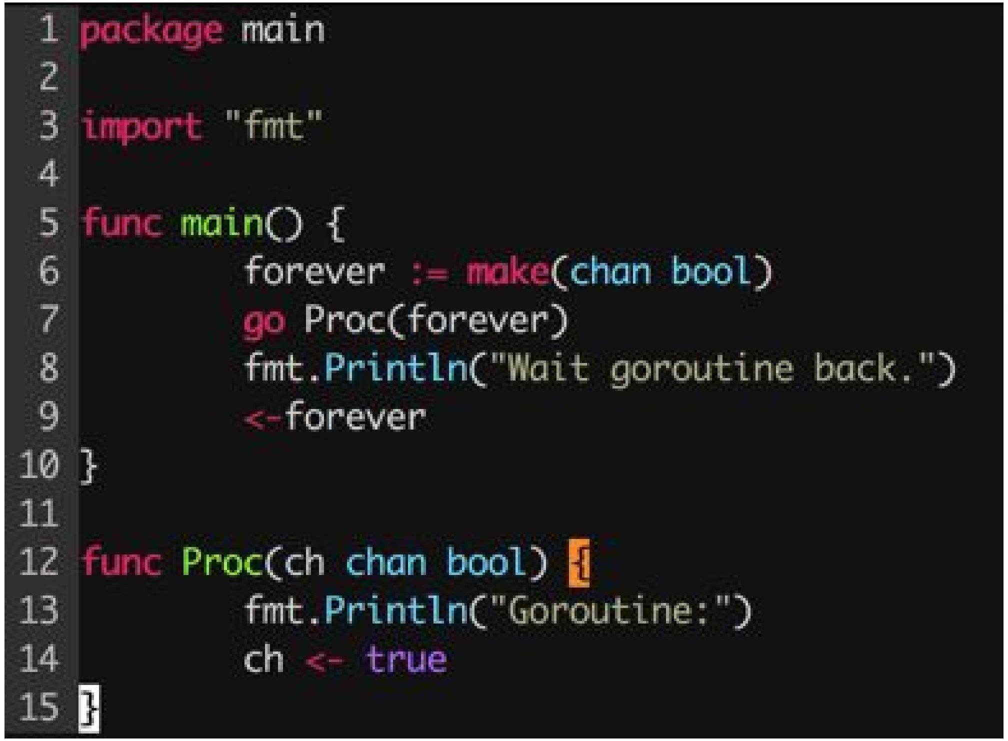
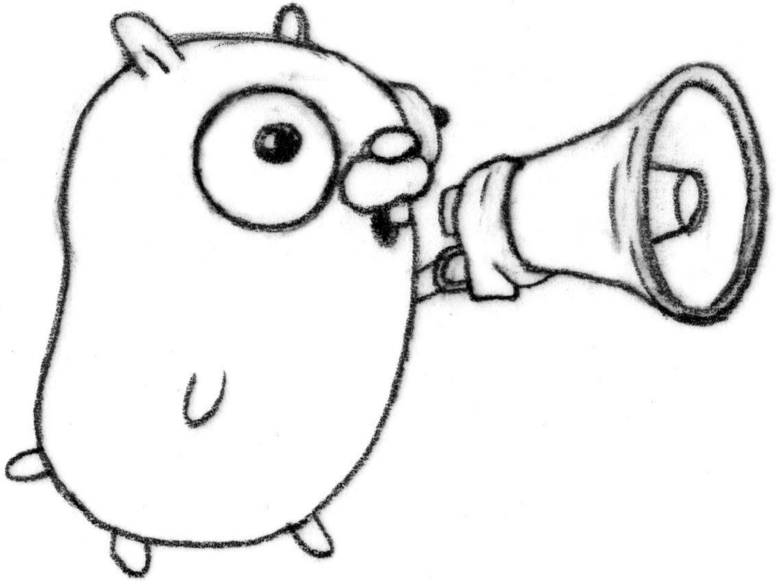
Use Go Channel to write a Disk Queue

