



Basic Go

USING GO ON THE WEB

[HTTP://S3.JFH.ME/470.PDF](http://s3.jfh.me/470.pdf)

About me

- John Hilliard
- MIT '09
- 6-3
- Web Developer for Next Jump since 2008



-
- eCommerce and loyalty product installed in the majority of Fortune 1000 companies
 - In other words, we run a large scale web application that handles lots of traffic and transactions
 - Majority engineers
 - Web developers
 - Database
 - Networking
 - Stack
 - Go, PHP, SQL Server, Apache, Linux

Agenda

- Go overview
 - Why we use it and why you might want to use it
 - Why you might not want to use it
- Crash course
 - Scratching the surface
- Simple web application in Go
 - End to end example using Go to make a web application

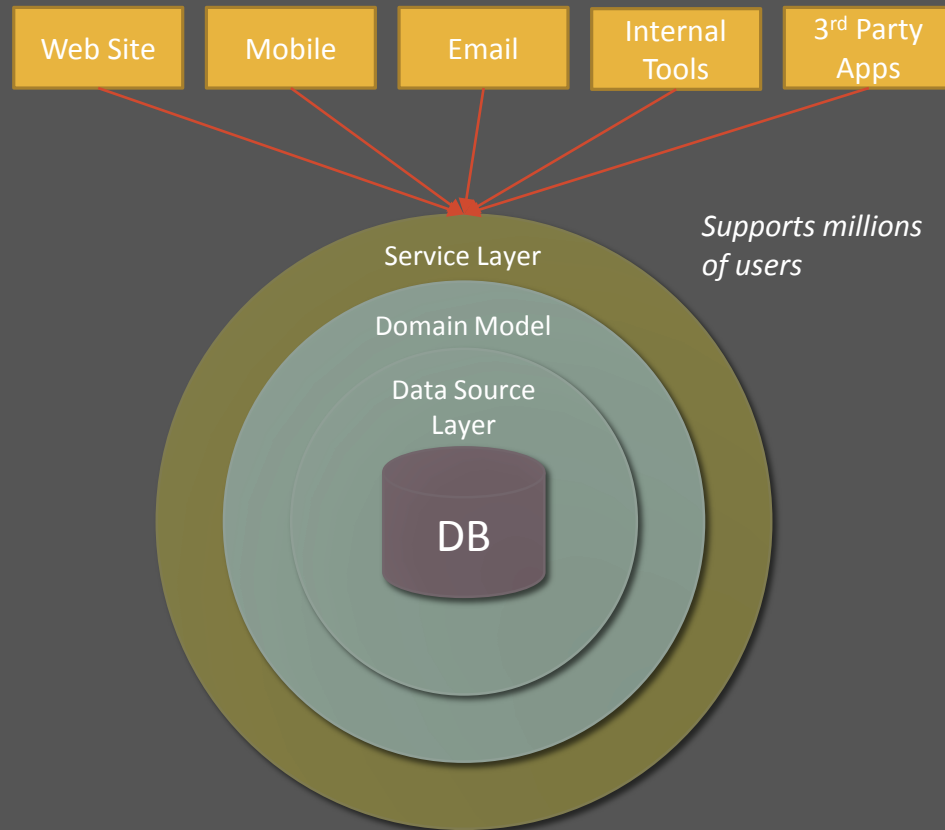
Where Go fits in

Example Products / Services:

- Virtual Currency
- Travel reservation engine
- Transaction processing

We've implemented our service layer in Go

- Consistent data access
 - Logging
 - Security
- Smarter caching
- Shared definitions of objects



