Item to be moved through the Pipe.
```

Stage corresponds to stages of computations in 1. Item refers to any type satisfying 6.

```
StageFunc func(Item) Item
```

StageFunc is a wrapper that converts functions which would satisfy Stage into Stages.

```
class Pipe(...Stage):
```

```
* Receive(Item): Receive puts the Item in the Pipe to be processes.
```

```
* Deliver() Item: Deliver blocks until an Item is done being processed and returns it.
```

Pipe connects Stages in the given order satisfying 4 and the rest of 1. Receive and Deliver hide concurrency by immediately returning after putting an Item in the Pipe and blocking until an Item is done satisfying 5. Concurrent functions are started for every Stage whenever the Pipe has an Item that handles Items concurrently when available then places them in the next Stage satisfying 2 and 3. These functions exit when the Pipe is empty. Pipe accepts varargs since the more common use-case involves a hard-coded list of Stages.

```
func Process(Pipe, ...Item) []Item
```

Process is a utility to run many Items through a Pipe. It accepts varargs so that arguments don't need to be put in a []Item from their original type.

v1.1.0

```
interface Consumer:
```

```
* Consume(Item): Consumes the Item.
```

Consumer corresponds to the consumer of Items in 1 and receives Items as they come out of the Pipe.

```
ConsumerFunc func(Item)
```

ConsumerFunc is a wrapper that converts functions which would satisfy Consumer into Consumers.

```
ProcessAndConsume(Pipe, Consumer, ...Item)
```

ProcessAndConsume is a utility that works like Process except passes the Items to the Consumer as they come out of the Pipe.