Evan Lin @ Linker Networks

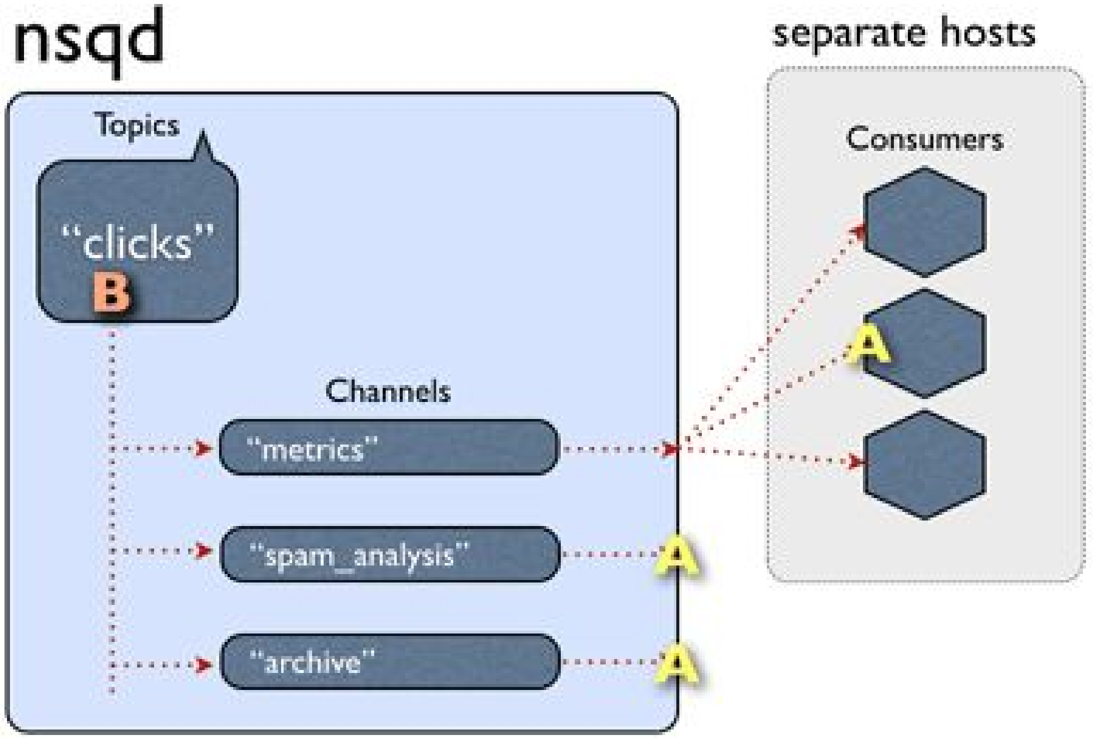
Go Channel



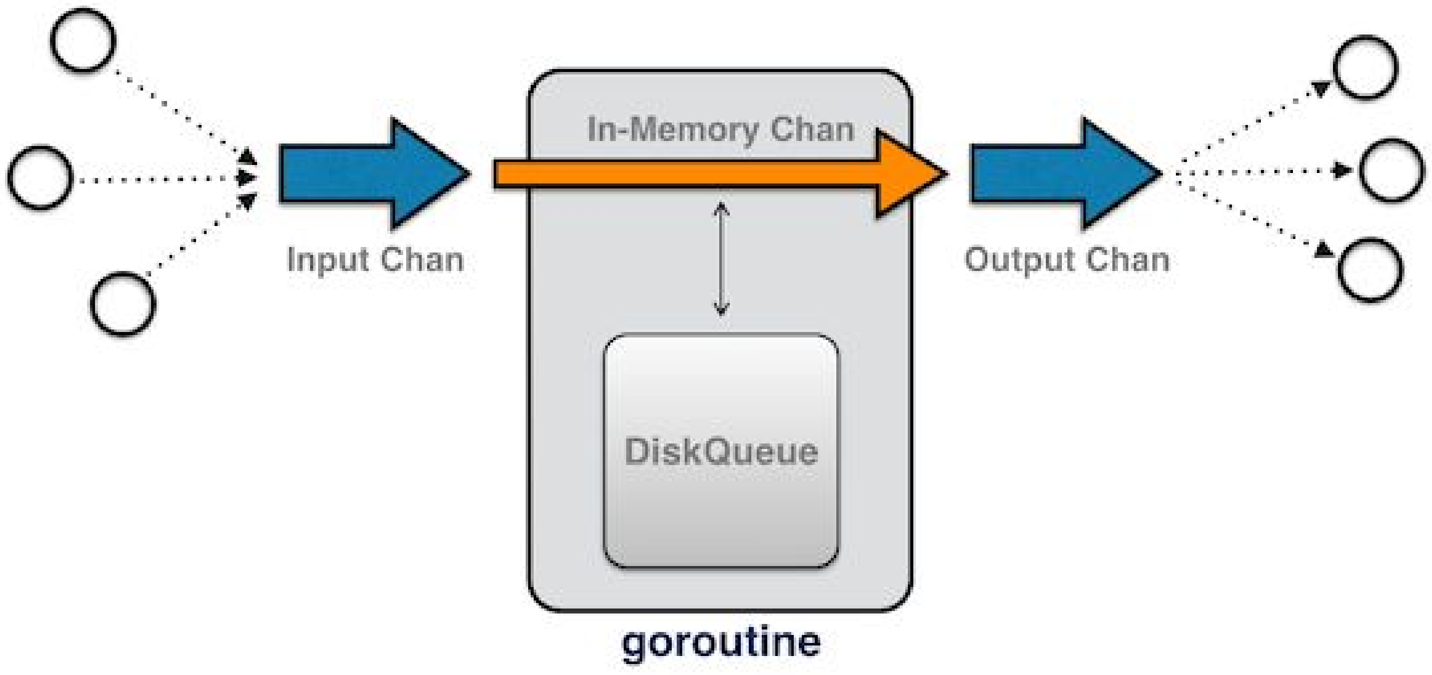
```go
package main

import "fmt"

func main() {
	forever := make(chan bool)
	go Proc(forever)
	fmt.Println("Wait goroutine back.")
	<-forever
}

func Proc(ch chan bool) {
	fmt.Println("Goroutine:")
	ch <- true
}
```



