

# Replicação de Máquinas de Estado Baseada em Prioridade com PRaft

Paulo Rogério de Pinho Filho\*,  
Luciana de Oliveira Rech\*,  
Lau Cheuk Lung\*,  
Miguel Correia\*\*,  
**Lásaro Camargos\*\*\***

\*Departamento de Informática e Estatística, Universidade Federal de Santa Catarina,, Brasil

\*\*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal

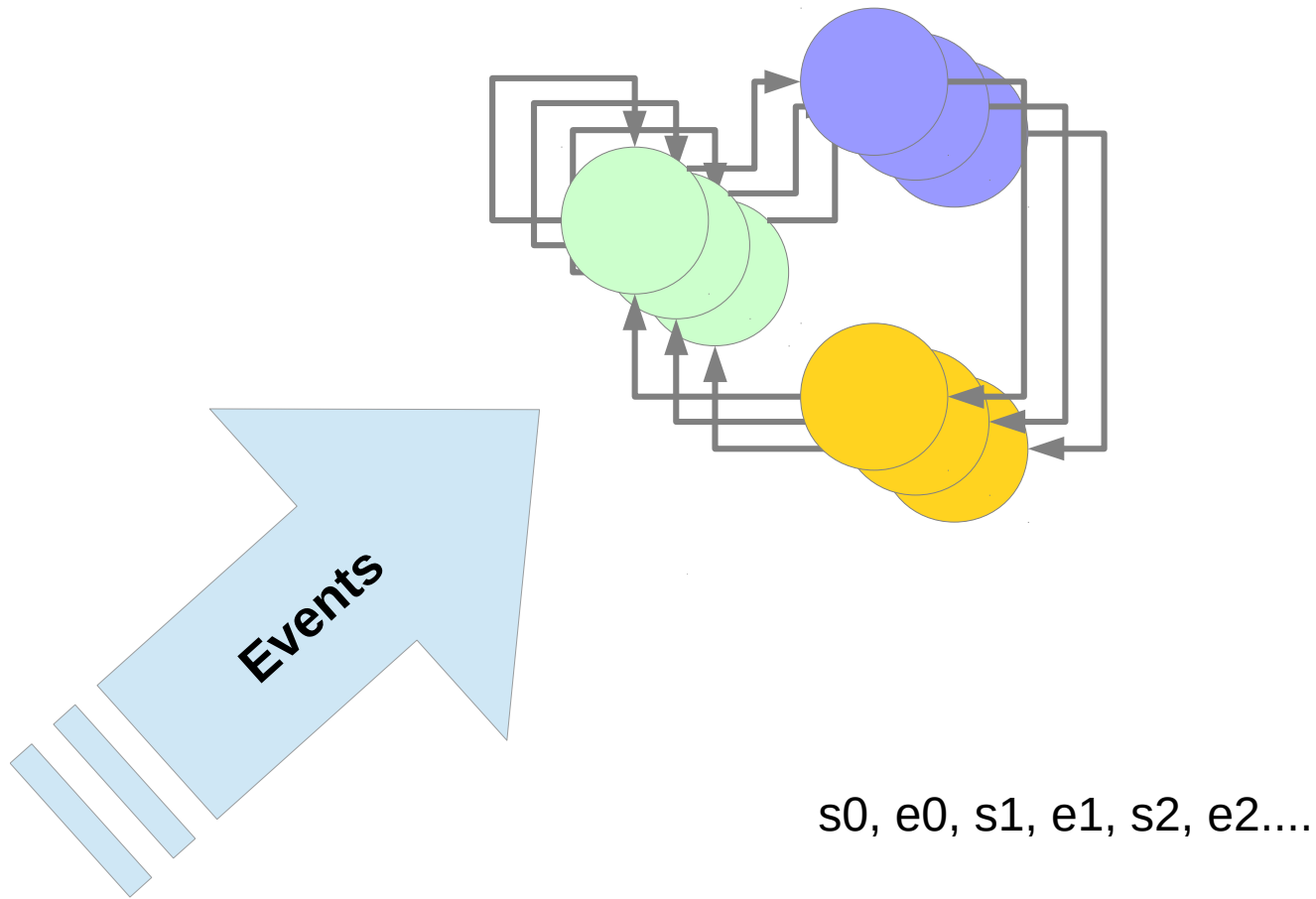
\*\*\*Faculdade de Computação, Universidade Federal de Uberlândia, Brasil



