

# Um estudo do Modelo de Atores aplicado à concorrência de recursos

Fellipe Souto Sampaio

Orientadora: Prof. Dra. Ana Cristina Vieira de Melo

MAC0499 - Trabalho de Formatura Supervisionada  
Instituto de Matemática e Estatística - Universidade de São Paulo

16 de Novembro de 2016

# Motivação

- A programação concorrente está intimamente ligada com a histórica da ciência da computação
- Muitos problemas tem natureza concorrente ou são melhor modelados dessa forma
- Criar aplicações concorrentes não é uma tarefa trivial, as interações entre as diferentes partes de um programa são suscetíveis a uma série de falhas
- A atual tendência do mercado de computadores é o aumento de unidades de processamento em um único chip
- Para tirar vantagem desse poder computacional e resolver os problemas de concorrência de forma mais natural as ferramentas certas precisam ser utilizadas

# Objetivos

- **Aprender** os fundamentos do modelo de atores
- **Entender** a concorrência de recursos através do problema do Jantar dos Filósofos
- **Modelar** o problema dos Filósofos e **implementar** três soluções diferentes utilizando a biblioteca Akka e a linguagem de programação Scala
- **Simular** o problema e **avaliar** o desempenho dos algoritmos a partir de diferentes métricas

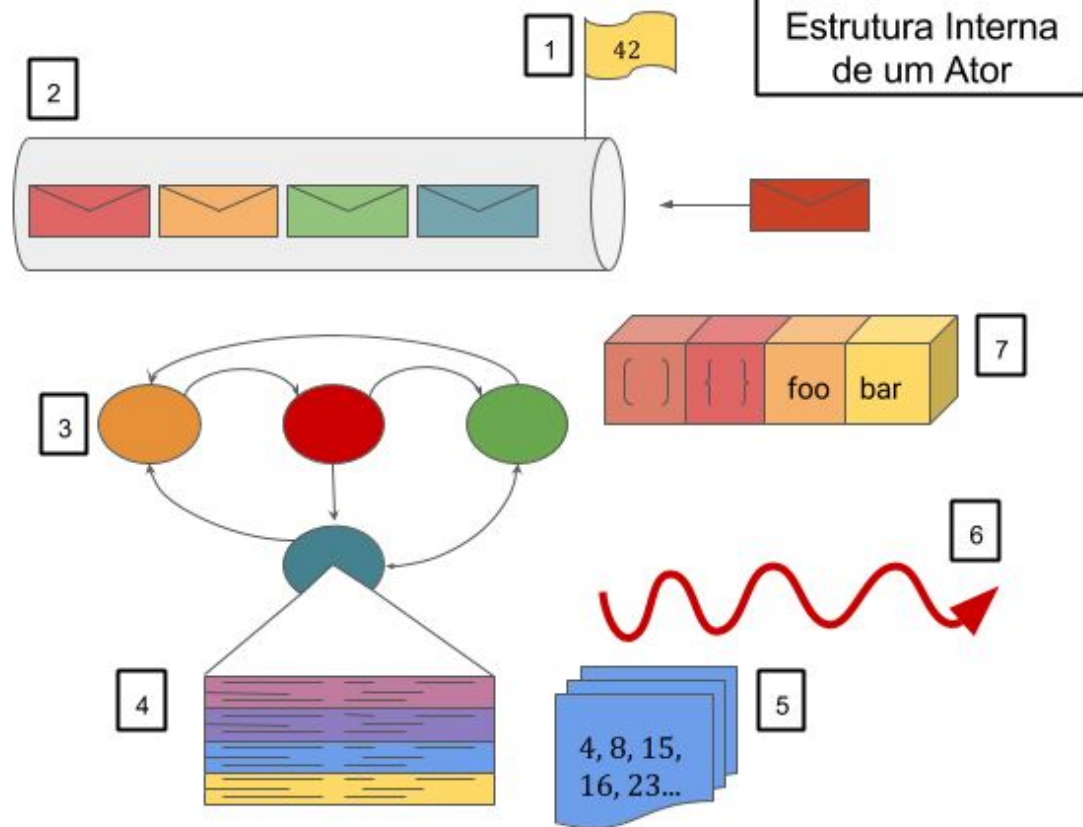
# O Modelo de Atores

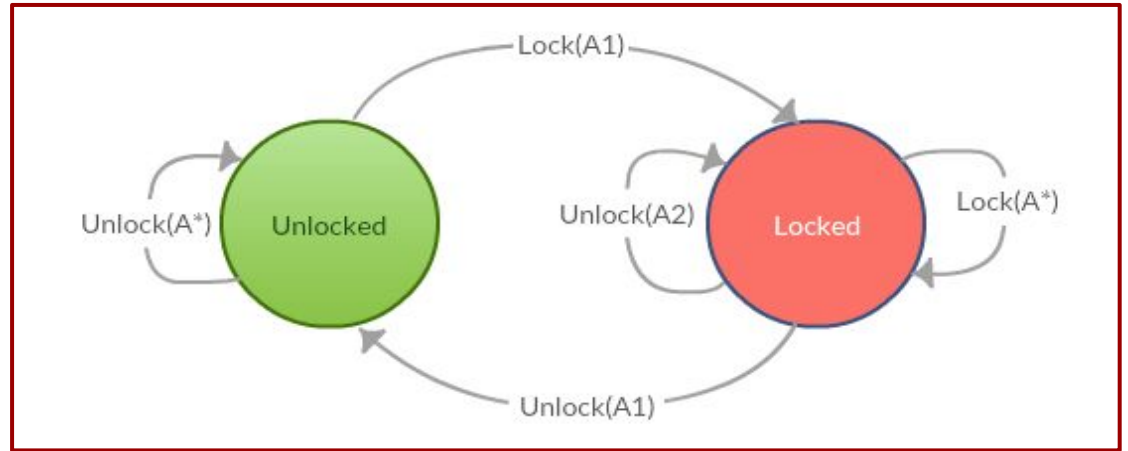
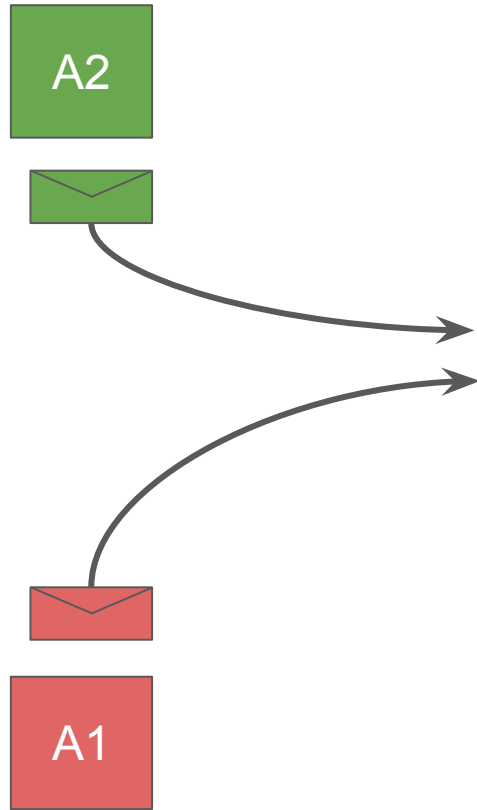
- Um modelo matemático para descrever a **computação concorrente**
- Um método alternativo para lidar com **recursos compartilhados**
- Uma forma alternativa de modelar problemas clássicos de concorrência e suas soluções, como o **Jantar dos filósofos**
- A base para a biblioteca **Akka**, que permite criar sistemas concorrentes baseados em atores que sejam: **assíncronos**, **imutáveis** e **orientados a mensagens**

# O Modelo de Atores

Foi idealizado por Carl Hewitt em 1973 [1]. Atores são unidades funcionais com processamento, armazenamento e comunicação e que podem [2]:

- Tomar decisões locais e alterar variáveis internas
- Mandar mensagens para si, ou para outros atores
- Criar novos atores
- Mudar seu comportamento interno



