

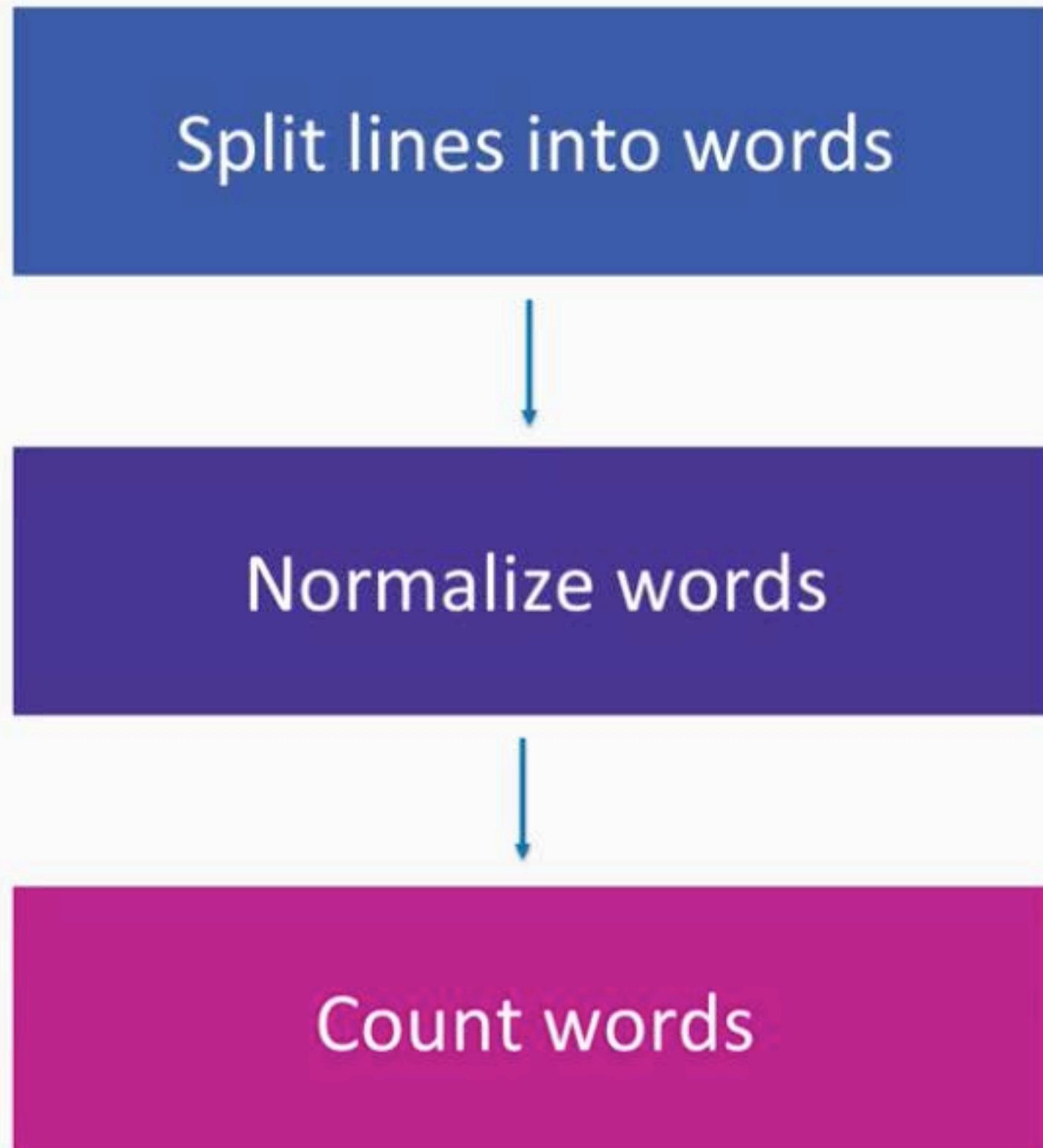
Streams

Streams

- Are lazy Enumerables
- Composing streams creates recipes rather than immediate results
- Useful for processing large or infinite data sets efficiently

Word Counting

Given a list of lines, count the occurrence of all words.



Word Counting

Given a list of lines, count the occurrence of all words.

```
→ Projects mix new words
```

```
* creating README.md
```

```
* creating .gitignore
```

```
* creating mix.exs
```

```
* creating config
```

```
* creating config/config.exs
```

```
* creating lib
```

```
* creating lib/words.ex
```

```
* creating test
```

```
* creating test/test_helper.exs
```

```
* creating test/words_test.exs
```

Your Mix project was created successfully.