M. Fowler, *Patterns of Enterprise Application Architecture*, 1 edition. Boston: Addison-Wesley Professional, 2002.

Hello World

```
package main

import "fmt"

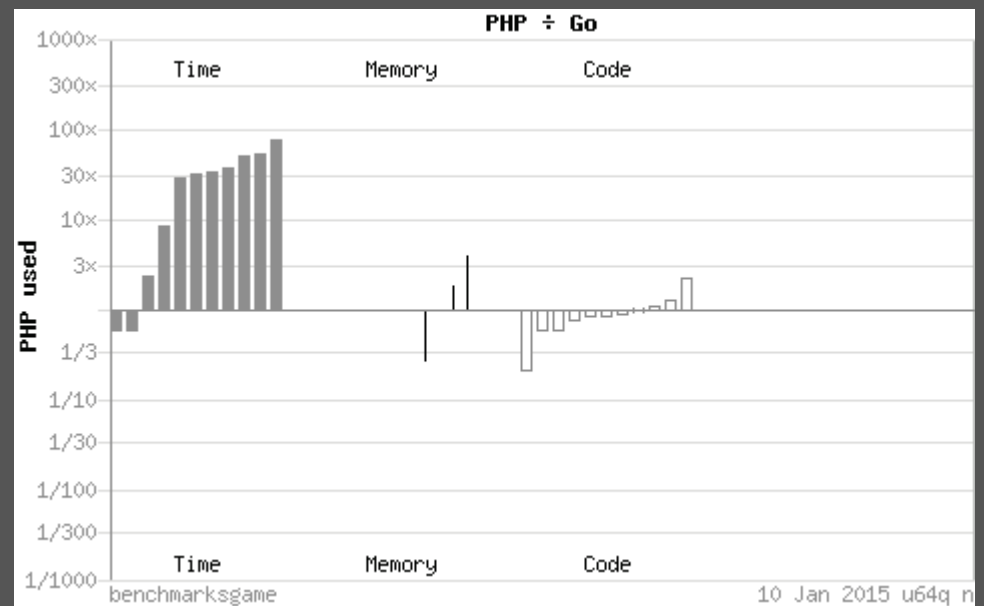
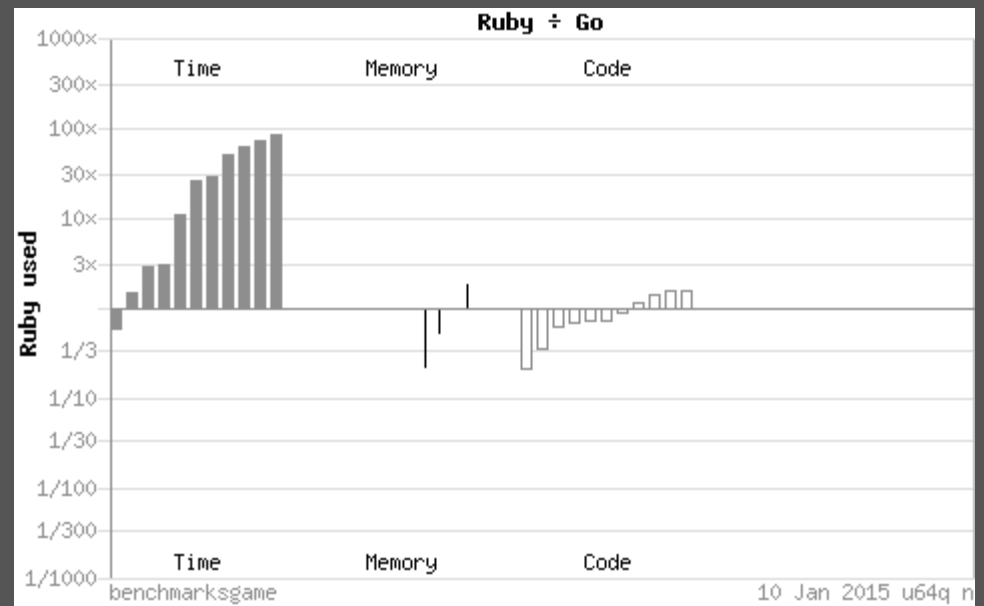
func main() {
    fmt.Println("Hello, world")
}
```

Important Characteristics

- Created in 2009
- Statically typed
- Compiled
- Similar to C

Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt



Note: These aren't web-specific benchmarks

<http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=yarv&lang2=go>

<http://s3.ifh.me/470.pdf>

Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

```
package main

import "fmt"

func main() {
    var a int64
    var b string
    a = 10
    b = "10"
    fmt.Println(a + b)
}
```

invalid operation: a + b (mismatched types int64 and string)

Compiler helps keep code clean:

- No unused variables
- Catches simple mistakes
- Safer refactoring

Predictable code is important. No surprises.