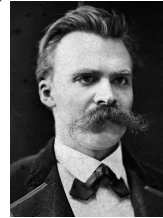
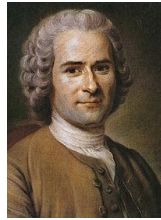
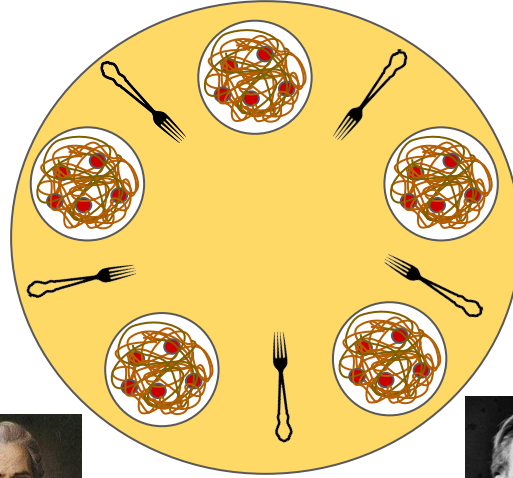
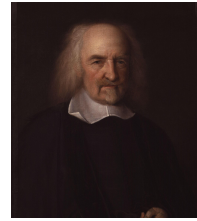
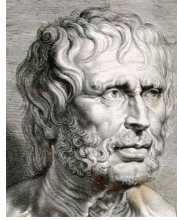


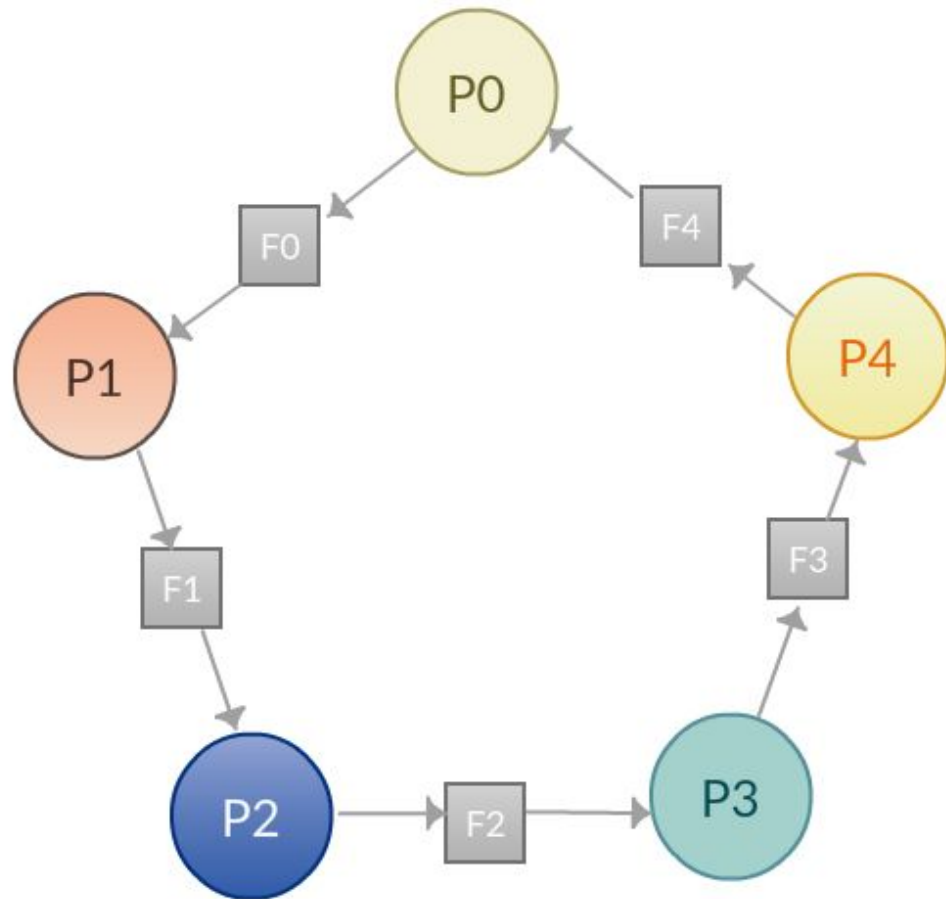
# O Jantar dos Filósofos

O problema pode ser descrito como:

- Cinco filósofos estão dispostos em uma mesa circular, com um prato de espaguete na frente de cada um
- Os filósofos alternam entre pensar e comer
- Para comer um filósofo precisa de dois garfos ao mesmo tempo
- Existe apenas um garfo entre cada prato
- Filósofos adjacentes não podem comer ao mesmo tempo







# O Jantar dos Filósofos

Uma solução válida para o problema deve verificar as seguintes propriedades:

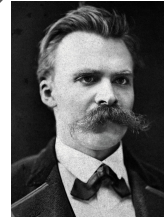
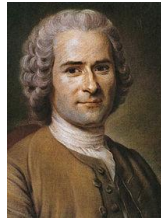
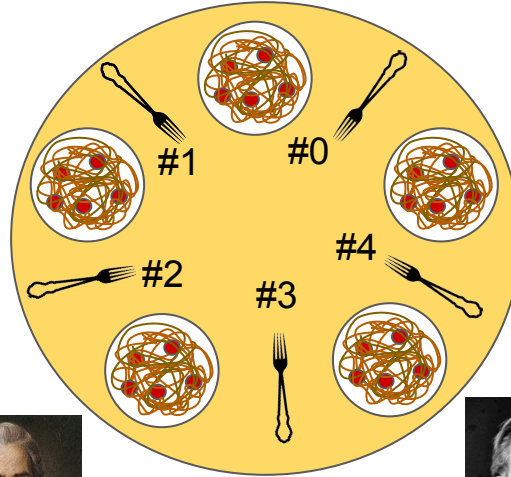
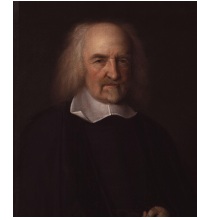
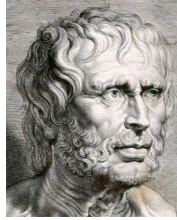
- Fornecer um comportamento para os filósofos
- Possibilitar que todos os filósofos consigam comer
- Evitar que os filósofos morram de fome
- Coordenar o acesso aos recursos compartilhados
- Ser justa, de forma que todos comam em média o mesmo e esperem uma quantidade equivalente de tempo

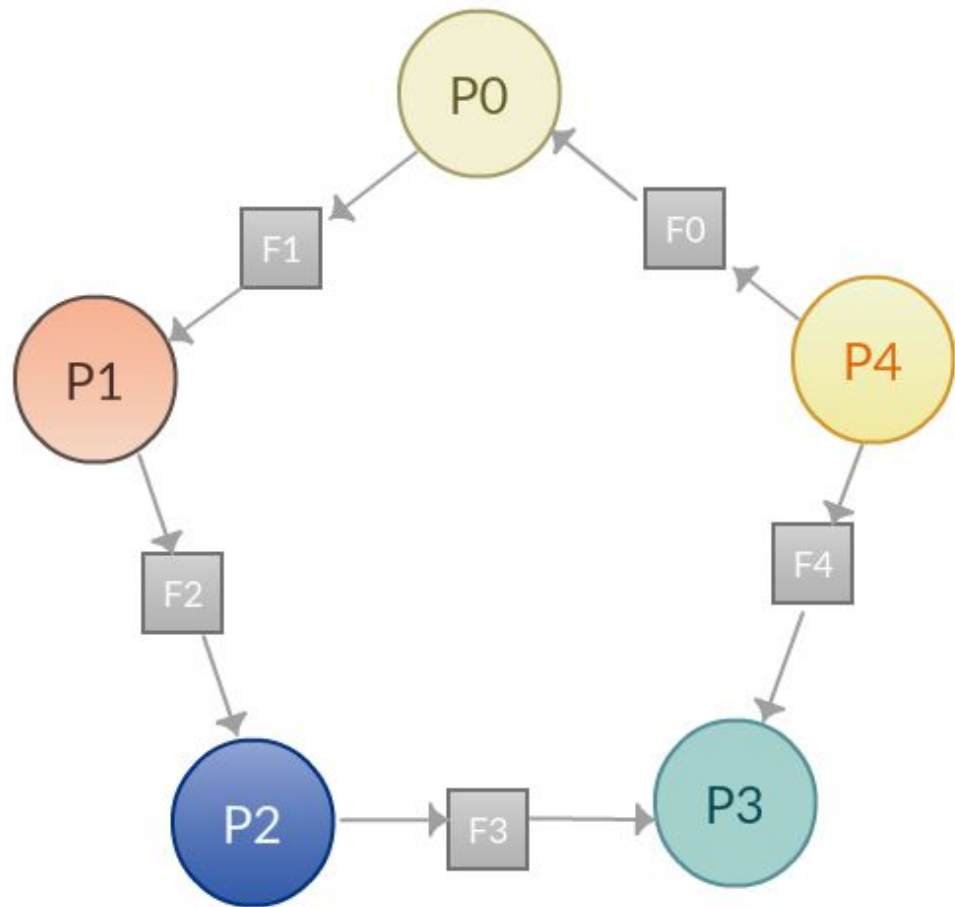
Os algoritmos Hierarquia de Recursos [3] [4], Waiter [5] e Chandy-Misra [6] verificam as propriedades apresentadas.