You can use "mix" to compile it, test it, and more:

```
cd words
```

```
mix test
```

Run "mix help" for more commands.

```
→ Projects cd words
```

```
→ words em
```

Word Counting

Given a list of lines, count the occurrence of all words.

```
0.200.5@25.1.1 (spacemacs$  
Welcome to  
SPACEMACS beta  
[The best editor is neither Emacs nor Vim, it's Emacs+Vim]  
[?] [Homepage] [Documentation] [Gitter Chat] [Update Spacemacs]  
[Update Packages] [Rollback Package Update]  
[Release Notes] [Search in Spacemacs]  
  
257 packages loaded in 2.493s (e:199 r:2 l:12 b:44)  
  
Warnings:  
- Unknown layer company-mode declared in dotfile.  
- Unknown layer perspectives declared in dotfile.  
① % 1.8k *spacemacs* Spacemacs buffer K utf-8 | 10: 8 Top  
SPC p f
```

Word Counting

Given a list of lines, count the occurrence of all words.

```
defmodule Words do
  @moduledoc """
  Documentation for Words.
  """

  @doc """
  Hello world.

  ## Examples

      iex> Words.hello
      :world

  """
  def hello do
    :world
  end
end
```

1 - 191 words.ex Elixir alchemist Y S P @ K unix | 1: 0 All
Beginning of buffer

Word Counting

Given a list of lines, count the occurrence of all words.

```
defmodule Words do  
  end
```

1 * 26 words.ex Elixir alchemist Y S P @ K unix | 2: 2 All
-- INSERT --

Word Counting

Given a list of lines, count the occurrence of all words.

```
defmodule Words do
  def count(lines) do
    lines
    |> Enum.flat_map(&String.split/1)
    |> Enum.map(&String.downcase/1)
    |> Enum.map(&remove_special_chars/1)
    |> Enum.reduce(%{}, &count_word/2)
  end

  defp remove_special_chars(string) do
    string
    |> String.normalize(:nfd)
    |> String.replace(~r/[^\A-z\s]/u, "")
  end

  defp count_word(word, map) do
    Map.update(map, word, 1, &(&1 + 1))
  end
end
```


Word Counting

Given a list of lines, count the occurrence of all words.

```
* creating lib
* creating lib/words.ex
* creating test
* creating test/test_helper.exs
* creating test/words_test.exs
```

Your Mix project was created successfully.
You can use "mix" to compile it, test it, and more:

```
cd words
mix test
```

Run "mix help" for more commands.

```
→ Projects cd words
→ words emacs -nw
→ words iex -S mix
```

Erlang/OTP 19 [erts-8.2] [source] [64-bit] [smp:8:8] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Compiling 1 file (.ex)

Generated words app

Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)

```
iex(1)> Words.count(["a", "a a a", "b a"])
```

```
%{"a" => 5, "b" => 1}
```

```
iex(2)> 
```

Word Counting

Given a list of lines, count the occurrence of all words.

Getting “War and Peace” and trying it out!

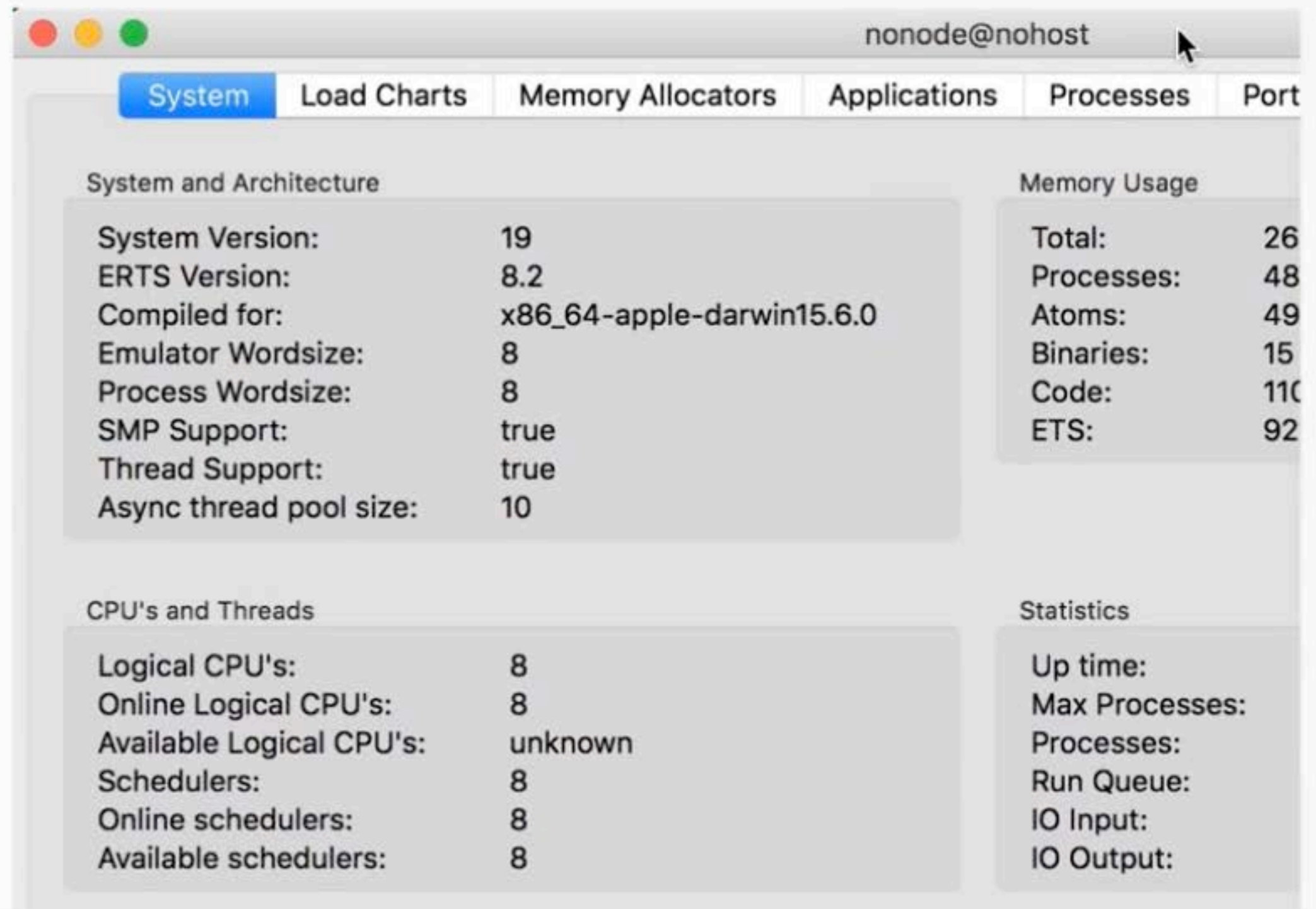
```
→ words iex -S mix  
Erlang/OTP 19 [erts-8.2] [source] [64-bit] [smp:8:8] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]
```

```
Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)  
iex(1)> :observer.start
```