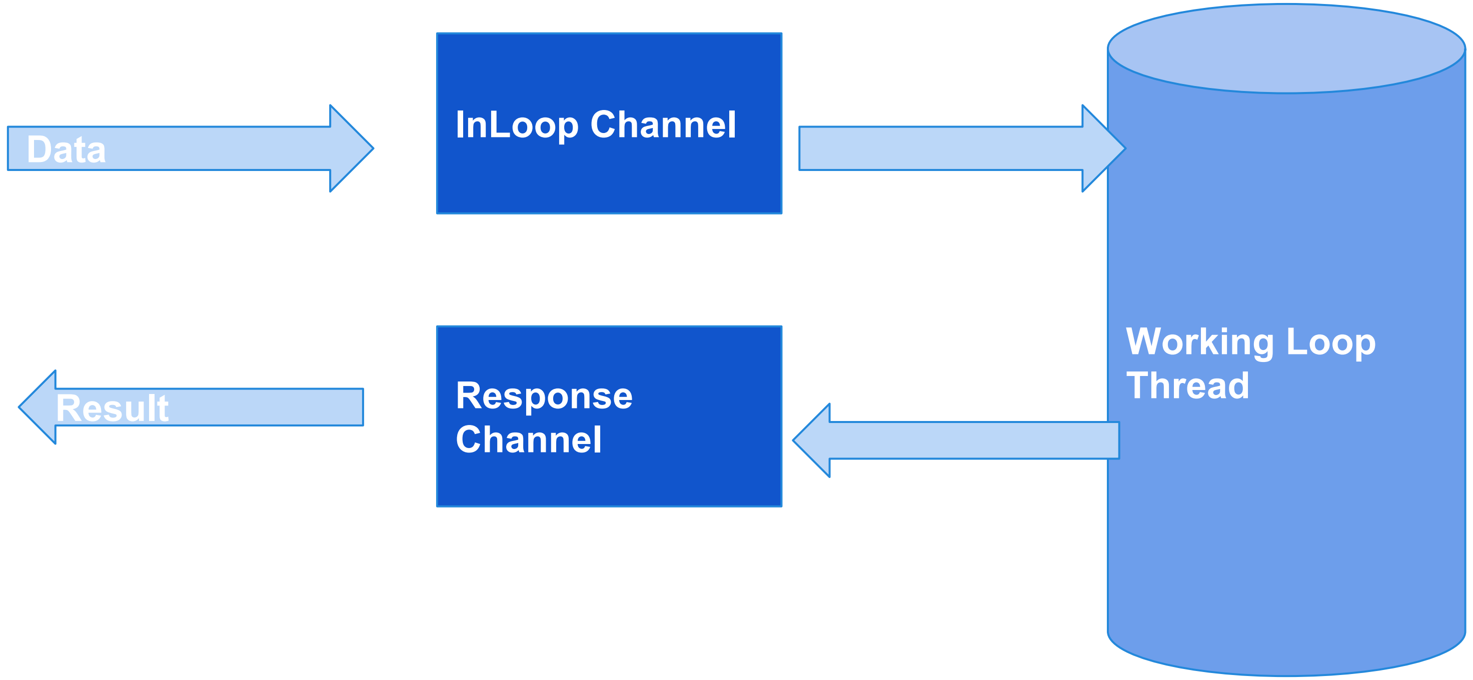
What is NSQ


nsqd
Topics
"clicks"
B
Channels
"metrics"
"spam_analysis"
"archive"
A
A
A
separate hosts
Consumers