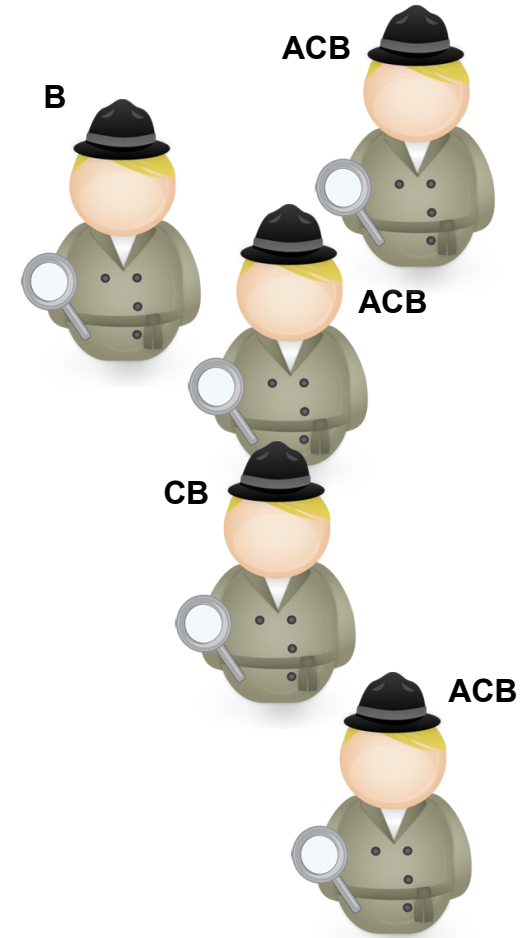
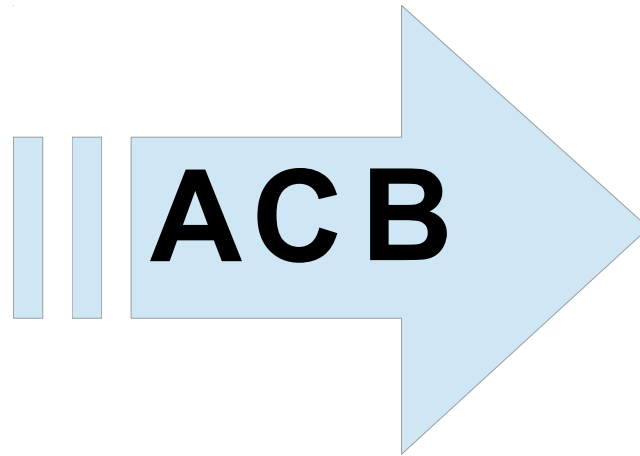
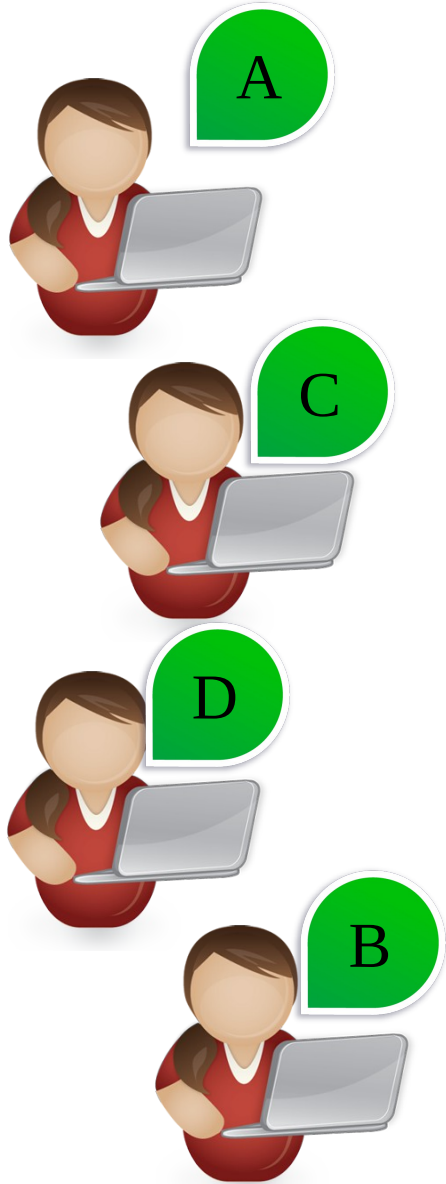
UNIVERSIDADE FEDERAL  
DE SANTA CATARINA



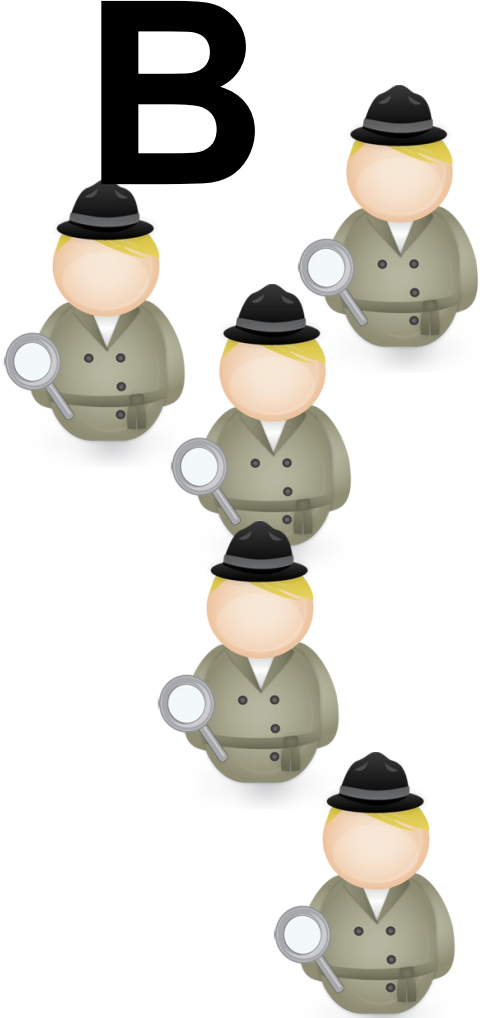
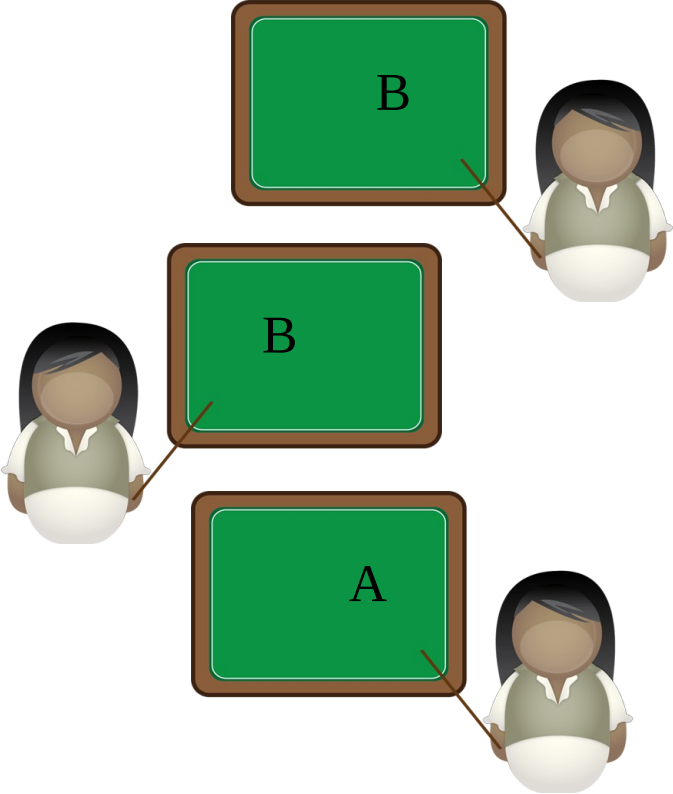
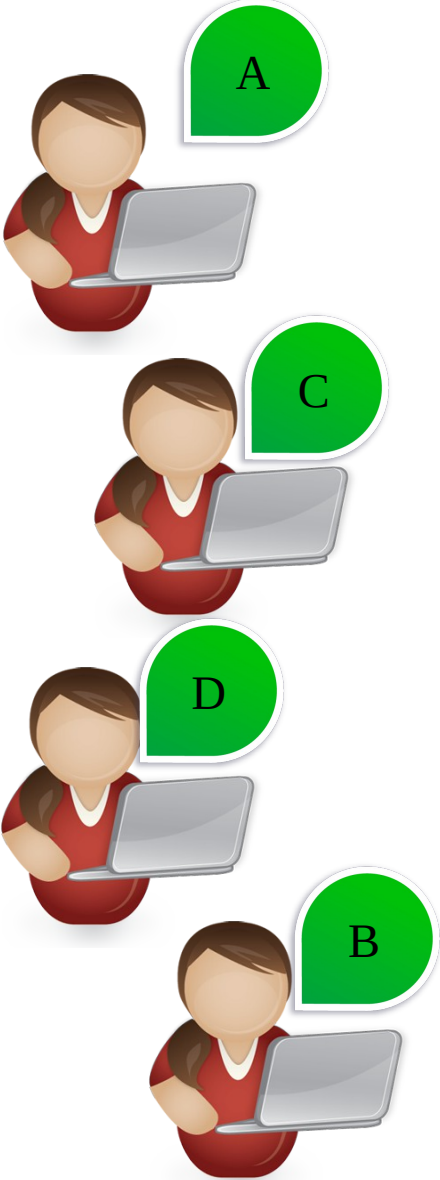
# Replicação de Máquinas de Estados