Similar code in Java Script

```
var a;  
var b;  
  
a = 10;  
b = "10";  
  
console.log(a + b);
```

→1010

Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

Predictable code is important. No surprises.

```
package main

import "fmt"

func main() {
    var numbers []int64 = []int64{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    for index, value := range numbers {
        fmt.Printf("Index: %d Value: %d\n", index, value)
    }
}
```

Java Script Looping

```
var numbers = [1,2,3,4,5,6,7,8,9,10];

var number;
for (number in numbers) {
    console.log("Index: " + number + " Value: " + numbers[number]);
}

var len = numbers.length;
var i = 0;
while (i < len) {
    console.log("Index: " + i + " Value: " + numbers[i]);
    i = i + 1
}

i = 0;
do {
    console.log("Index: " + i + " Value: " + numbers[i]);
    i = i + 1
} while(i < len)

numbers.map(function(value, index) {
    console.log("Index: " + index + " Value: " + value);
});
```

Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

Notable Features Missing From Go

- Inheritance
- Overloading
- Generics

Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

Other Examples

- Prefix increment operator

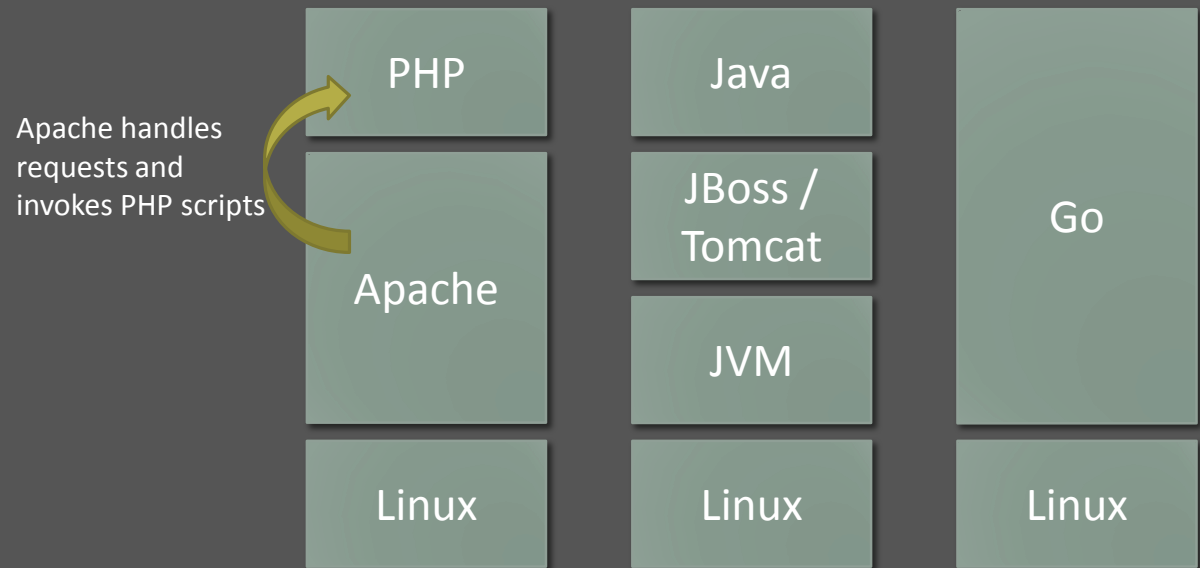
- ++i

- Ternary form

- ([if] ? [true-value] : [false-value])

<http://golang.org/doc/faq>

Configuration: Go vs PHP vs Java

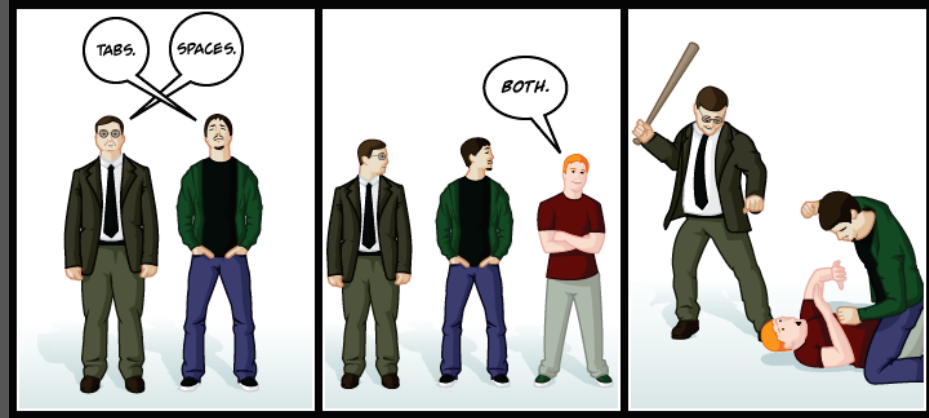


Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

- Running a Go server has less overhead
- Less configuration (xml files)
- Tradeoff (you have to do more yourself)
 - Risk reinventing the wheel

go fmt – no more code standards



<http://www.emacswiki.org/emacs/TabsSpacesBoth>

Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

go fmt your code:

- Easier to **write**: never worry about minor formatting concerns while hacking away
- Easier to **read**: when all code looks the same you need not mentally convert others' formatting style into something you can understand
- Easier to **maintain**: mechanical changes to the source don't cause unrelated changes to the file's formatting; diffs show only the real changes
- **Uncontroversial**: never have a debate about spacing or brace position ever again!

Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

Summary

We use Go because:

- It's fast
- It's simple
 - Easy to learn
 - There aren't wildly different styles – no need to be clever
 - Readable
- It's predictable
 - Static typing gives confidence
- Our developers voted for it
 - Top contenders: Go, Scala, Java

Why not Go?

Random
Internet
Opinions

“Go has the potential to become the next C, that is, to hold the computing world back for 4 (or more?) decades to come. I wish it never had cone into existence.”

- [fishface60](#)

“Seriously? Why don't we just go back to writing programs by using punch cards? I understand the need to reduce moving parts, but deliberately omitting a very useful abstraction mechanism is just insane.”

- [torquay](#)

Nobody's perfect...

- Lacks features that are common in many modern languages
- Strict types can be a burden
- Precise memory management is not easy
- New Language → new libraries
- Simple spec → more code (less clever)

Go Crash Course

- Packages
- Variables
- Functions
- Flow Control
- Important Types

For a great intro visit: <http://tour.golang.org>

Packages

- Packages are modules of code
 - Allows you to split code across multiple files and folders
- Packages are **imported** based on the path
 - The last part of the path is typically the package name
- The default package is named main
 - Starting point of your Go application is in the main package

```
package main

import (
    "fmt"
    "html"
)

func main() {
    var data string = "<h1> hi </h1>"
    fmt.Println(html.EscapeString(data))
}
```

Variables

- Can be declared at the function or package level
- := is a great shortcut inside functions
- Case matters for package level variables
 - First letter upper case: Public
 - First letter lower case: Protected
- Variables are typed
 - Can't assign to a string then assign to a number

Common types:

Bool, int, uint, byte, float, string

<https://golang.org/ref/spec#Types>

```
package main

import (
    "fmt"
)

var (
    FirstName string // Accessible everywhere
    LastName  string // only accessible in the package
)

func main() {
    var age int // only accessible in this function
    FirstName = "John"
    LastName = "Hilliard"
    age = 28

    city := "Cambridge" // Declare and initialize automatically
    fmt.Printf("%s %s is %d years old and lives in %s.\n",
        FirstName, LastName, age, city)
}
```

Functions

- Exported names work like variables
- Functions take zero or more arguments
- Functions can return zero or more results

```
package main

import "fmt"

func square(x int) int {
    return x * x
}

func main() {
    squaredNumber := square(42)
    fmt.Println(squaredNumber)
}
```

Flow Control

- if, switch, and for
 - No parenthesis
 - No semicolons
- No do, or while
- range is a special function for iterating

```
package main

import "fmt"

func main() {
    numbers := []int{1, 2, 3, 4, 5}
    for index, value := range numbers {
        if index <= 1 {
            fmt.Printf("Beginning Number: %d\n", value)
        } else if index <= 3 {
            fmt.Printf("Middle Number: %d\n", value)
        } else {
            fmt.Printf("End Number: %d\n", value)
        }
    }
}
```

Important Types

- There are a few other important types
 - Slices – growable arrays
 - Maps – like a dictionary or JS object
 - Structs – more rigid data type
- The make function is used to allocate maps and slices
- The new function is used to allocate a new struct

```
package main

import "fmt"

type (
    Person struct {
        Name string
        Age  int
    }
)

func main() {
    numbers := []int{1, 2, 3, 4, 5}
    numbers = append(numbers, 10)
    fmt.Println(numbers)

    // Create a map
    var numberMap map[int]string
    // Allocation with make
    numberMap = make(map[int]string)
    numberMap[1] = "one"
    numberMap[2] = "two"
    numberMap[3] = "three"
    fmt.Println(numberMap)

    // Declare a person
    var john *Person
    // Allocate with new
    john = new(Person)
    john.Name = "John Hilliard"
    john.Age = 28
    fmt.Println(john)
}
```