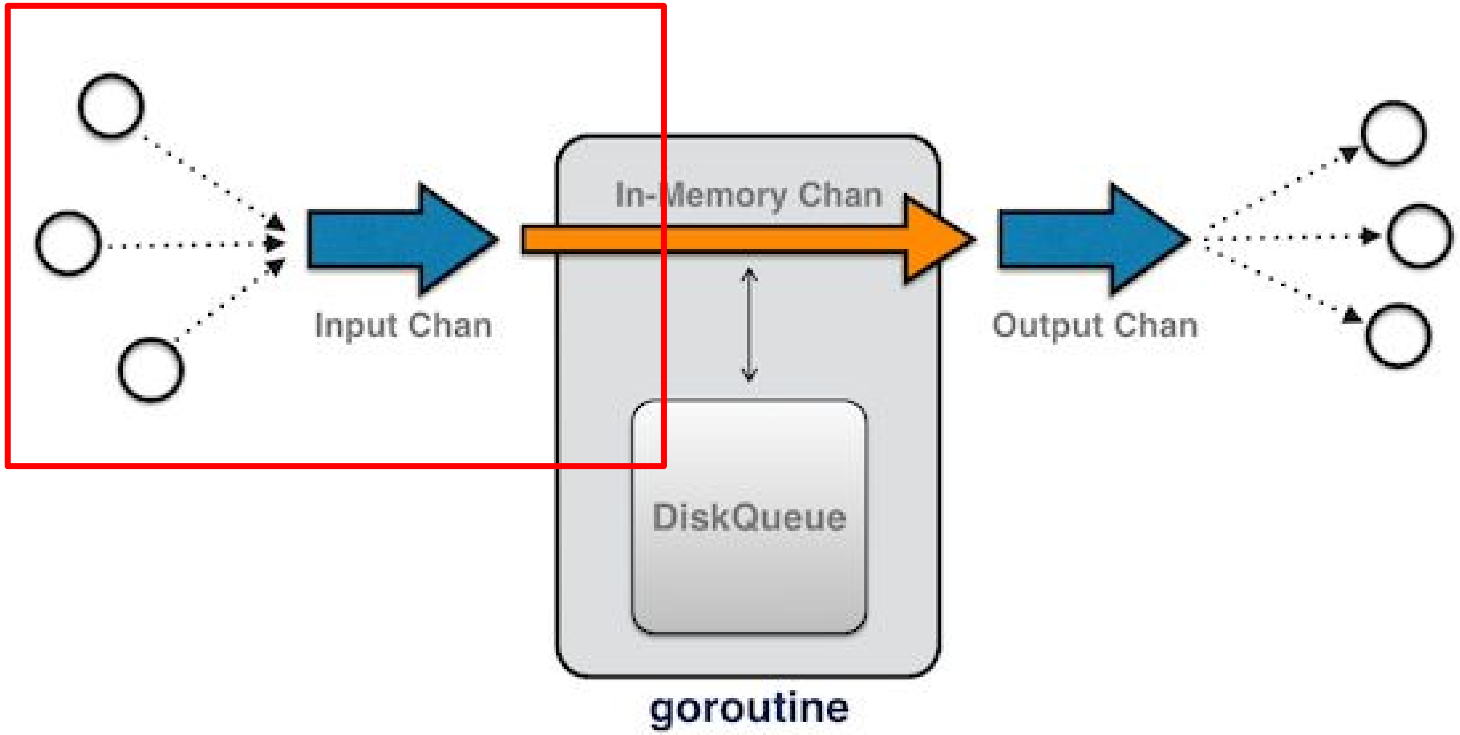
What is DiskQueue in NSQ



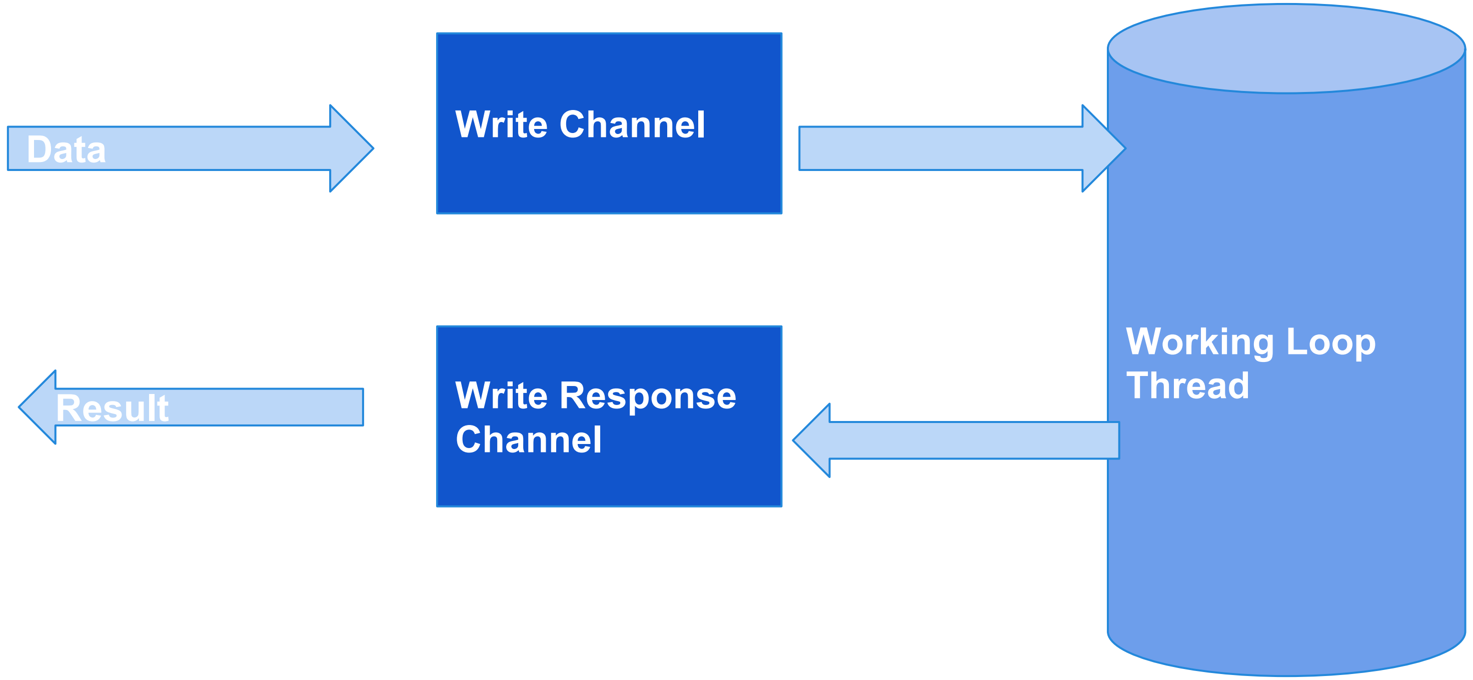
Basic Channel Concept in Disk Queue

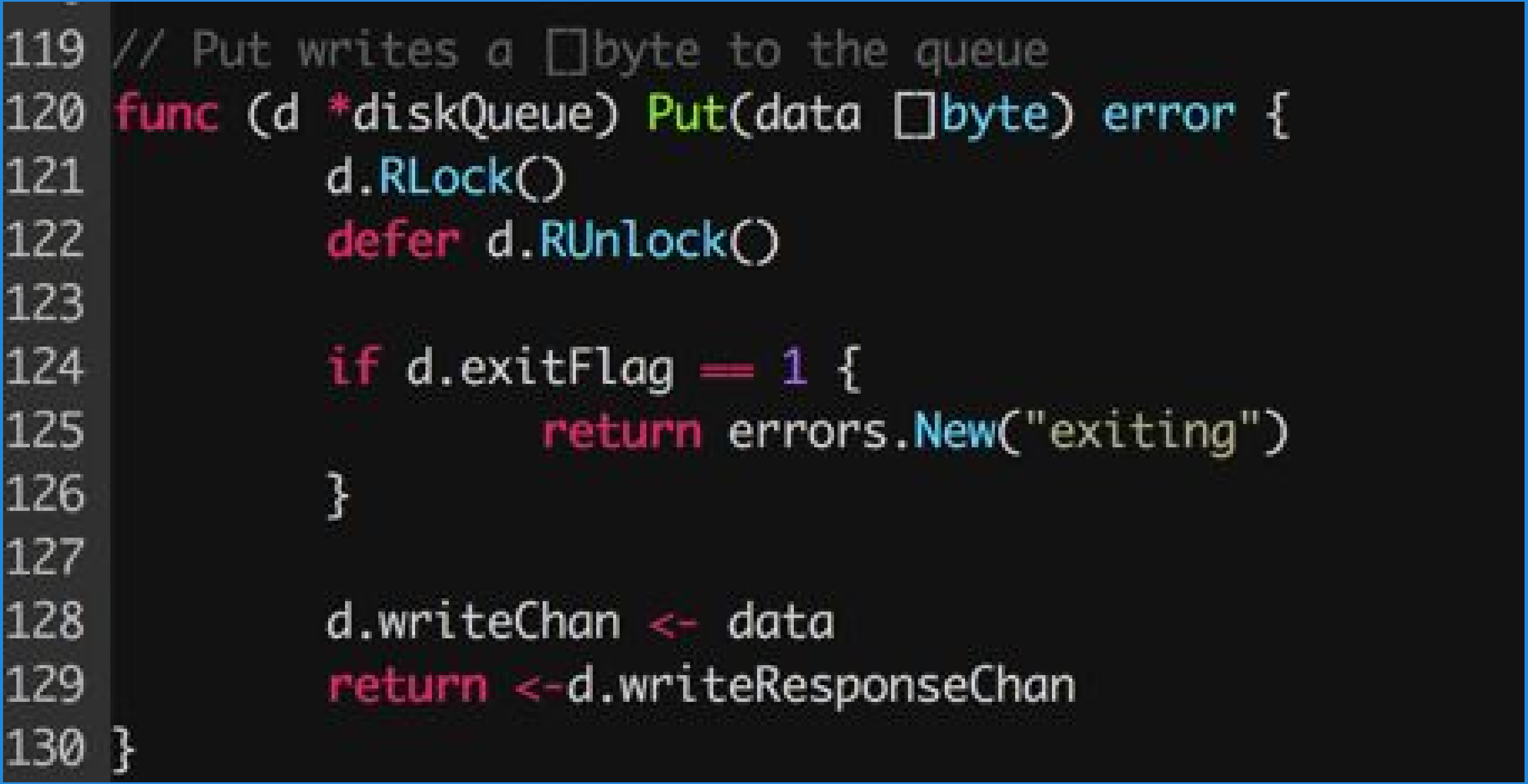