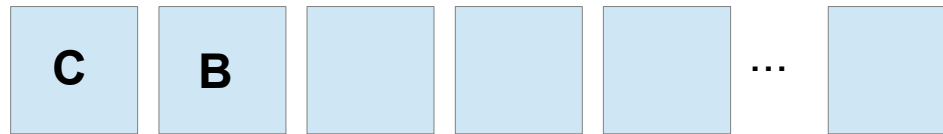
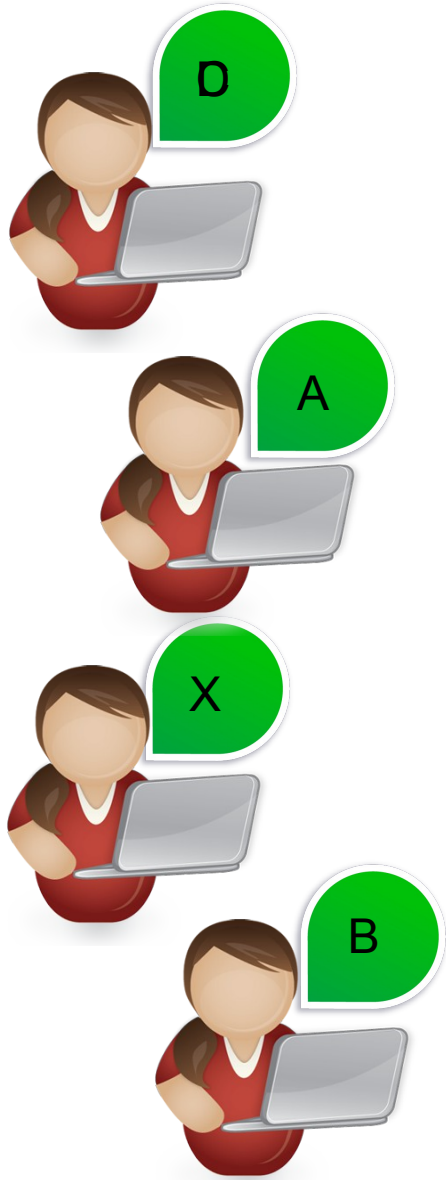
# Difusão Totalmente Ordenada



# Consenso

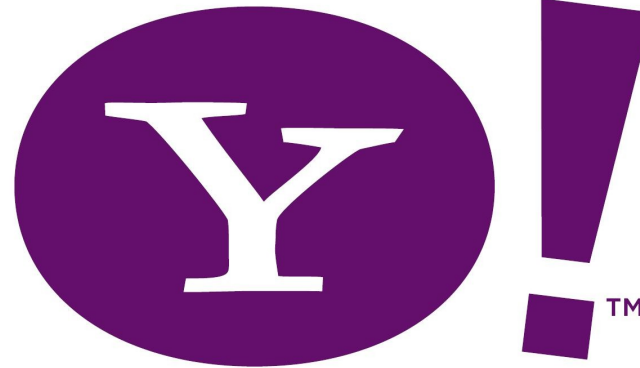
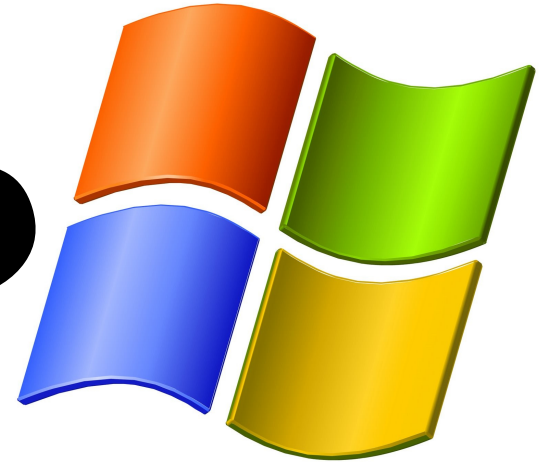