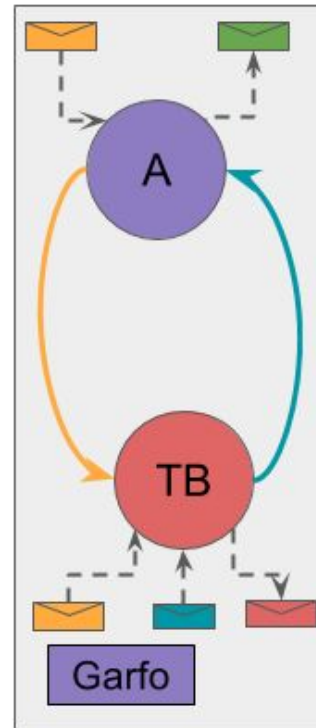
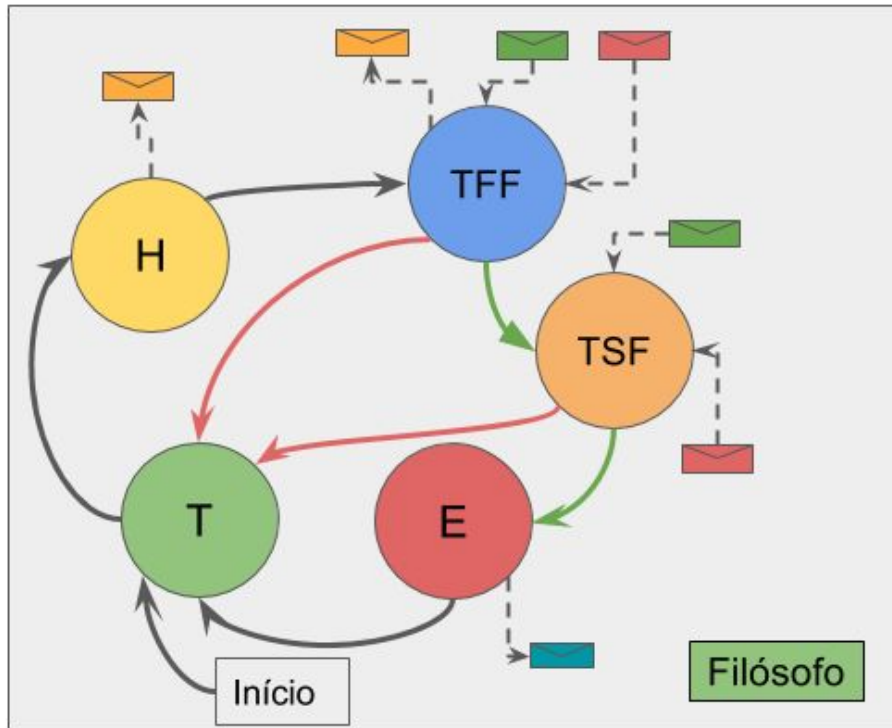
# Algoritmos

# Hierarquia de Recursos

1. Pense por uma quantidade fixa de tempo então sinta fome
2. Ao sentir fome tente pegar o garfo de menor índice que estiver ao seu alcance, caso consiga prossiga para o próximo passo, caso contrário volte para o passo 1
3. Tente pegar o garfo restante, caso consiga prossiga para o próximo passo, caso contrário solte o garfo que tem em mãos e volte para o passo 1
4. Comece a comer por uma quantidade fixa de tempo
5. Largue o garfo de menor índice
6. Largue o garfo restante
7. Volte para o passo 1









# Waiter

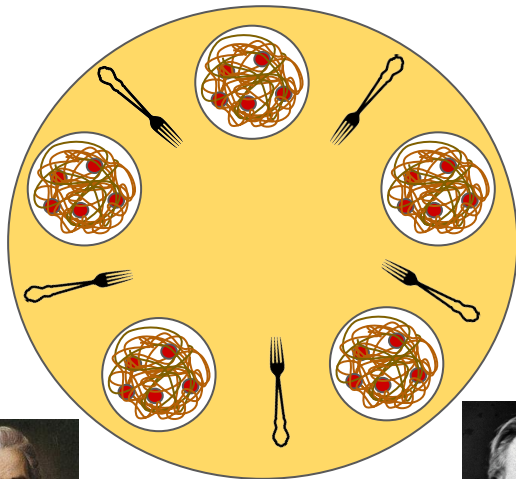
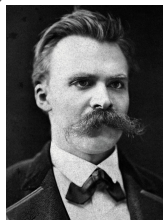
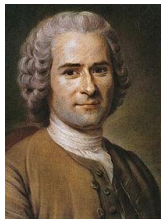
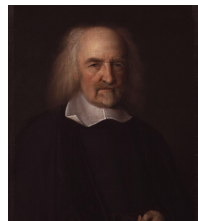
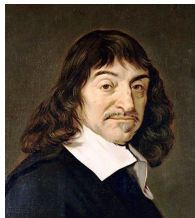
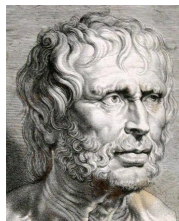
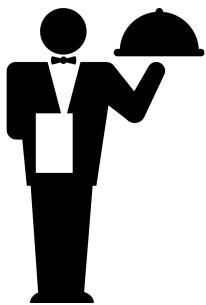
## Comportamento do garçom

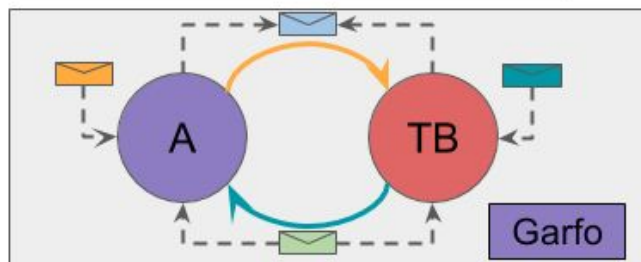
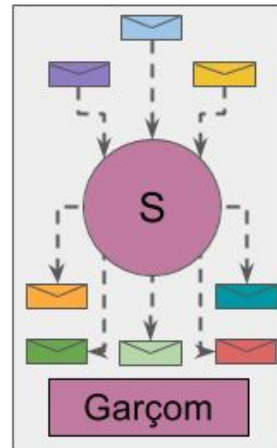
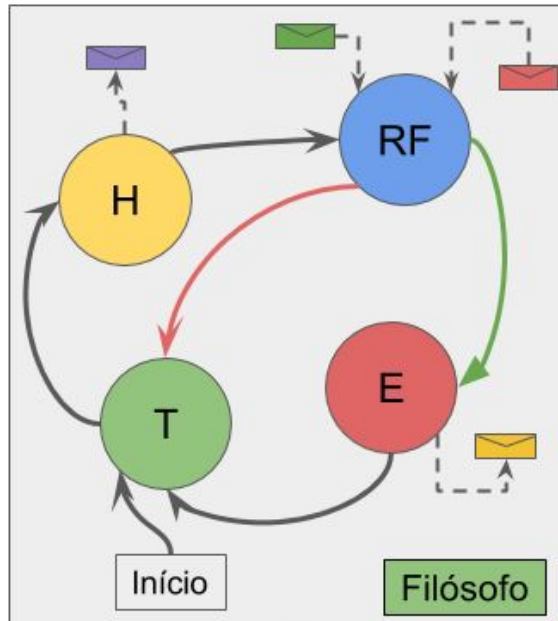
1. Espere até receber uma nova mensagem de um filósofo Pi
2. Ao receber uma nova mensagem verifique o tipo dela
3. Se for do tipo 'requisição':
  - a. Verifique se os garfos de Pi estão livres
  - b. Em caso positivo autorize Pi a comer, caso contrário impeça que Pi coma
  - c. Volte para o passo 1
4. Se for do tipo 'liberação':
  - a. Libere os garfos de Pi
  - b. Volte para o passo 1