Word Counting

Given a list of lines, count the occurrence of all words.

Getting “War and Peace” and trying it out!



The screenshot shows a web browser window with the address bar displaying 'nonode@nohost'. The page has a navigation bar with tabs: 'System' (selected), 'Load Charts', 'Memory Allocators', 'Applications', 'Processes', and 'Port'. The main content area is divided into four panels:

- System and Architecture:**

| | |
|-------------------------|---------------------------|
| System Version: | 19 |
| ERTS Version: | 8.2 |
| Compiled for: | x86_64-apple-darwin15.6.0 |
| Emulator Wordsize: | 8 |
| Process Wordsize: | 8 |
| SMP Support: | true |
| Thread Support: | true |
| Async thread pool size: | 10 |
- Memory Usage:**

| | |
|------------|-----|
| Total: | 26 |
| Processes: | 48 |
| Atoms: | 49 |
| Binaries: | 15 |
| Code: | 110 |
| ETS: | 92 |
- CPU's and Threads:**

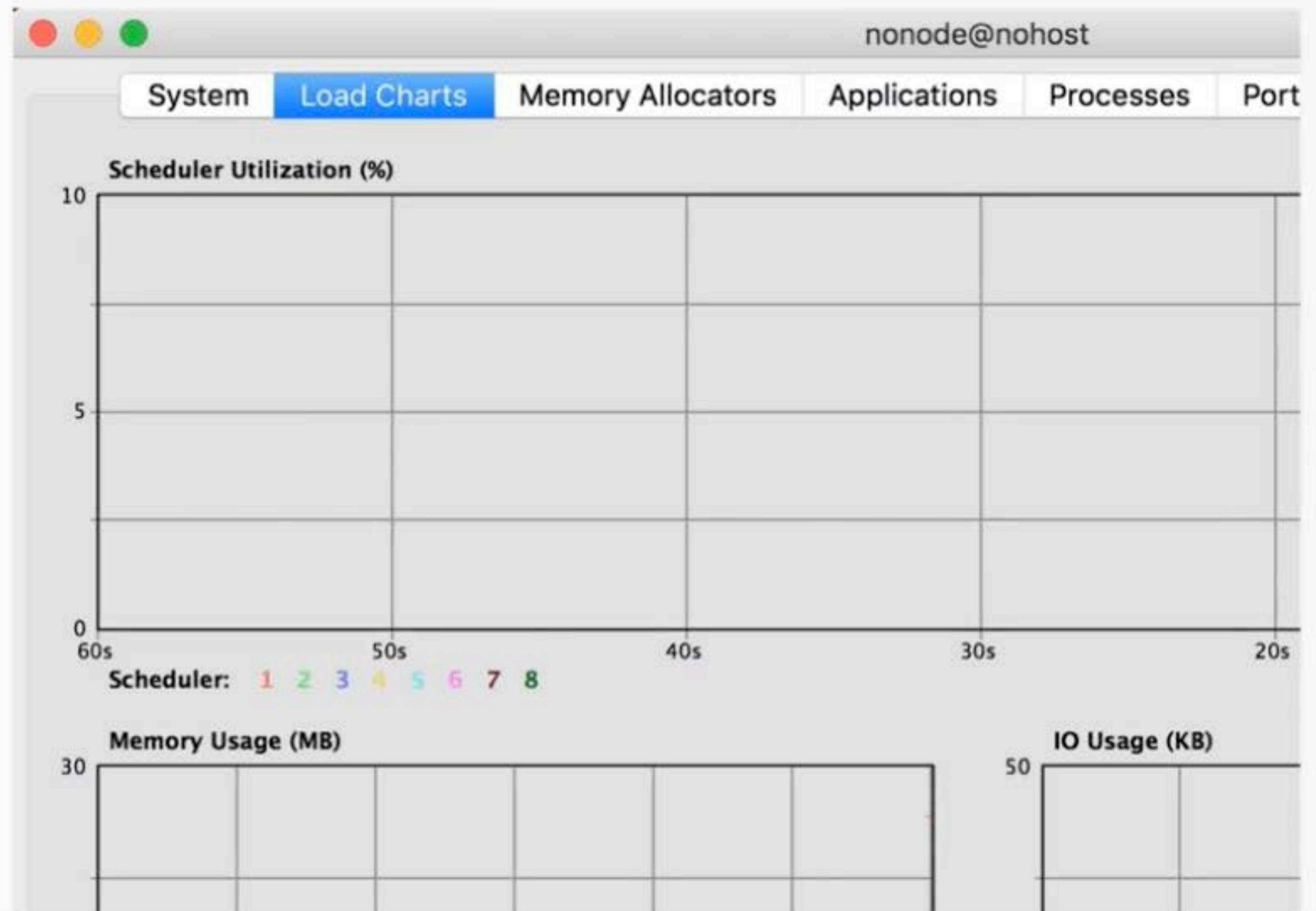
| | |
|--------------------------|---------|
| Logical CPU's: | 8 |
| Online Logical CPU's: | 8 |
| Available Logical CPU's: | unknown |
| Schedulers: | 8 |
| Online schedulers: | 8 |
| Available schedulers: | 8 |
- Statistics:**