# De Consensus A ABCast



Google

Paxos



- Google Chubby
- Microsoft Boxwood
- Apache Zookeeper

- LibPaxos
- OpenReplica/Concord
- Apache Kafka
- Doozer
- TreoDB

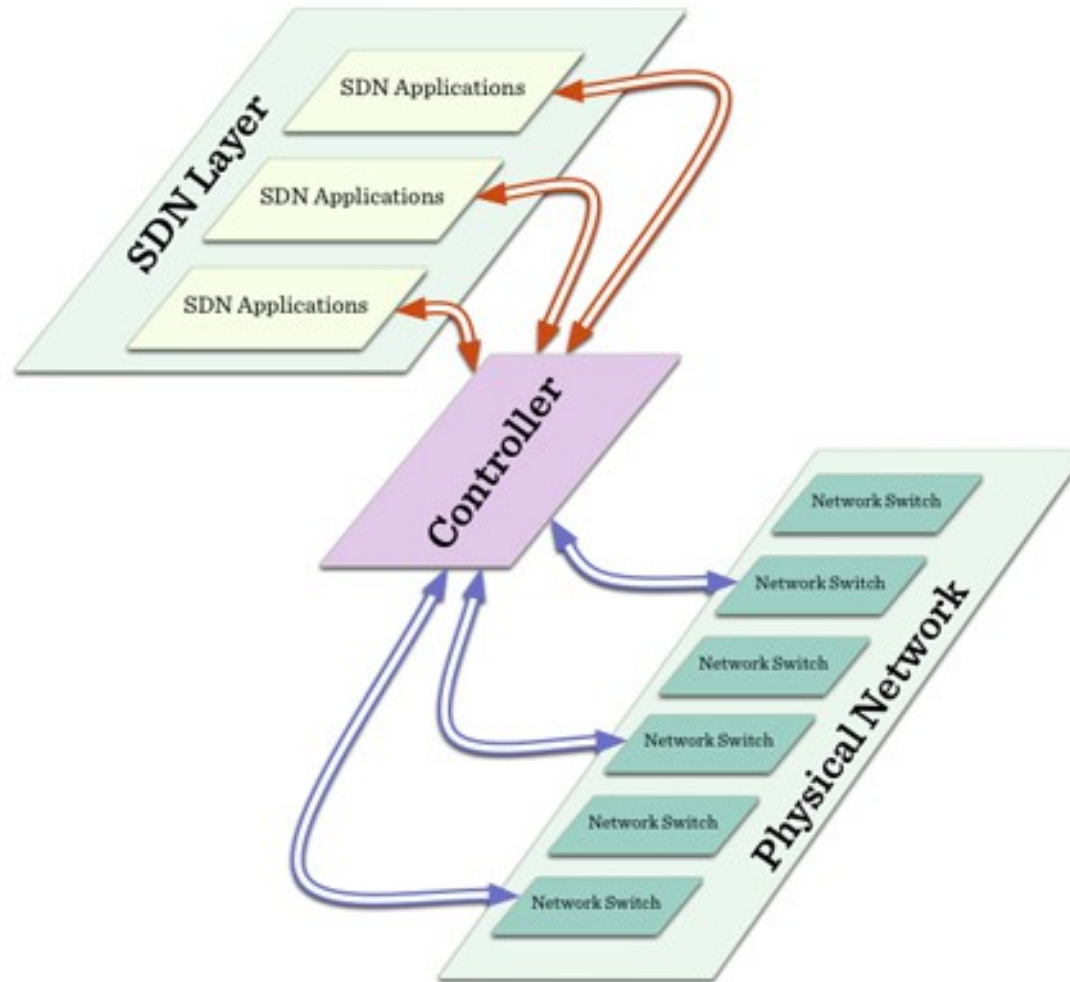
- EPaxos
- Cheap Paxos
- Ring Paxos

# Prioridades

Requisições de prioridade mais alta executam antes das de prioridade mais baixa.