# Waiter

## Comportamento dos filósofos

1. Pense por uma quantidade fixa de tempo então sinta fome
2. Ao sentir fome mande uma mensagem do tipo 'requisição' para o garçom e aguarde sua resposta por um período de tempo máximo, caso este tempo se esgote volte para o passo 1
3. Se for autorizado pelo garçom vá para o próximo passo, caso seja impedido volte para o passo 1
4. Comece a comer por uma quantidade fixa de tempo
5. Mande uma mensagem do tipo 'liberação' para o garçom
6. Volte para o passo 1







### Legenda


T - Pensando  
 H - Faminto  
 RF - Requisitando o par de garfos  
 E - Comendo  
 S - Servindo  
 A - Disponível  
 TB - Ocupado


 Requisita o par de garfos


 Recebe o par de garfos

 Garfos ocupados

 Libera o par de garfos

 Pergunta estado do garfo

 Recebe estado do garfo

 Requisita o garfo

 Libera o garfo

# Chandy-Misra

O algoritmo funciona a partir de:

- Um conjunto de variáveis booleanas
- Um conjunto de regras definidas
- Um grafo de precedência  $H$
- Um conjunto de condições iniciais

$fork_u(f_i)$ : O filósofo  $u$  detem o garfo  $f_i$

$reqf_u(f_i)$ : O filósofo  $u$  detem um *request token* para o garfo  $f_i$

$dirty_u(f_i)$ : O garfo  $f_i$  está em posse do filósofo  $u$  e está sujo

$thinking_u/hungry_u/eating_u$ : O filósofo  $u$  está pensando/faminto/comendo

- Filósofo  $u$  requisita o garfo  $f_i$ :

$hungry_u, reqf_u(f_i), \neg fork_u(f_i) \Rightarrow$

Mande o *request token* para o filósofo que detem  $f_i$ ;

$reqf_u(f_i) := false$

- Filósofo  $u$  libera o garfo  $f_i$ :

$\neg eating_u, reqf_u(f_i), dirty_u(f_i) \Rightarrow$

Mande o garfo para o filósofo que enviou o *request token* de  $f_i$ ;

$dirty_u(f_i) := false$ ;

$fork_u(f_i) := false$

- Filósofo  $u$  recebe um *request token* para o garfo  $f_i$ :

Ao receber um *request token* para o garfo  $f_i \Rightarrow$

Mande o garfo para o filósofo que enviou o *request token* de  $f_i$ ;

$reqf_u(f_i) := true$

- Filósofo  $u$  recebe o garfo  $f_i$ :

Ao receber o garfo  $f_i \Rightarrow$

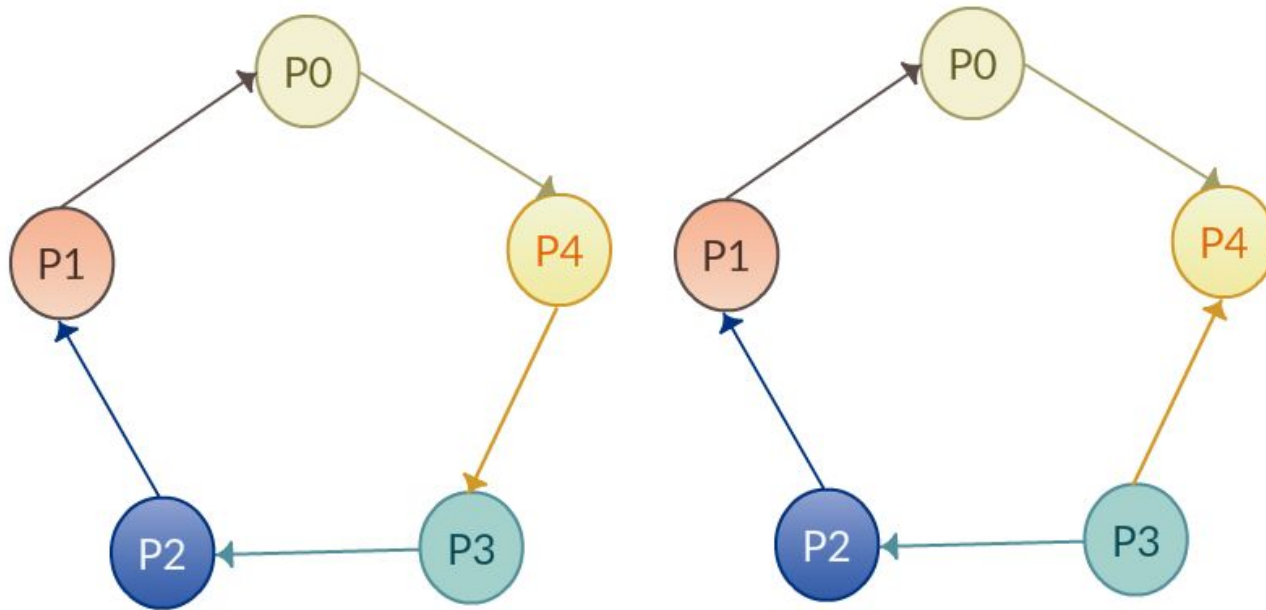
$fork_u(f_i) := true$ ;

$\neg dirty_u(f_i)$

- Filósofo  $u$  está usando o garfo  $f_i$ :