Learn more

- There is a lot more to learn

- Pointers
- Concurrency
- Interfaces
- Methods
- Embedded types
- Higher order

- The standard library

The Go Programming Language

Documents Packages The Project Help Blog Search

Try Go

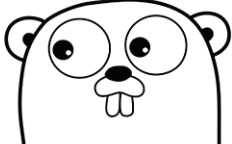
Pop-out

```
// You can edit this code!  
// Click here and start typing.  
package main  
  
import "fmt"  
  
func main() {  
    fmt.Println("Hello, 世界")  
}
```

Hello, World!

Run Share Tour

Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.



Download Go
Binary distributions available for Linux, Mac OS X, Windows, and more.

Featured video

Andrew Gerrand - Go: a simple programming environment - Railsberry 2013
Andrew Gerrand
from Railsberry Go: a simple programming environment

27:16 vimeo

Featured articles

Errors are values
A common point of discussion among Go programmers, especially those new to the language, is how to handle errors. The conversation often turns into a lament at the number of times the sequence
Published 12 January 2015

GothamGo: gophers in the big apple
Last November more than two hundred gophers from all across the United States got together for the first full-day Go conference in New York City.
Published 9 January 2015

Read more

Build version go1.4.
Except as noted, the content of this page is licensed under the Creative Commons Attribution 3.0 License, and code is licensed under a BSD license.
Terms of Service | Privacy Policy

Example Website

End to end Example:

- I'll walk through my process
- Libraries that I used

- Product goals
 - Find movies to watch
 - Ability to search
 - Link two different data sets



Build Data

- Spider the BFI site
- Use the IMDb API to pull data
- Insert it all into SQL Db



Design

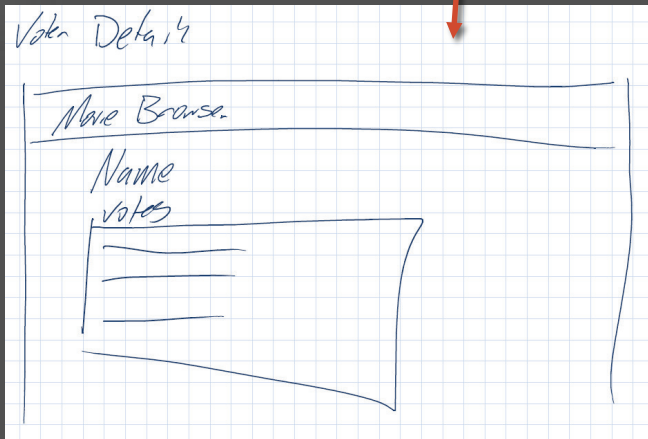
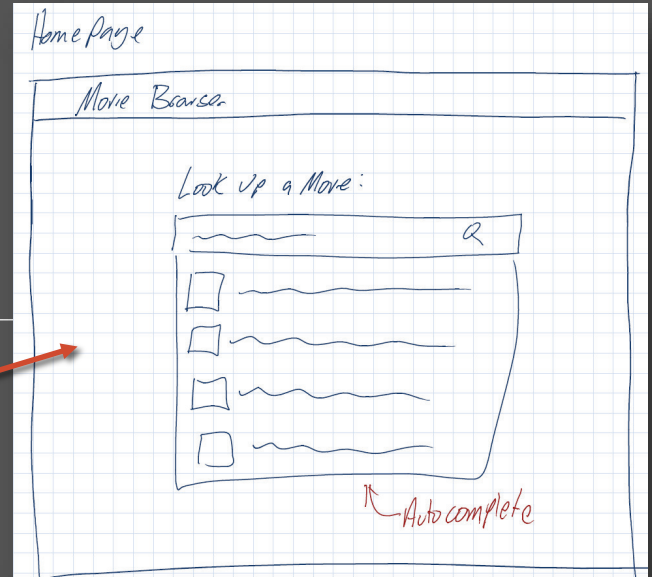
3 main pages

Home Page → Search

Movie Detail → The details of the movie
↳ who voted for it

Voter Detail → people who liked this also liked

↳ list of movies
↳ similar voters



Code

Three Important Libraries

-[net/http](#)

- HTTP server
- HTTP client

-[database/sql](#)

- Generic interface for various SQL implementations

-[html/template](#)