# Software Defined Networks



# Problema

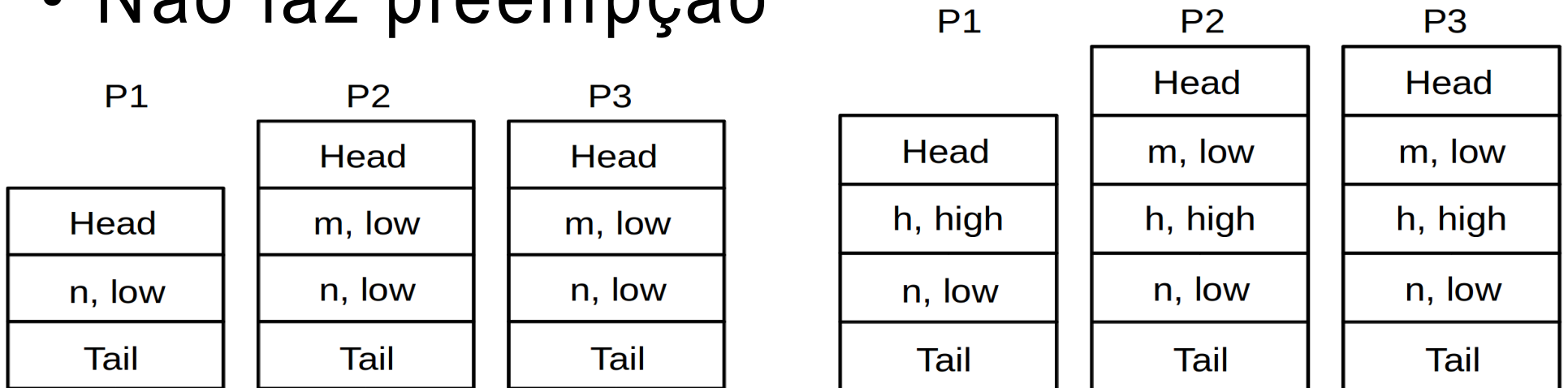
- Replicação de Máquinas de Estado Baseada em Prioridades.  
(*Priority Based State Machine Replication – PB-SMR*)
- Algoritmos existentes não lidam bem com prioridades.
  - Detectores de falhas
  - Sincronismo
  - Serviço de consenso
  - Complexidade exponencial de mensagens

# PriTO

- Modelo síncrono
- Não tolera faltas

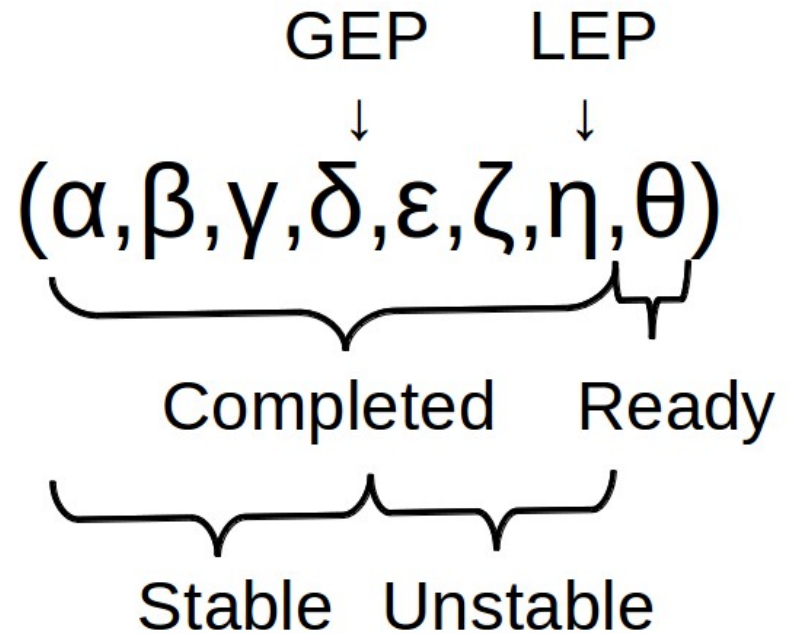
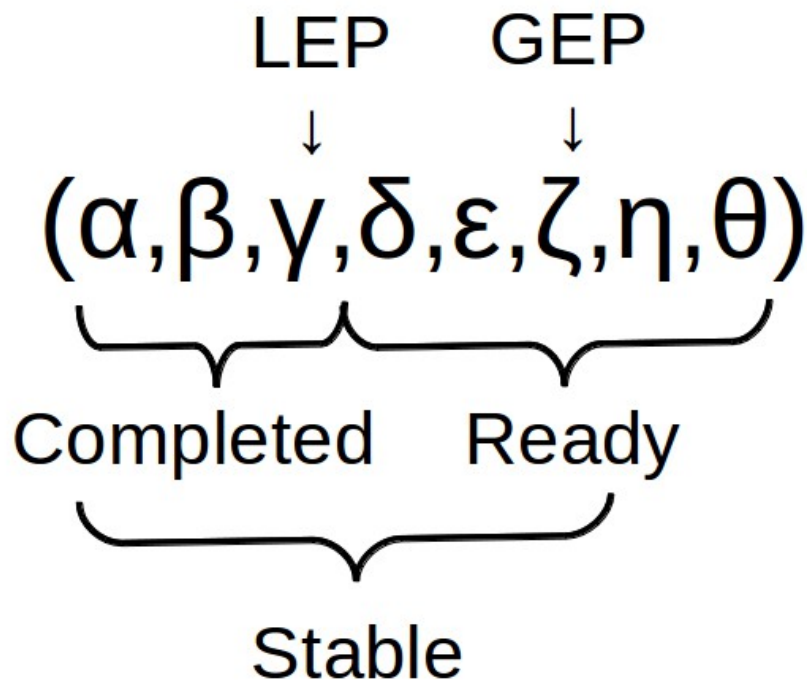
# Rodrigues

- *Virtual Synchrony*
- Detector de faltas
- Tolera faltas de crash
- Não faz preempção



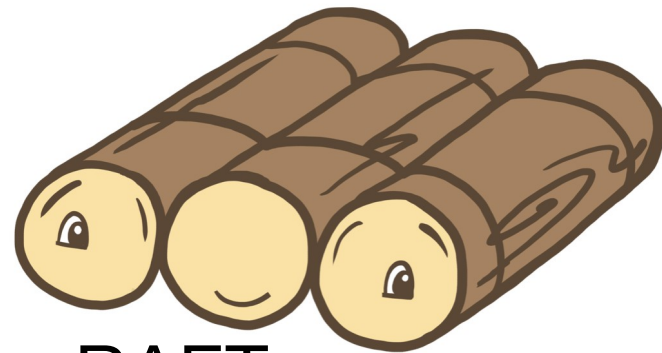
# Wang

- *Generic Agreement Framework*
- Detector de Faltas
- Tolera faltas de *crash*



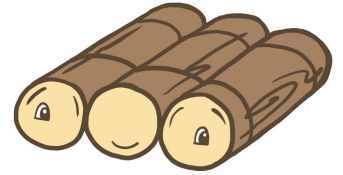
# Objetivo

- Resolver PB-SMR de forma simples
- Atomic Broadcast
- Incorpora Detecção de Falhas
- Algoritmo Intuitivo