$eating_u, fork_u(f_i) \Rightarrow$

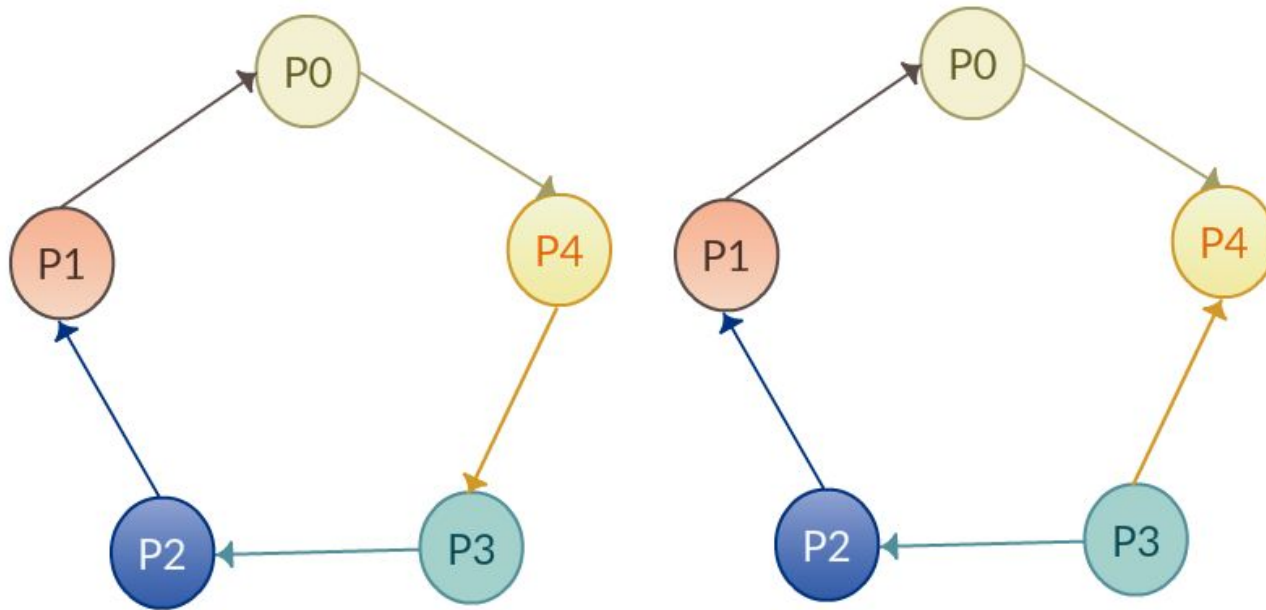
$dirty_u(f_i) := true$



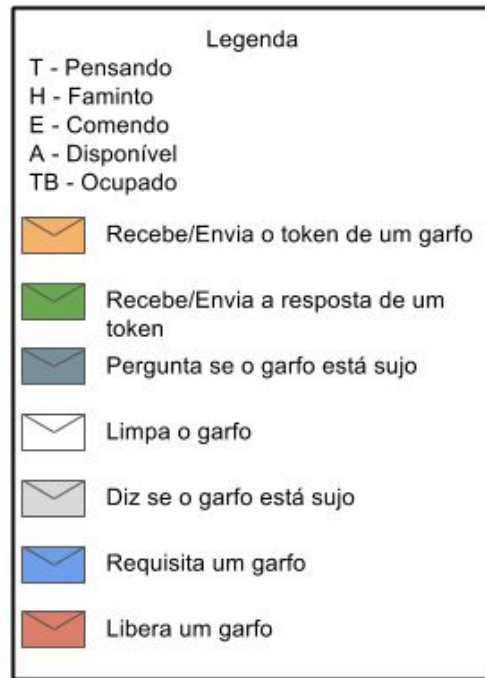
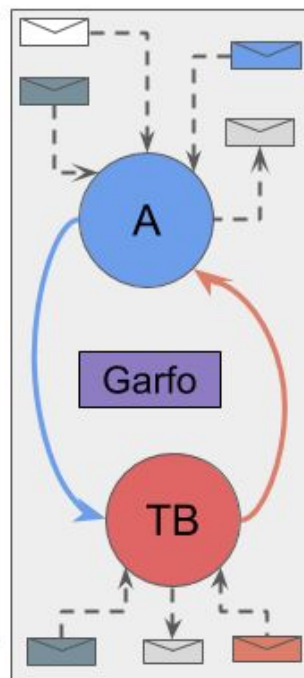
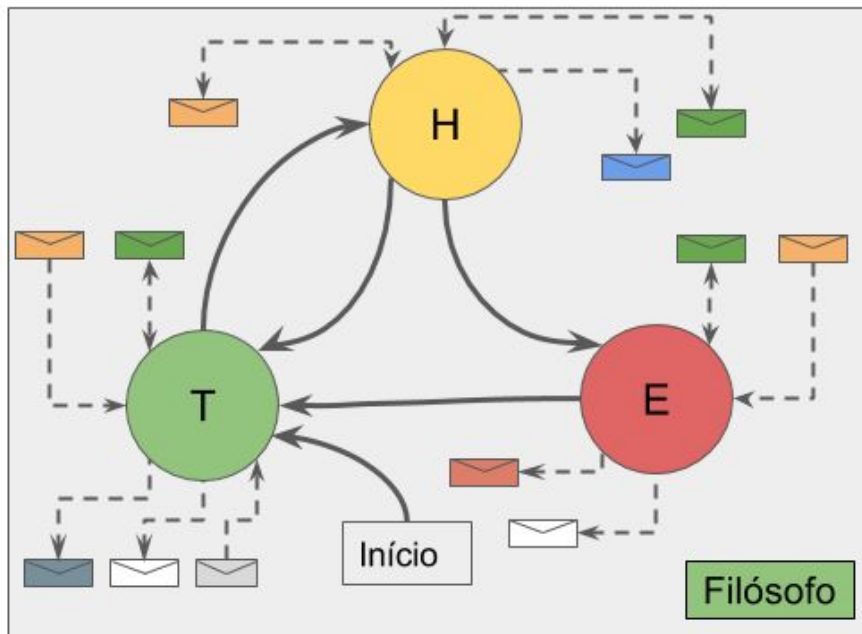
$P_u \rightarrow P_v$  sse:

- $P_u$  detêm a posse do garfo compartilhado entre ele e  $P_v$ , ou
- $P_v$  detêm o garfo, e o garfo está sujo, ou
- O garfo está indo de  $P_v$  para  $P_u$





1. Todos os garfos estão inicialmente sujos
2. Para cada garfo  $F_i$  que é dividido entre dois filósofos um deles segurará o request token de  $F_i$  e o outro o garfo
3. O grafo de precedência  $H$  é acíclico



# Simulação e Resultados

## Especificações do computador:

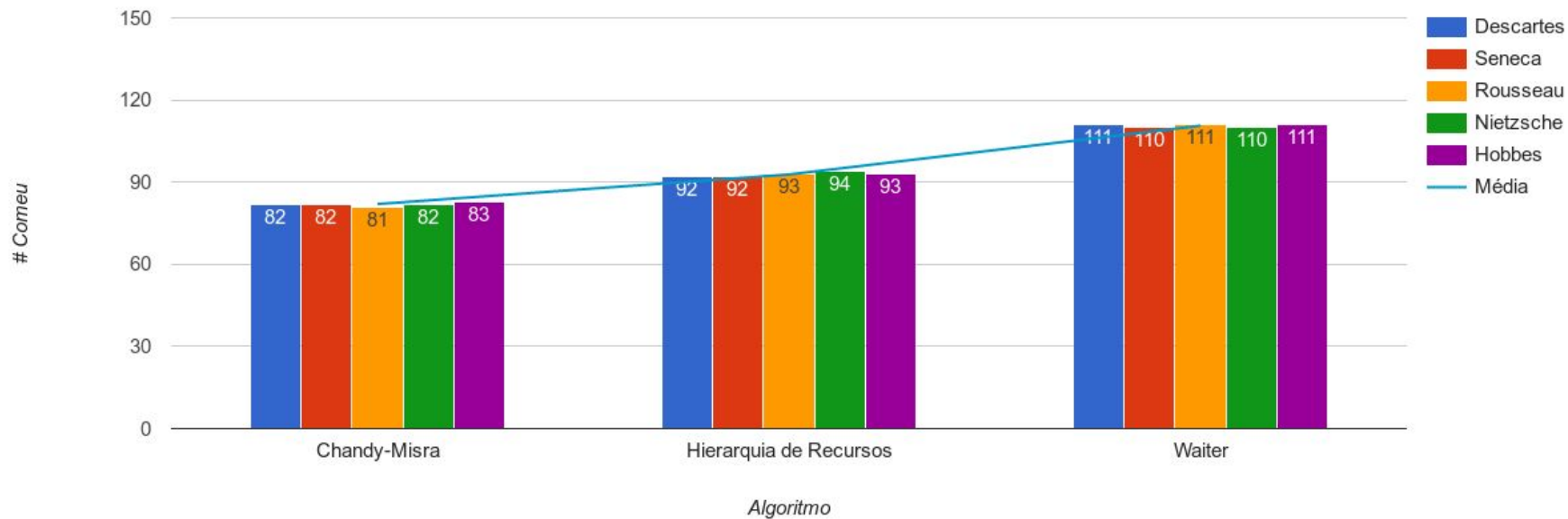
- Lenovo Thinkpad T420
- Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz com 4 núcleos
- 8gb RAM
- SSD 250gb Kingston UV400
- Ubuntu 16.04 LTS Mate
- Kernel Linux 4.4.0-47-generic (x86\_64)