| | |
|----------------|--|
| Up time: | |
| Max Processes: | |
| Processes: | |
| Run Queue: | |
| IO Input: | |
| IO Output: | |

Word Counting

Given a list of lines, count the occurrence of all words.

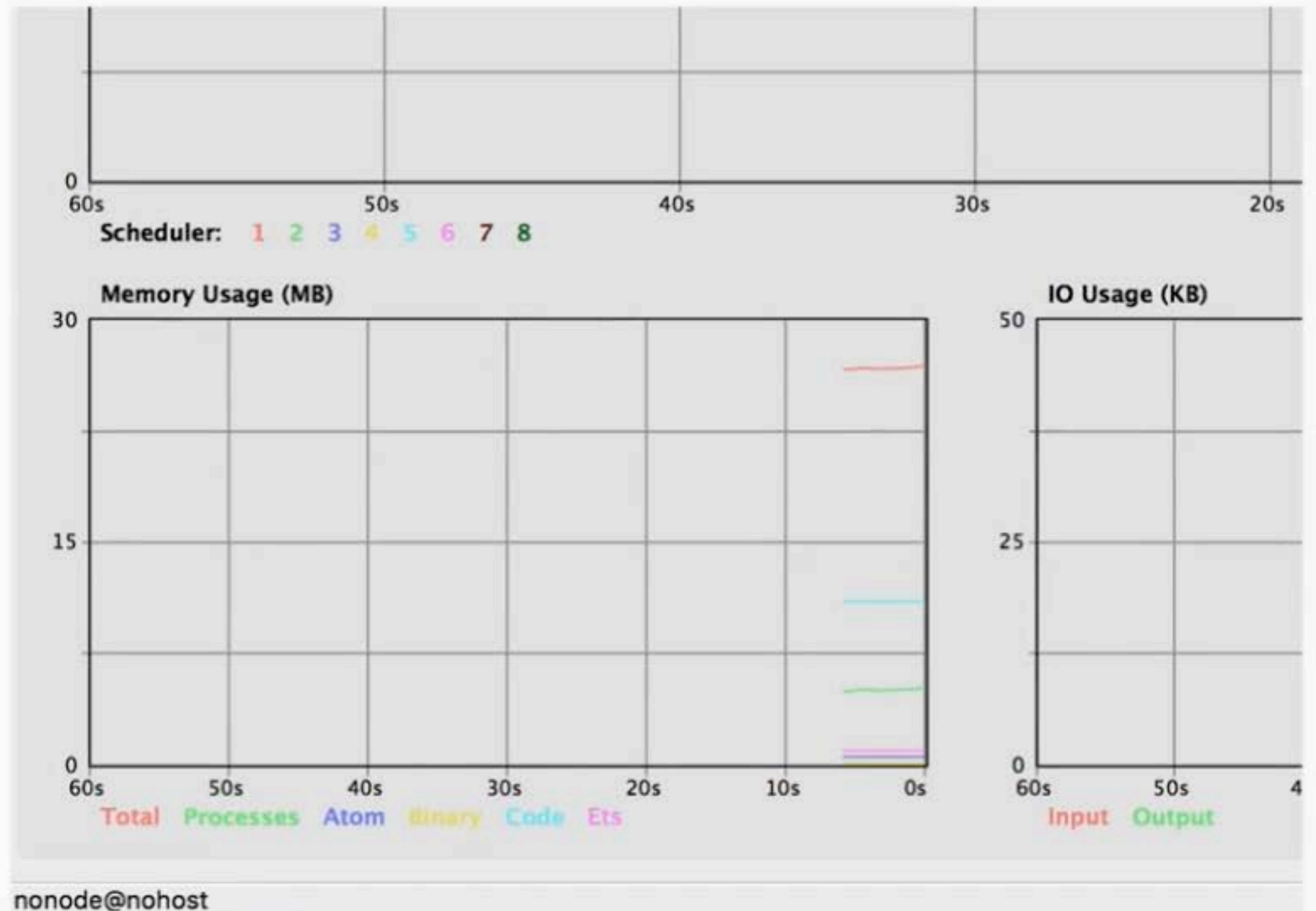
Getting “War and Peace” and trying it out!