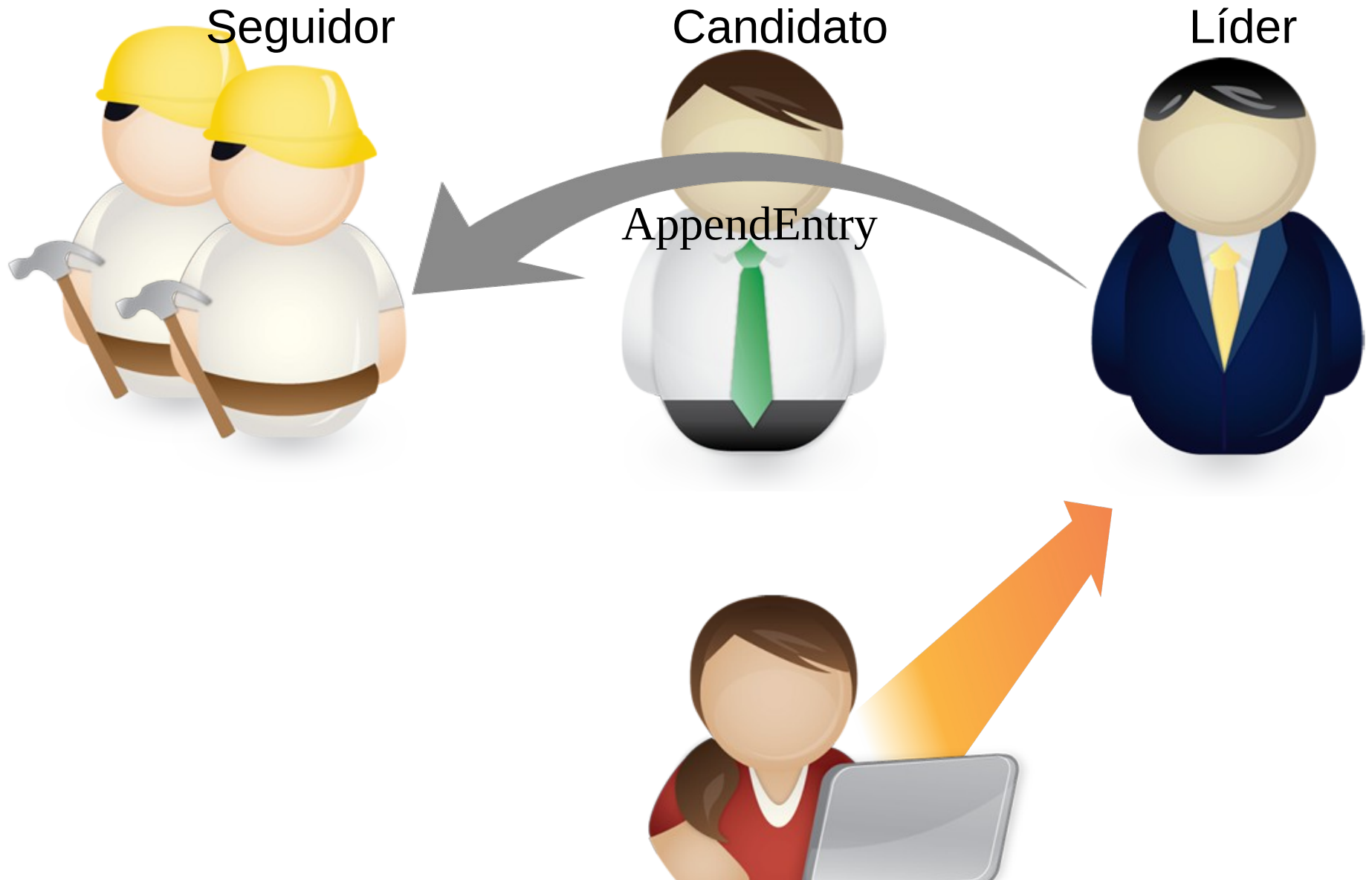
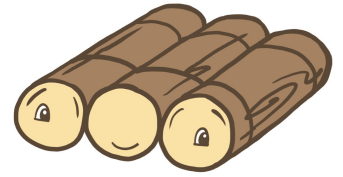
RAFT