Input (write) Channel



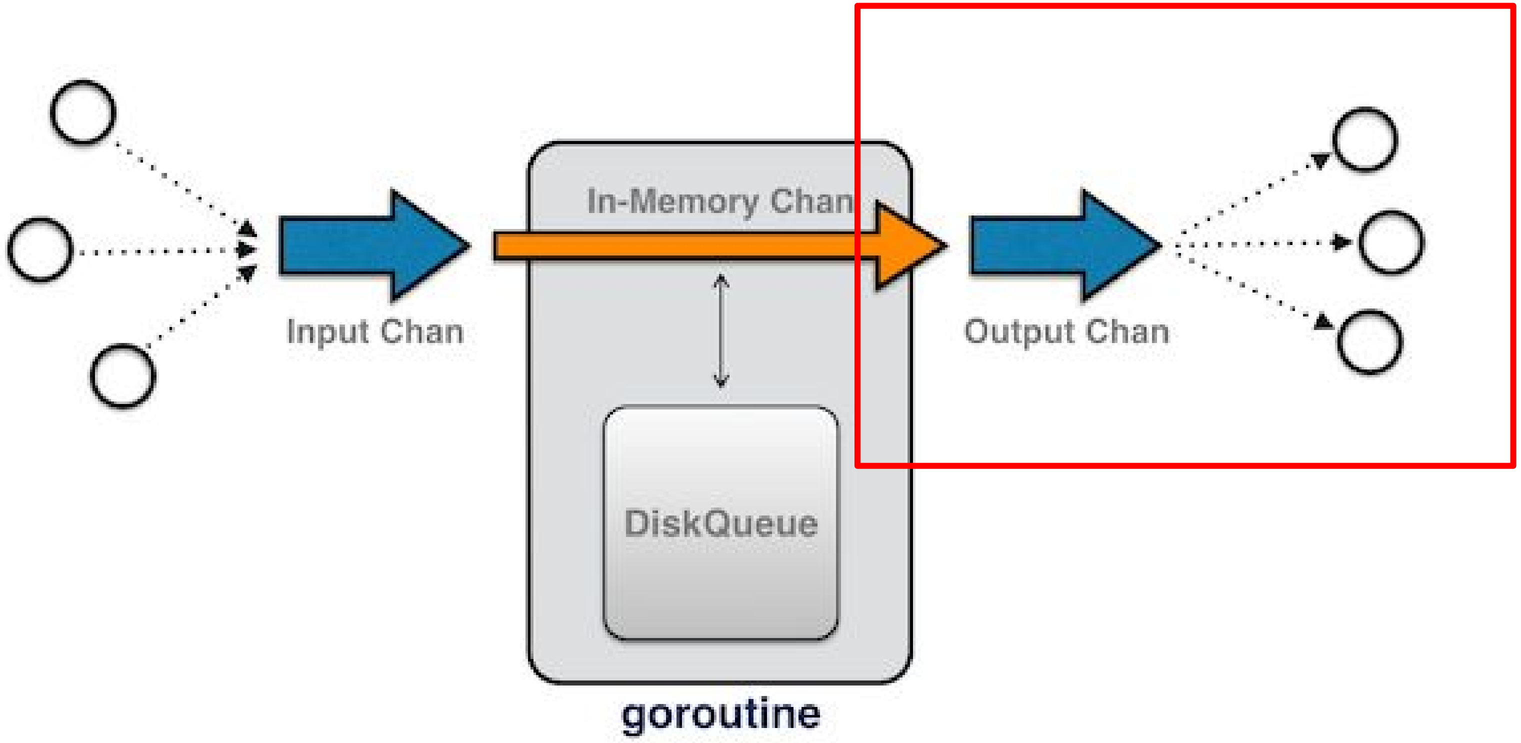
Input (write) Channel



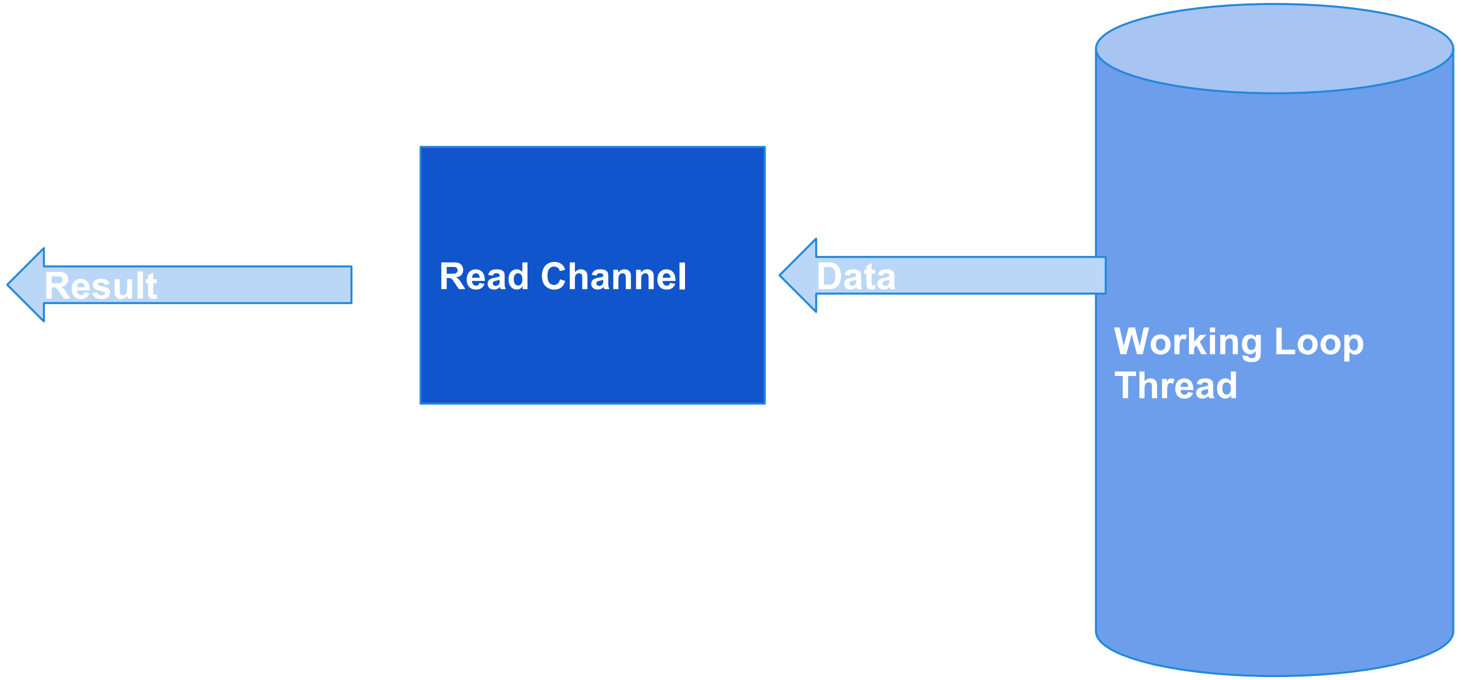


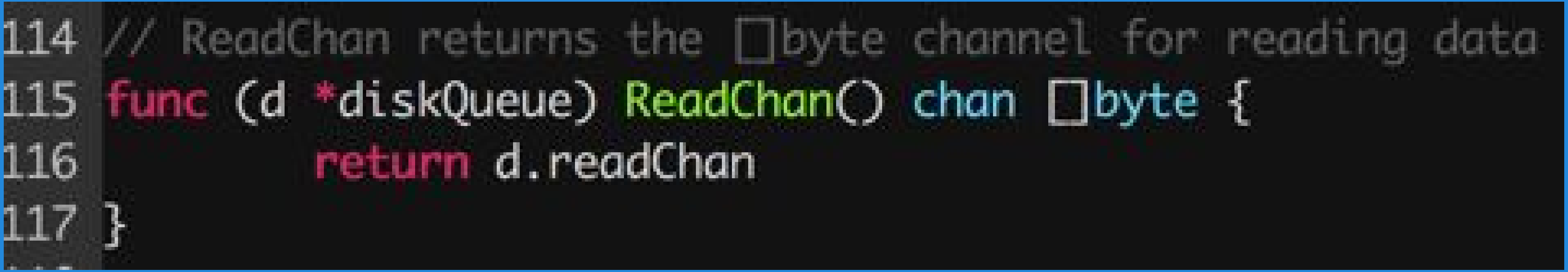
```go
// Put writes a []byte to the queue
func (d *diskQueue) Put(data []byte) error {
	d.RLock()
	defer d.RUnlock()

	if d.exitFlag == 1 {
		return errors.New("exiting")
	}

	d.writeChan <- data
	return <-d.writeResponseChan
}
```