Word Counting

Given a list of lines, count the occurrence of all words.

Getting “War and Peace” and trying it out!



Word Counting

Given a list of lines, count the occurrence of all words.

Getting “War and Peace” and trying it out!

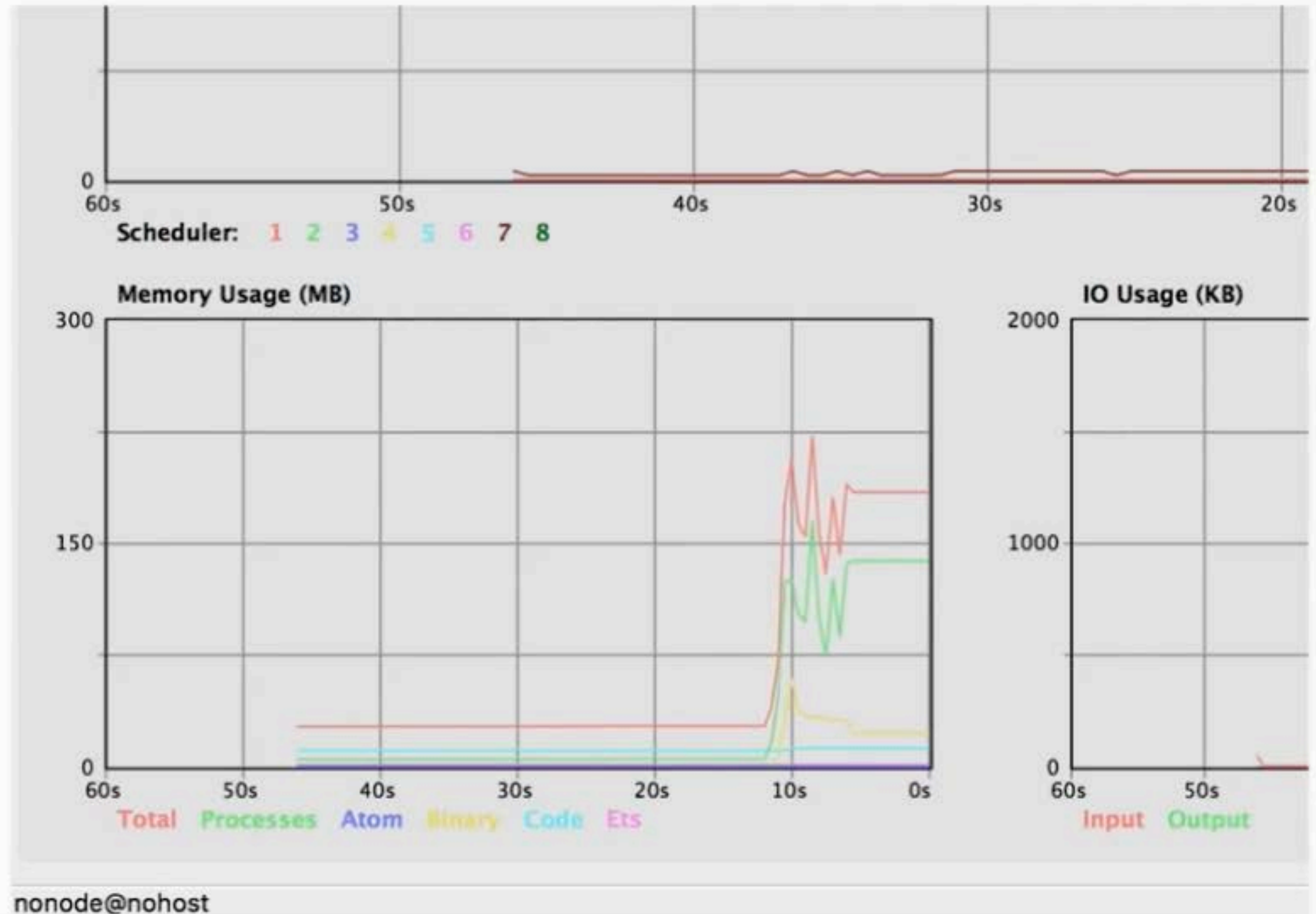
```
→ words iex -S mix
Erlang/OTP 19 [erts-8.2] [source] [64-bit] [smp:8:8] [async-threads:10] [hipe] [
kernel-poll:false] [dtrace]

Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> :observer.start
:ok
iex(2)> Words.count(File.stream!("war_and_peace.txt"))
```


Word Counting

Given a list of lines, count the occurrence of all words.