# Modelo de Sistema

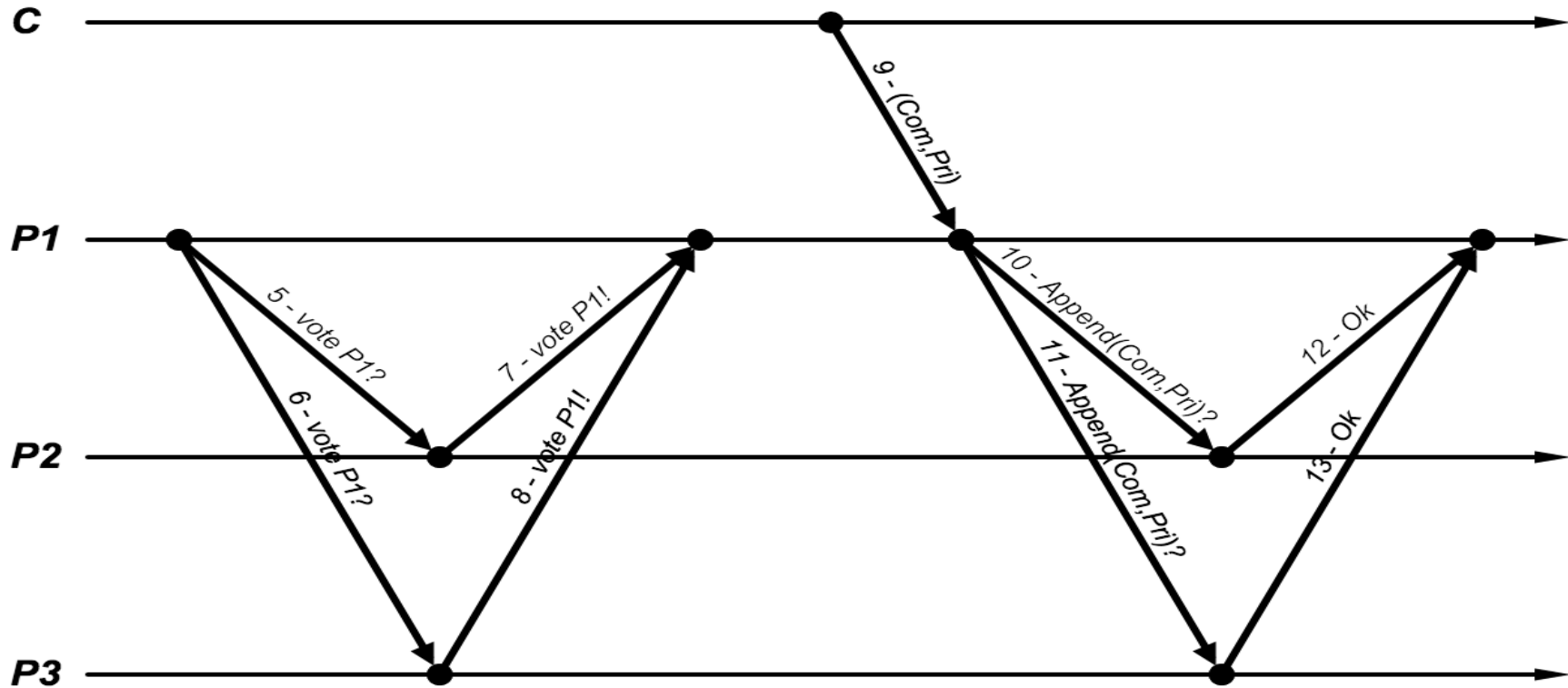
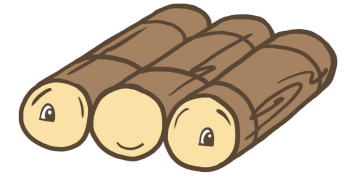


- $F < 2n$
- Assíncrono
- *Crash*
- Comunicação confiável
- Log de requisições:  $\log \{ \alpha, \beta, \gamma \dots \}$

# Raft

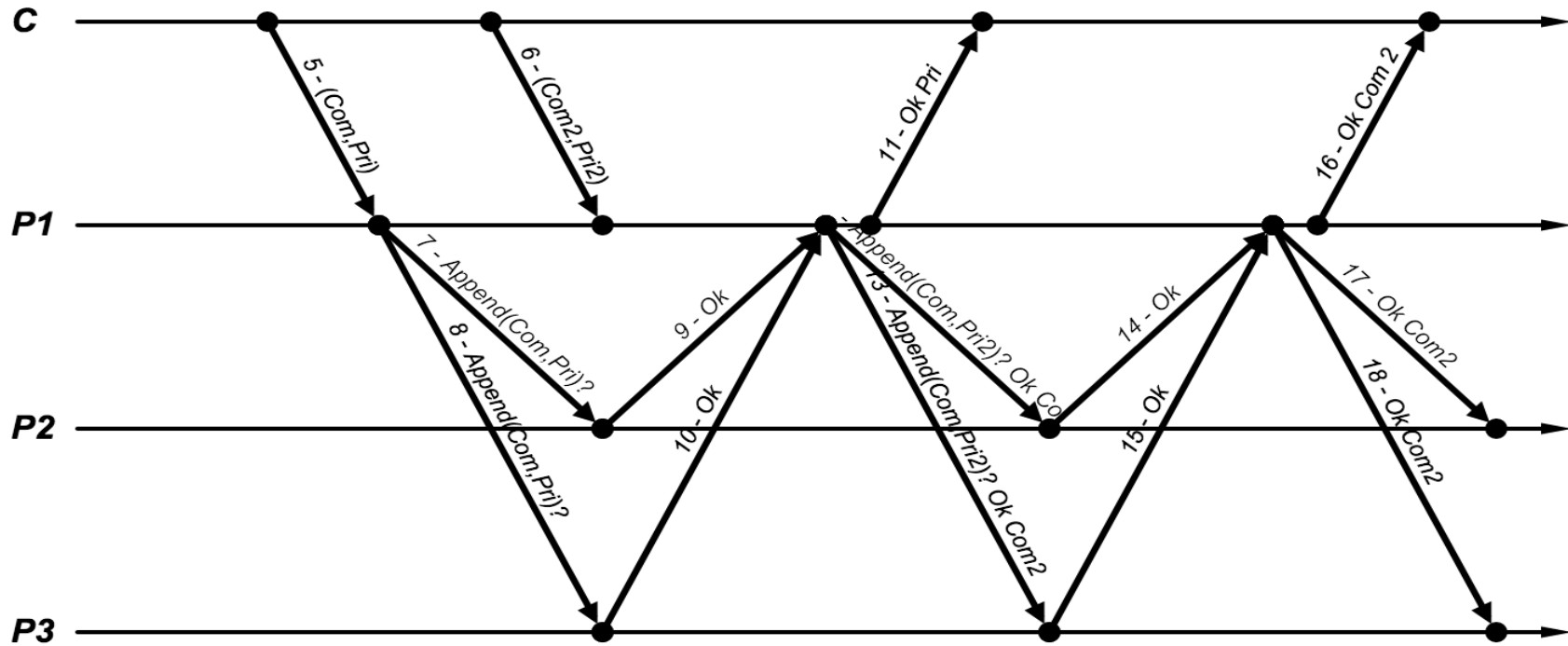


# Eleição



- Candidato com log mais atualizado é eleito
- Líder descobre requisições possivelmente “comitadas”
- 1 líder por mandato