Output (read) Channel



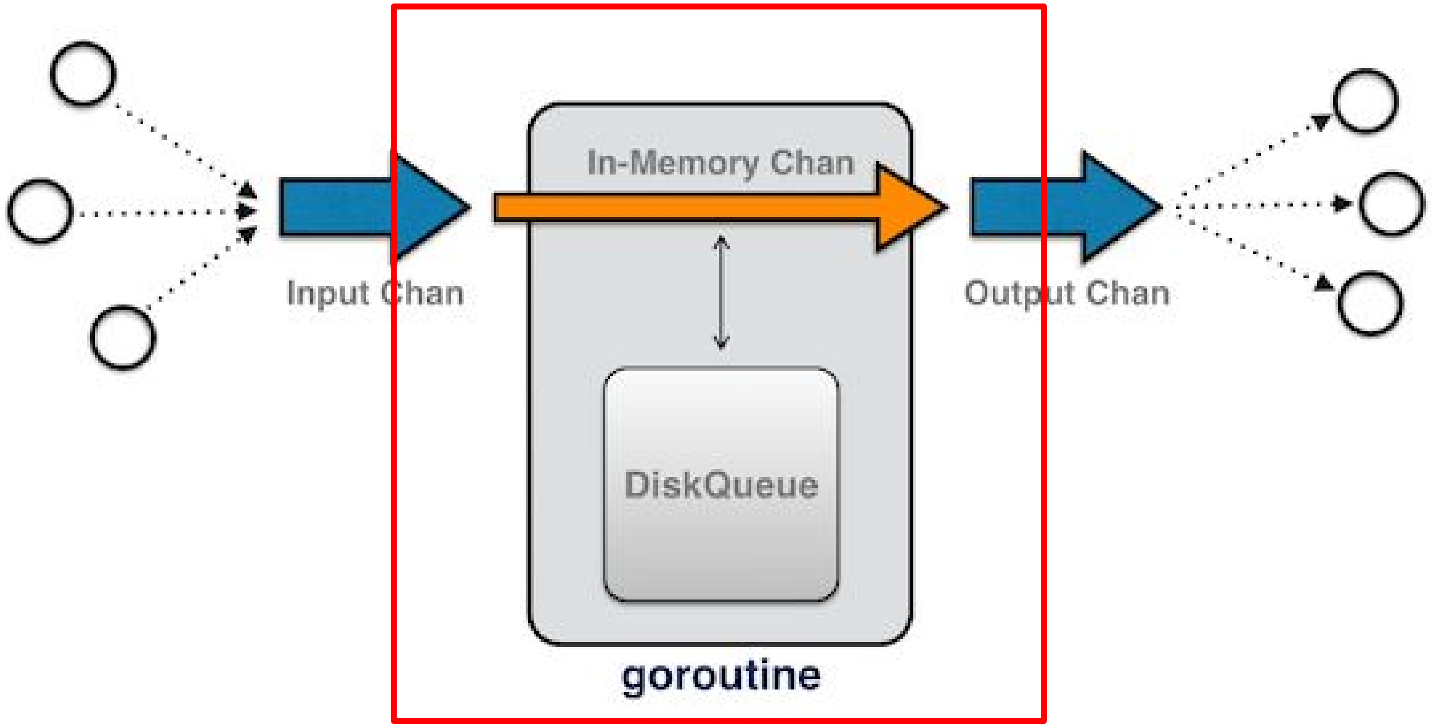
Out (read) Channel

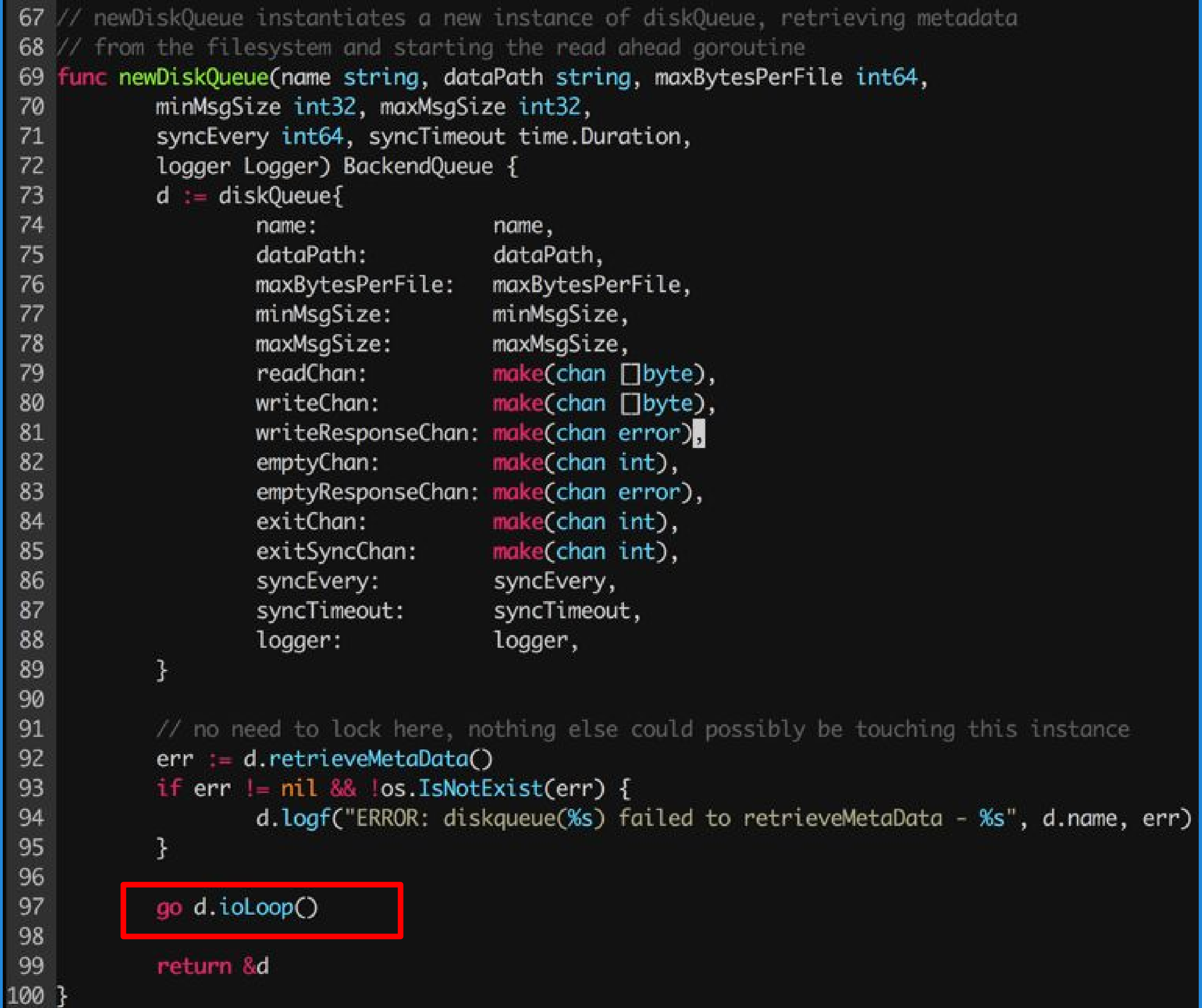




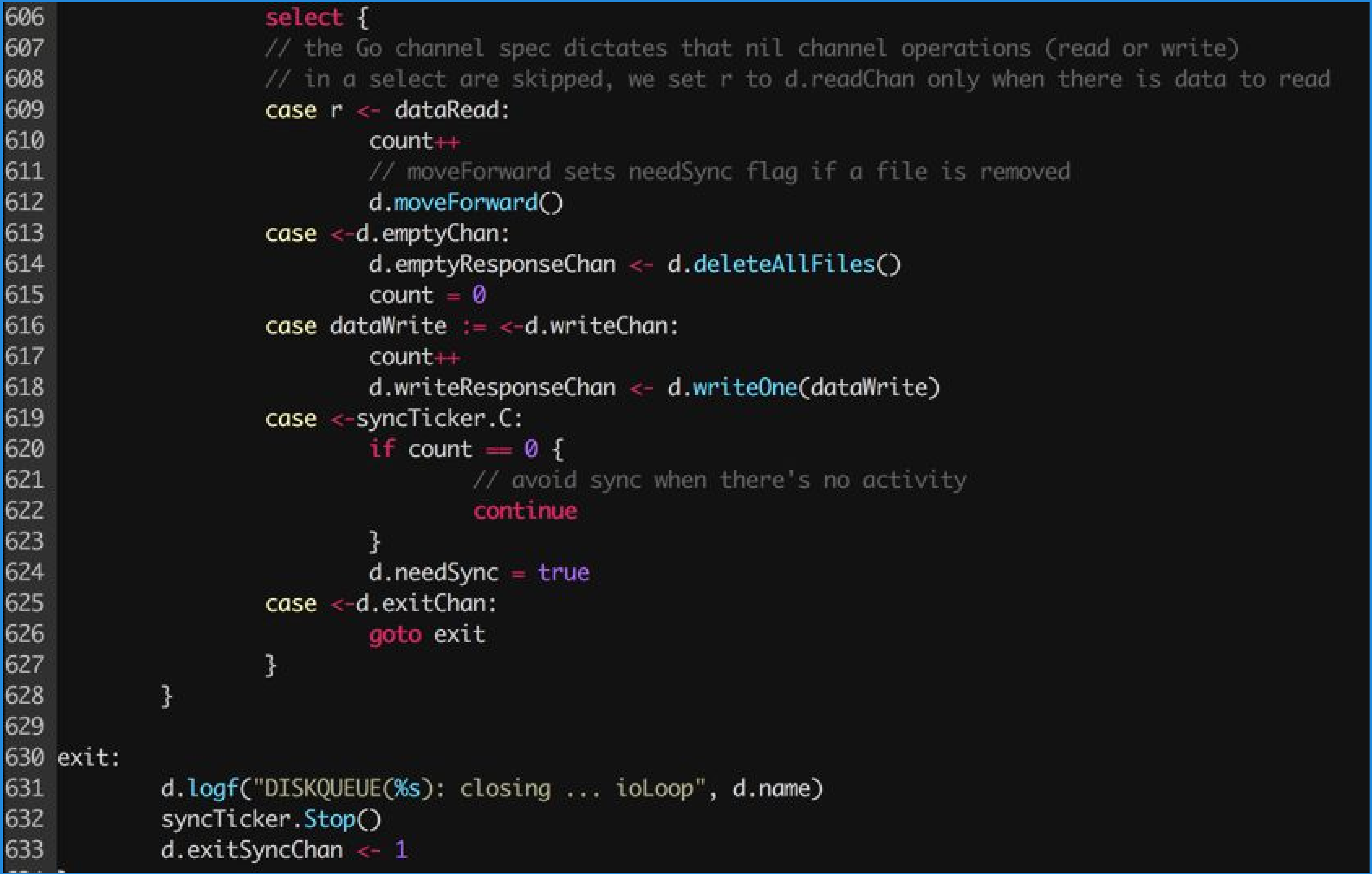
```go
// ReadChan returns the □byte channel for reading data
func (d *diskQueue) ReadChan() chan □byte {
	return d.readChan
}
```

Loop GoRoutine





```go
// newDiskQueue instantiates a new instance of diskQueue, retrieving metadata
// from the filesystem and starting the read ahead goroutine
func newDiskQueue(name string, dataPath string, maxBytesPerFile int64,
	minMsgSize int32, maxMsgSize int32,
	syncEvery int64, syncTimeout time.Duration,
	logger Logger) BackendQueue {
	d := diskQueue{
		name:              name,
		dataPath:          dataPath,
		maxBytesPerFile:   maxBytesPerFile,
		minMsgSize:        minMsgSize,
		maxMsgSize:        maxMsgSize,
		readChan:          make(chan []byte),
		writeChan:         make(chan []byte),
		writeResponseChan: make(chan error),
		emptyChan:         make(chan int),
		emptyResponseChan: make(chan error),
		exitChan:          make(chan int),
		exitSyncChan:      make(chan int),
		syncEvery:         syncEvery,
		syncTimeout:       syncTimeout,
		logger:            logger,
	}

	// no need to lock here, nothing else could possibly be touching this instance
	err := d.retrieveMetaData()
	if err != nil && !os.IsNotExist(err) {
		d.logf("ERROR: diskqueue(%s) failed to retrieveMetaData - %s", d.name, err)
	}

	go d.ioLoop()

	return &d
}
```



```go
		select {
		// the Go channel spec dictates that nil channel operations (read or write)
		// in a select are skipped, we set r to d.readChan only when there is data to read
		case r <- dataRead:
			count++
			// moveForward sets needSync flag if a file is removed
			d.moveForward()
		case <-d.emptyChan:
			d.emptyResponseChan <- d.deleteAllFiles()
			count = 0
		case dataWrite := <-d.writeChan:
			count++
			d.writeResponseChan <- d.writeOne(dataWrite)
		case <-syncTicker.C:
			if count == 0 {
				// avoid sync when there's no activity
				continue
			}
			d.needSync = true
		case <-d.exitChan:
			goto exit
		}
	}

exit:
	d.logf("DISKQUEUE(%s): closing ... ioLoop", d.name)
	syncTicker.Stop()
	d.exitSyncChan <- 1
```