## Especificação dos softwares:

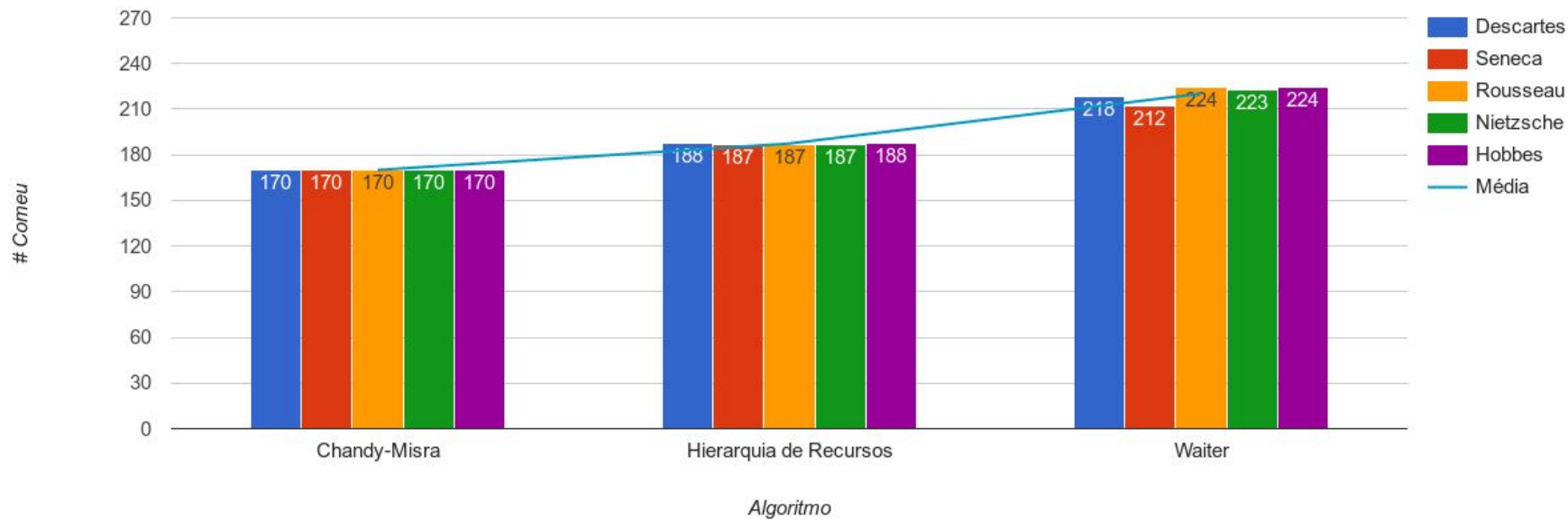
- Scala version 2.11.8
- Java version 1.8.0\_101
- Akka version 2.4.8
- Sbt version 0.13.12