Getting “War and Peace” and trying it out!



Word Counting

Given a list of lines, count the occurrence of all words.

Getting “War and Peace” and trying it out!

```
→ words iex -S mix
```

```
Erlang/OTP 19 [erts-8.2] [source] [64-bit] [smp:8:8] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]
```

```
Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)
```

```
iex(1)> :observer.start
```

```
:ok
```

```
iex(2)> Words.count(File.stream!("war_and_peace.txt"))
```

```
%{"annihilated" => 1, "sedateness" => 1, "citizens" => 6, "roots" => 1,
  "dissect" => 1, "destroys" => 2, "attempting" => 2, "reminder" => 4,
  "supervision" => 1, "ledge" => 4, "bandy" => 3, "handleless" => 1,
  "onelet" => 1, "injunction" => 1, "warlike" => 7, "affably" => 1, "txt" => 1,
  "incursions" => 1, "walnuts" => 1, "executed" => 32, "silent" => 164,
  "vessel" => 3, "ordering" => 4, "contemplation" => 3, "pillagedthey" => 1,
  "hunting" => 23, "tawny" => 1, "krieg" => 1, "enjoined" => 1, "gervinus" => 2,
  "exceptional" => 8, "eventsagain" => 1, "disdaining" => 1, "refixing" => 1,
  "cease" => 24, "frederick" => 4, "convey" => 6, "woodcutting" => 1,
  "resentment" => 2, "debonair" => 1, "pounced" => 3, "selfwill" => 1,
  "caressing" => 9, "inappropriate" => 1, "nods" => 2, "mamene" => 1,
  "oudinots" => 1, "zherkov" => 37, "dappled" => 1, "efficiency" => 1, ...}
```

```
iex(3)> █
```

It uses too much memory!
~160 MB

Why?

Because of intermediate
representations

- Each step of the computation outputs a copy of the enumerable
- With big data sets, this becomes unfeasible

Word Counting v2

Given a list of lines, count the occurrence of all words.

Using streams for the intermediate steps.

```
defmodule Words do
  def count(lines) do
    lines
    |> Enum.flat_map(&String.split/1)
    |> Enum.map(&String.downcase/1)
    |> Enum.map(&remove_special_chars/1)
    |> Enum.reduce(%{}, &count_word/2)
  end

  defp remove_special_chars(string) do
    string
    |> String.normalize(:nfd)
    |> String.replace(~r/[^\A-z\s]/u, "")
  end

  defp count_word(word, map) do
    Map.update(map, word, 1, &(&1 + 1))
  end
end
```

1 - 422 words.ex Elixir alchemist Y S P @ K unix | 1: 0 All

Word Counting v2

Given a list of lines, count the occurrence of all words.

Using streams for the intermediate steps.

```
defmodule Words do
  def count(lines, mod) do
    lines
    |> mod.flat_map(&String.split/1)
    |> mod.map(&String.downcase/1)
    |> mod.map(&remove_special_chars/1)
    |> Enum.reduce(%{}, &count_word/2)
  end

  defp remove_special_chars(string) do
    string
    |> String.normalize(:nfd)
    |> String.replace(~r/[^\A-z\s]/u, "")
  end

  defp count_word(word, map) do
    Map.update(map, word, 1, &(&1 + 1))
  end
end
```

1 - 424 words.ex Elixir alchemist Y S P @ K unix | 8: 4 All
Saving file /Users/jpoverclock/.emacs.d/.cache/layouts/persp-auto-save...

Word Counting v2

Given a list of lines, count the occurrence of all words.

Using streams for the intermediate steps.