- Simple template library

net/http

Server

```
package main

import (
    "./handlers"
    "log"
    "net/http"
)

func main() {

    http.HandleFunc("/", handlers.Home)
    http.HandleFunc("/movie/", handlers.Movie)
    http.HandleFunc("/movie/list.json", handlers.MovieJson)
    http.HandleFunc("/voter/", handlers.Voter)

    http.Handle("/js/", http.FileServer(http.Dir("./static")))
    http.Handle("/css/", http.FileServer(http.Dir("./static")))
    http.Handle("/img/", http.FileServer(http.Dir("./static")))
    http.Handle("/favicon.ico", http.FileServer(http.Dir("./static/img")))

    log.Println("Starting Server")
    log.Fatal(http.ListenAndServe(":8888", nil))
}
```

Client

```
resp, err := http.Get(movie.PosterUrl)
if err != nil {
    log.Printf("There was an issue fetching the image: %q", err)
    http.NotFound(rw, rq)
    return
}

defer resp.Body.Close()
imageData, imgErr := ioutil.ReadAll(resp.Body)
```

database/sql

1. Create a connection

```
db, err := sql.Open("sqlite3", "./movies.db")
log.Printf("Opening connection to movies database")

if err != nil {
    log.Fatal("Could not open connection to DB: %q", err)
}
```

2. Pull the data

```
func GetMoviesByVoter(voterId string) []*objects.Movie {
    e := data.GetQueryEngine()
    rows, err := e.Query(`SELECT *
        FROM movie
        WHERE ssid IN (
            SELECT vote.movie_id
            FROM VOTE
            WHERE vote.voter_id = ?)
        ORDER BY movie.raw_title;`, voterId)
    movies := make([]*objects.Movie, 0)

    if err != nil {
        log.Printf("There was an issue fetching the movies: %q", err)
        return movies
    }
    defer rows.Close()

    for rows.Next() {
        movies = append(movies, scanMovie(rows))
    }
    return movies
}
```

html/template

```
<div class="row">
  <div class="col-md-4">
    
  </div>
  <div class="col-md-8">
    <h1> {{ .movie.Title }}</h1>

    <h3> Movie Details </h3>
    <table class="table table-striped">
      <tbody>
        <tr>
          <td> Director </td>
          <td> {{.movie.Director}}</td>
        </tr>
        <tr>
          <td> Language </td>
          <td> {{.movie.Language}}</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

Define a template

```
baseLayout.Execute(rw, body)
```

Execute


Putting it all together

Movie Browser

Look Up a Movie

- Choses de la Vie, Les/Little Things in Life, The
- Do The Right Thing
- Heart Is Deceitful Above All Things, The
- I Think They Call Him John
- Most Important Thing: Love, The
- Thin Blue Line, The
- Thin Red Line, The
- Thing, The
- Things to Come
- Two or Three Things I Know About Her...

Movie Browser



The Thin Red Line

Movie Details

Director	Terrence Malick
Language	English, Tok Pisin, Japanese, Greek
Year	1998
Genre	Drama, War
Runtime	170 min
Imdb Rating	7.6
Imdb Votes	121,379
Imdb Page	imdb.com/title/tt0120863
BFI Page	explore.bfi.org.uk/sightandsound

Who voted for this

Voter Name
Carlos Reviriego
Martin Botha
Charles Gant
Fernando Meirelles
David Michóid
Michal Oleszczyk
Hannah Patterson
Heinz Emigholz
Shirin Neshat
Elisabeth Bronfen
Ryan Gilbey
Jean-Marc Vallée
Gareth Evans
Tim Robey

Movie Browser

Khalid Mohamed

[BFI Page](#)

Votes

Movie Name	Director	Year	Imdb Rating
8½	Federico Fellini	1963	8.2
Battleship Potemkin	Sergei M. Eisenstein	1925	8.0
Citizen Kane	Orson Welles	1941	8.4
Imitation of Life	Douglas Sirk	1959	7.9
Pather Panchali	Satyajit Ray	1955	8.2
Pierrot le Fou	Jean-Luc Godard	1965	7.8
Rashomon	Akira Kurosawa	1950	8.4
Some Like It Hot	Billy Wilder	1959	8.3
Taxi Driver	Martin Scorsese	1976	8.4
Vertigo	Alfred Hitchcock	1958	8.4

Thanks!

Jhilliard@nextjump.com

[@praetorian](#)