Quantidade de vezes que os filósofos comeram

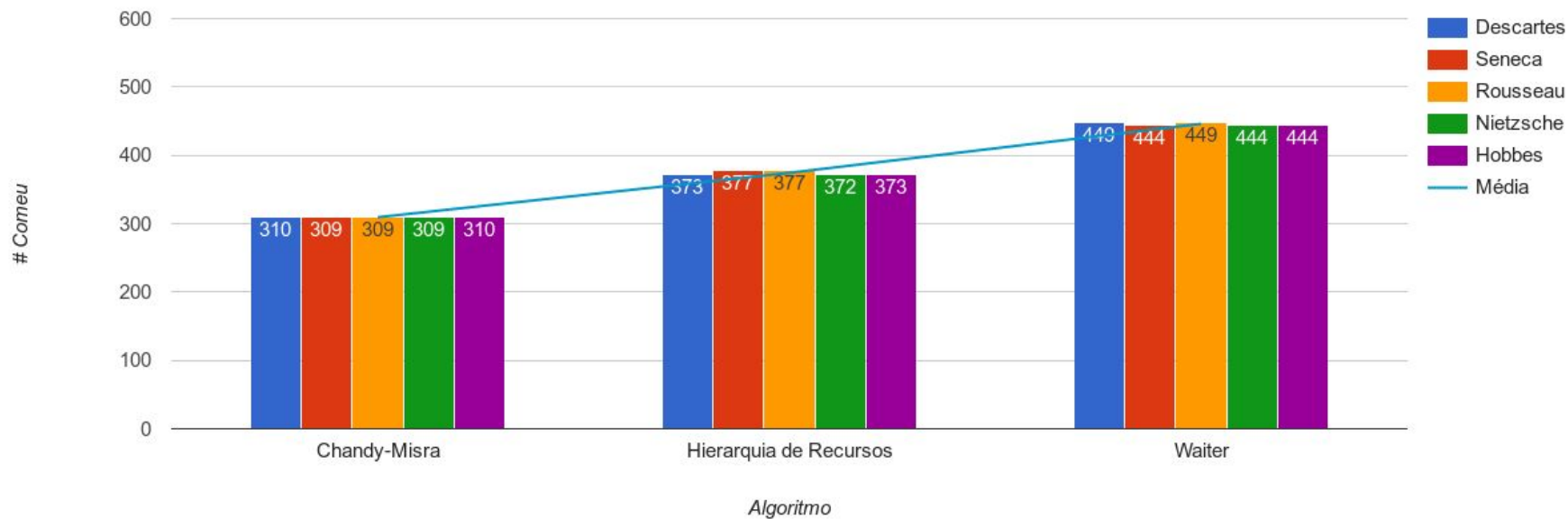
## 30 minutos de simulação



## 1 hora de simulação



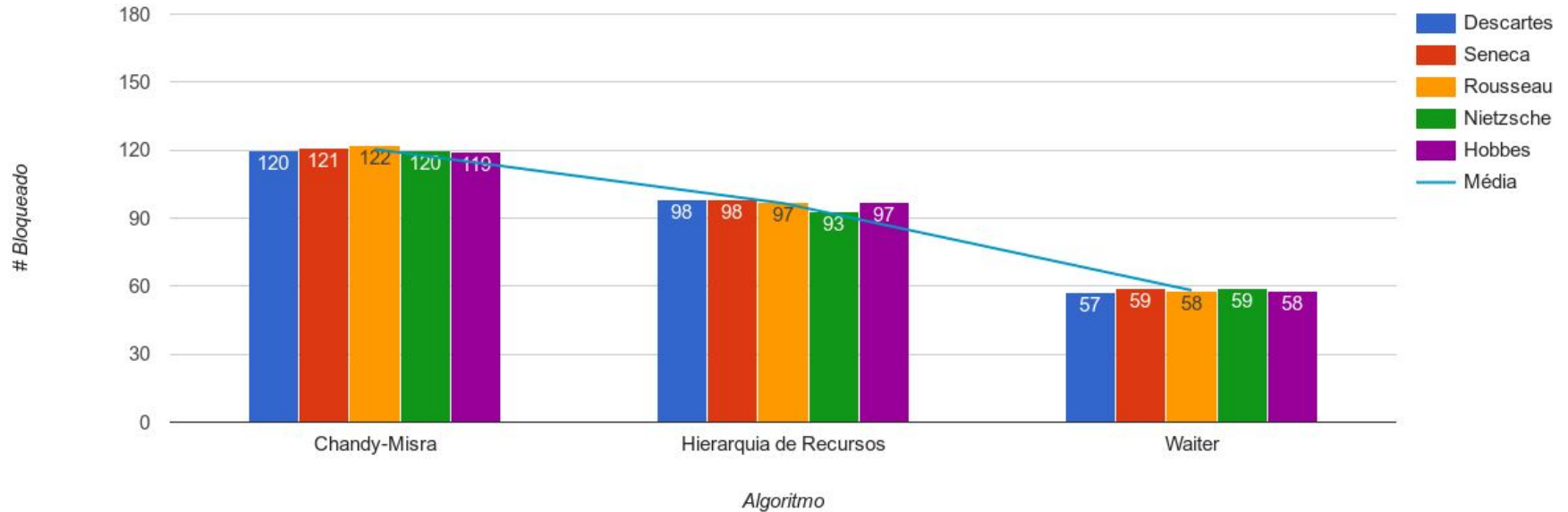
## 2 horas de simulação



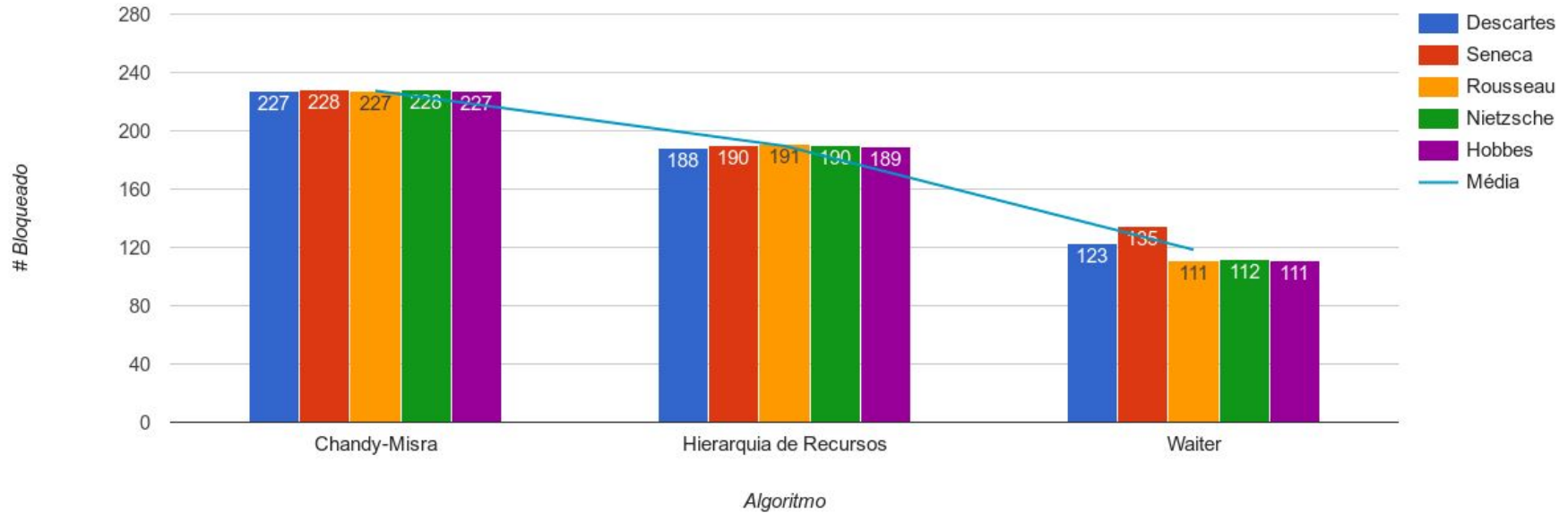
Quantidade de vezes que os filósofos foram bloqueados de comer