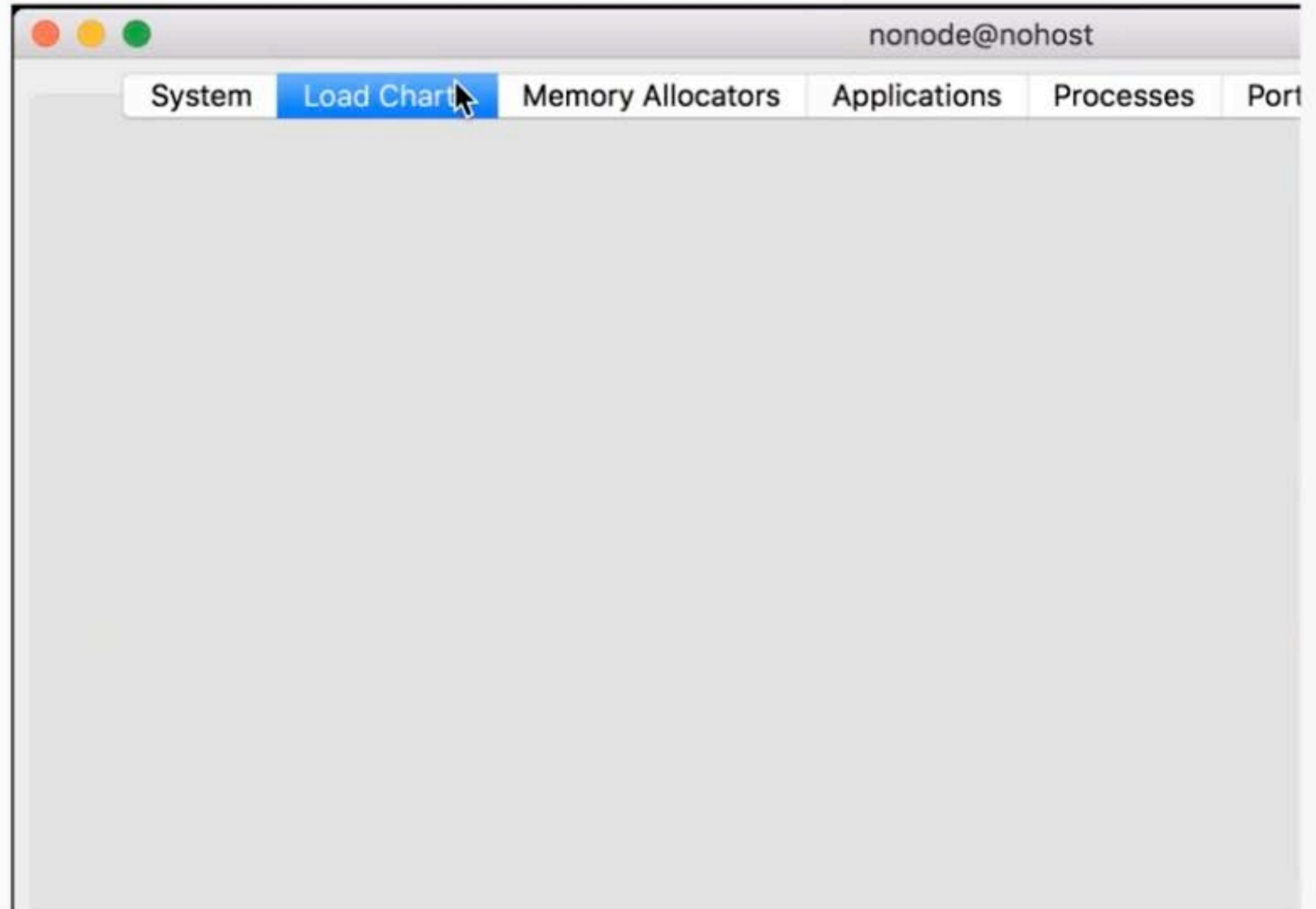
```
→ words emacs -nw
→ words iex -S mix
Erlang/OTP 19 [erts-8.2] [source] [64-bit] [smp:8:8] [async-threads:10] [hipe] [
kernel-poll:false] [dtrace]

Compiling 1 file (.ex)
Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> █
```

Word Counting v2

Given a list of lines, count the occurrence of all words.

Using streams for the intermediate steps.



Word Counting v2

Given a list of lines, count the occurrence of all words.

Using streams for the intermediate steps.

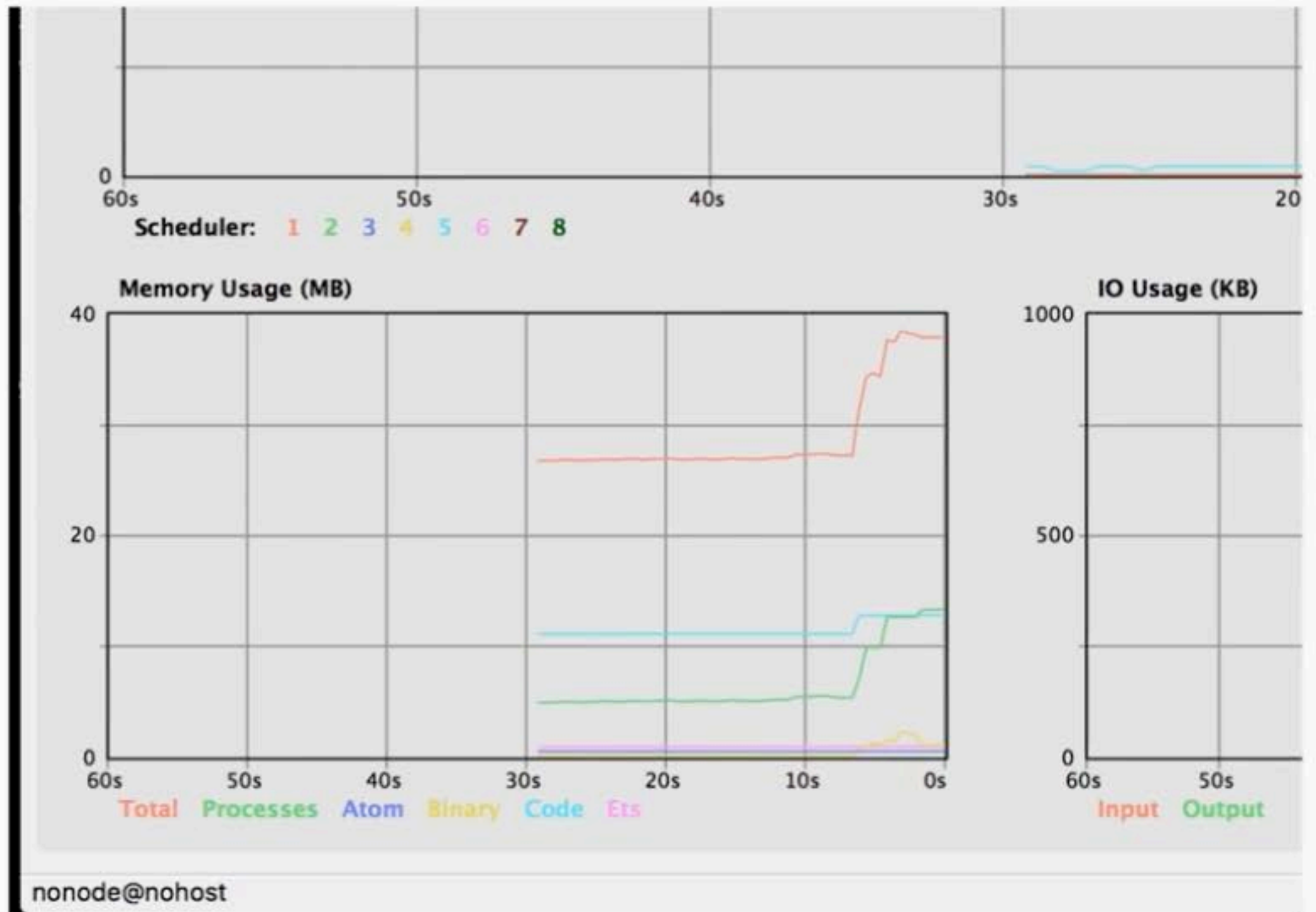
```
→ words emacs -nw
→ words iex -S mix
Erlang/OTP 19 [erts-8.2] [source] [64-bit] [smp:8:8] [async-threads:10] [hipe] [
kernel-poll:false] [dtrace]

Compiling 1 file (.ex)
Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> :observer.start
:ok
iex(2)> Words.count(File.stream!("war_and_peace.txt"), Stream)
```

Word Counting v2

Given a list of lines, count the occurrence of all words.

Using streams for the intermediate steps.



Word Counting v2

Given a list of lines, count the occurrence of all words.

Using streams for the intermediate steps.

```
→ words emacs -nw
→ words iex -S mix
Erlang/OTP 19 [erts-8.2] [source] [64-bit] [smp:8:8] [async-threads:10] [hipe] [
kernel-poll:false] [dtrace]

Compiling 1 file (.ex)
Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> :observer.start
:ok
iex(2)> Words.count(File.stream!("war_and_peace.txt"), Stream)
%{"annihilated" => 1, "sedateness" => 1, "citizens" => 6, "roots" => 1,
  "dissect" => 1, "destroys" => 2, "attempting" => 2, "reminder" => 4,
  "supervision" => 1, "ledge" => 4, "bandy" => 3, "handleless" => 1,
  "onelet" => 1, "injunction" => 1, "warlike" => 7, "affably" => 1, "txt" => 1,
  "incursions" => 1, "walnuts" => 1, "executed" => 32, "silent" => 164,
  "vessel" => 3, "ordering" => 4, "contemplation" => 3, "pillagedthey" => 1,
  "hunting" => 23, "tawny" => 1, "krieg" => 1, "enjoined" => 1, "gervinus" => 2,
  "exceptional" => 8, "eventsagain" => 1, "disdaining" => 1, "refixing" => 1,
  "cease" => 24, "frederick" => 4, "convey" => 6, "woodcutting" => 1,
  "resentment" => 2, "debonair" => 1, "pounced" => 3, "selfwill" => 1,
  "caressing" => 9, "inappropriate" => 1, "nods" => 2, "mamene" => 1,
  "oudinots" => 1, "zherkov" => 37, "dappled" => 1, "efficiency" => 1, ...}
iex(3)> █
```

$\frac{1}{4}$ Memory Usage

Computation is now clustered together, so intermediate representations are no longer an issue

Summary

- Explored Enumerables
- Discussed streams