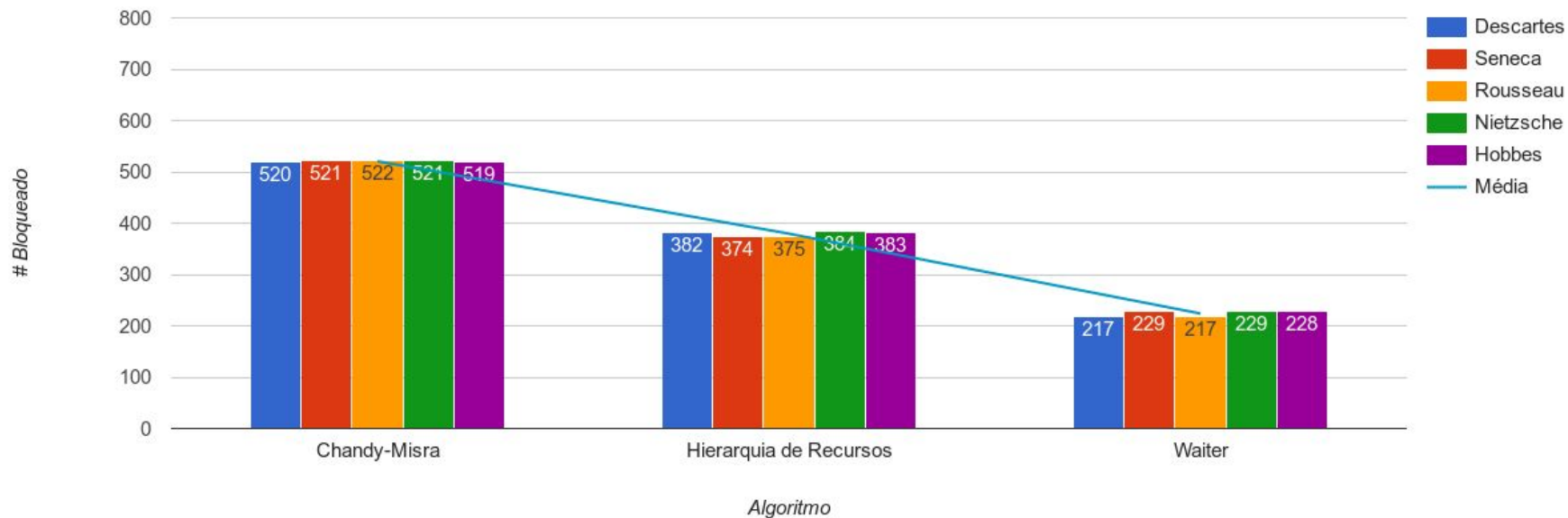
## 30 minutos de simulação



## 1 hora de simulação

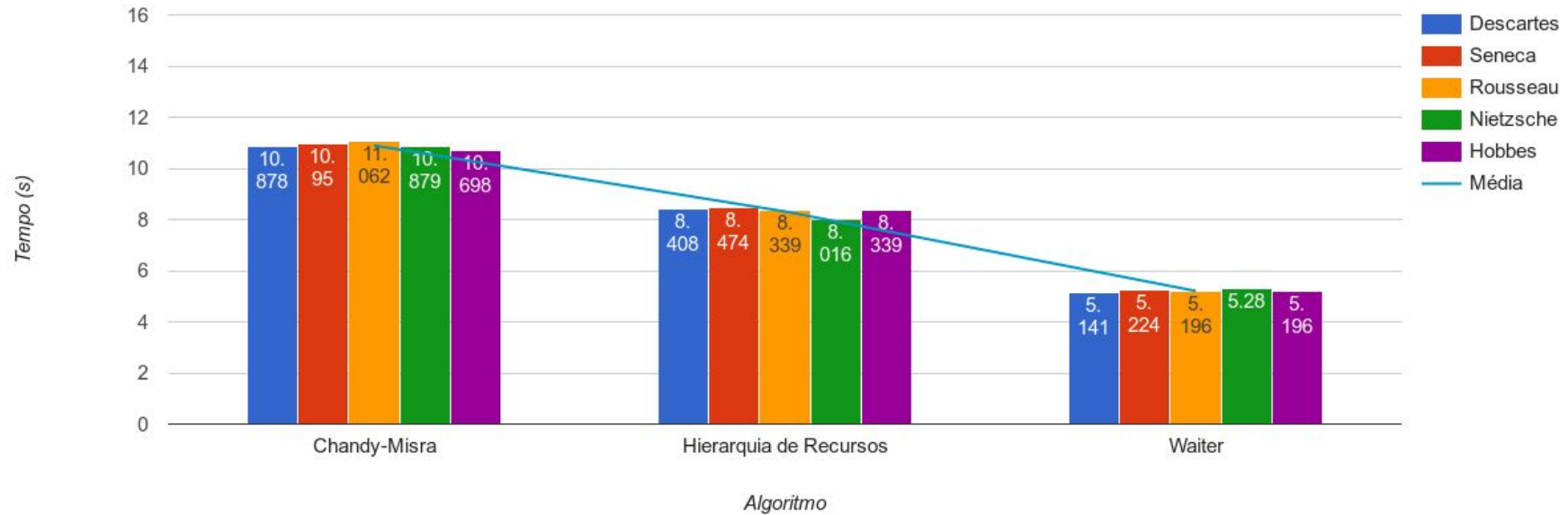


## 2 horas de simulação

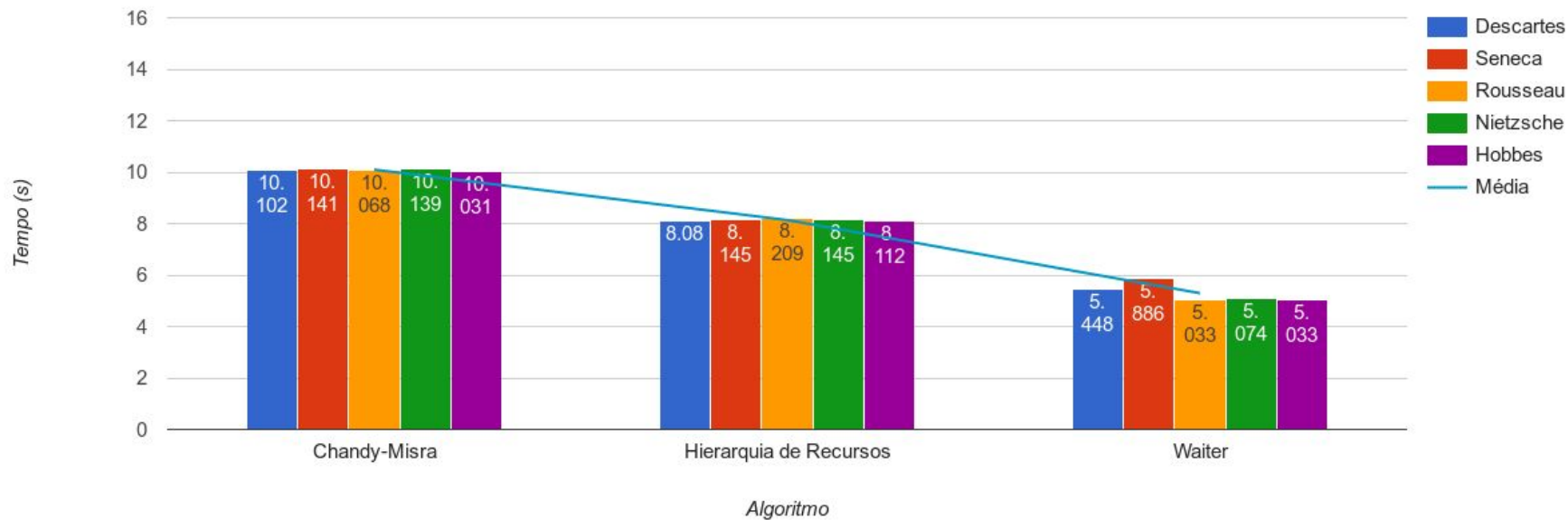


Tempo médio que um filósofo precisa esperar para comer

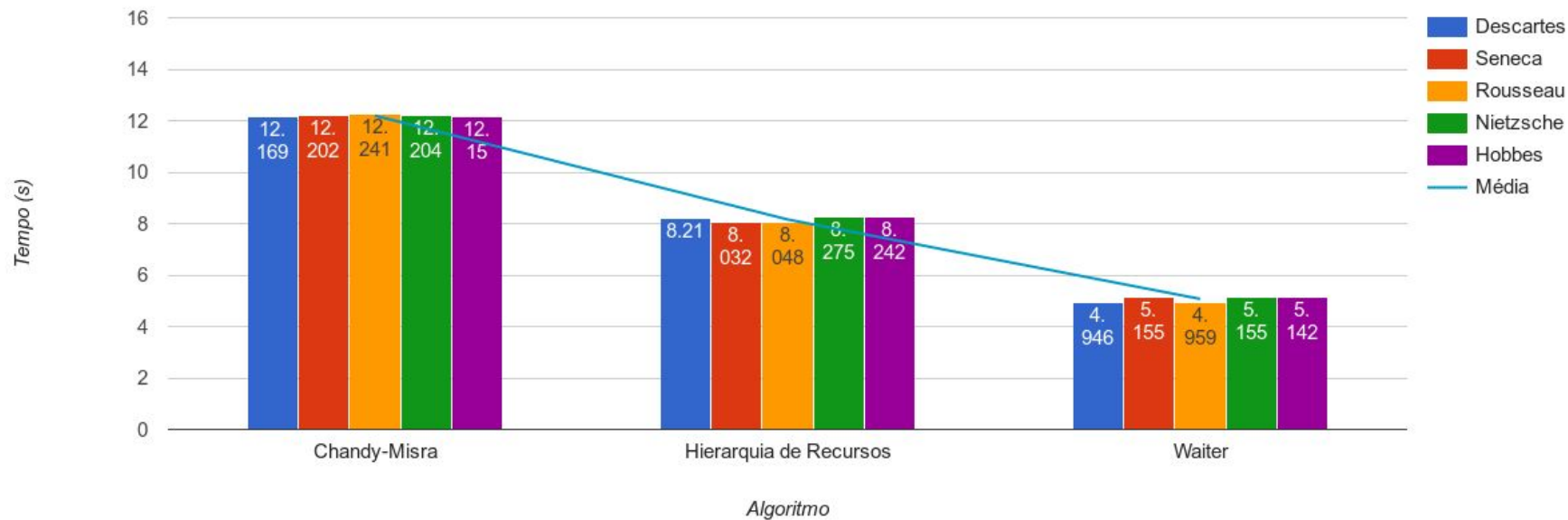
## 30 minutos de simulação



## 1 hora de simulação



## 2 horas de simulação



# Conclusão

Através da modelagem do problema e das simulações podemos concluir que:

- Cada filósofo, garfo e garçom é um ator, e seu comportamento é definido como um máquina de estados
- Cada um dos algoritmo define um comportamento para os filósofos
- Todos os filósofos se alimentaram, isso evidencia a ausência de impasses, como deadlocks e livelocks
- Um tempo de espera quase uniforme indica que nenhum filósofo morreu de fome
- A ausência de impasses indica que o acesso ao recurso foi coordenado de forma correta e simétrica entre os filósofos
- Cada algoritmo coordena o acesso aos recursos compartilhados de formas diferentes



# Referências

- [1] Carl Hewitt et al. A universal modular actor formalism for artificial intelligence. IJCAI'73, 1973, pp. 235–245.
- [2] Gul A. Agha. Actors: a Model of Concurrent Computation in Distributed Systems. MIT Artificial Intelligence Laboratory, 1985, pp. 12-31.
- [3] Edsger W. Dijkstra. Hierarchical ordering of sequential processes. EWD310, p. 7
- [4] Edsger W. Dijkstra. Two starvation-free solutions of a general exclusion problem. EWD625
- [5] C.A.R Hoare. Communication Sequential Process. Prentice Hall; 1 edition, 1985., pp. 57-61.
- [6] K.M. Chandy et al. The drinking philosopher problem. ACM Transactions on Programming Languages and Systems, Vol. 6, No. 4, 1984, pp. 632–646

Para mais informações sobre o trabalho, consulte:

<https://github.com/flsouto/mac0499>

ou envie um email para [flsouto@gmail.com](mailto:flsouto@gmail.com)

Obrigado!

# Um estudo do Modelo de Atores aplicado à concorrência de recursos

Fellipe Souto Sampaio

Orientadora: Prof. Dra. Ana Cristina Vieira de Melo

MAC0499 - Trabalho de Formatura Supervisionada  
Instituto de Matemática e Estatística - Universidade de São Paulo

16 de Novembro de 2016