# Raft



- Líder propõe extensões ao log
- Quando a maioria aceita, líder confirma

# PRaft: Propriedades

- **Não Trivialidade:**

- Somente *propostos* são confirmados

- **Acordo:**

- Se dois processos  $p$  e  $q$  confirmaram  $log[i]$ , então  $log_p[i] = log_q[i]$ .

Requisições confirmadas não podem mudar.

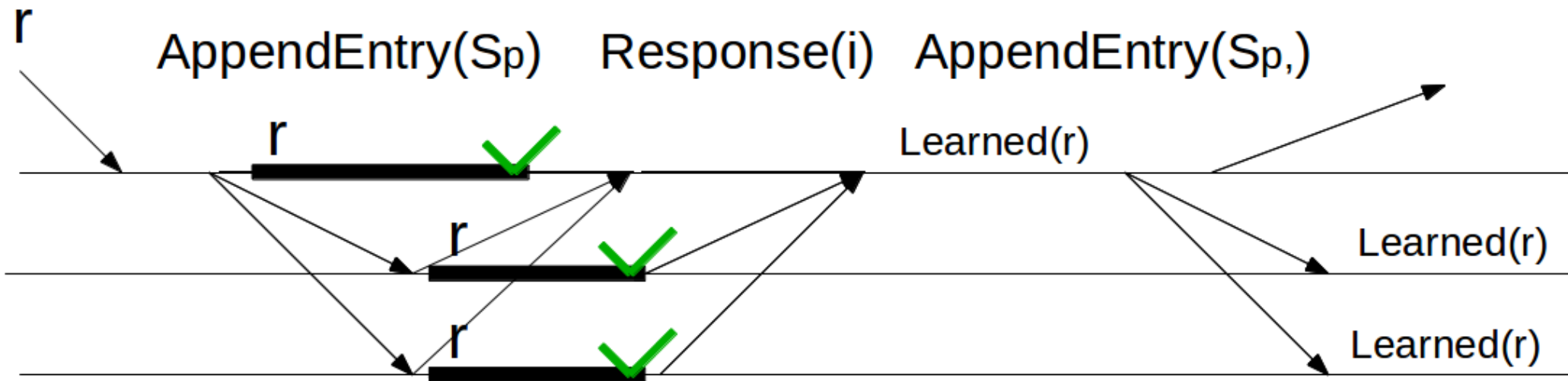
- **Ordem de Prioridade:**

- Requisições não confirmadas são confirmadas segundo a ordem de prioridade.

# PRaft: Modelo de Sistema

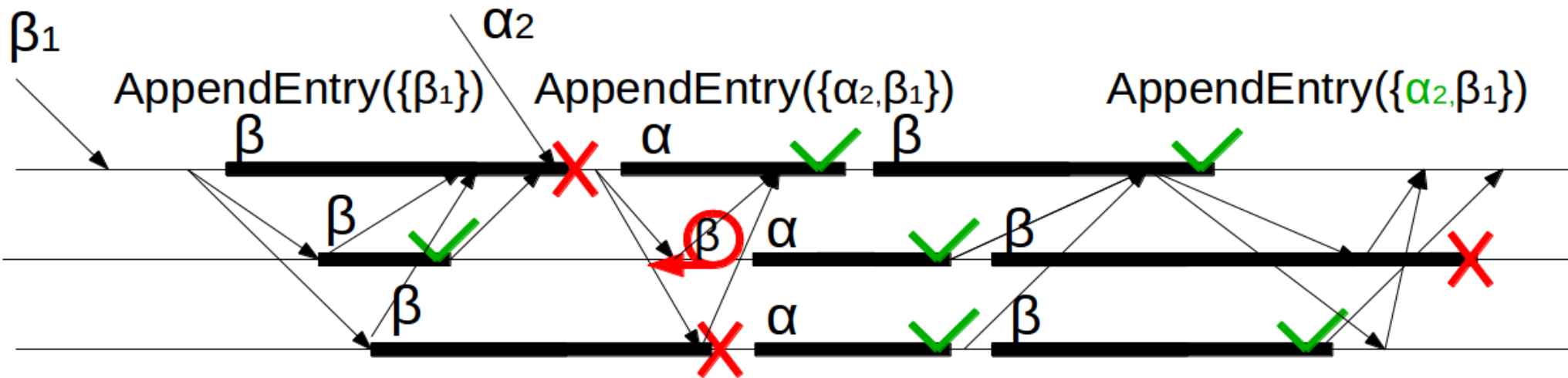
- $f < n/2$
- *Crash*
- *Eventually synchronous*
  
- Prioridades definidas pelos clientes
- $P(r) > P(r') \rightarrow r$  é executado antes de  $r'$

# PRaft



- O líder reordena as requisições prontas e as envia para as réplicas.
- Replicas respondem ao AppendEntry, executam o processo e então enviam ao líder a resposta da execução.

# Preempção



- Quando uma requisição de alta prioridade está pronta, o líder reordena a parte pronta do log com ela.
- Replicas interrompem e fazem *rollback* de β e executam α.

# Comparações

- Complexidade de transmissão de mensagens: linear

Algoritmo	Modelo de Tempo	Modelo de Falta	Tolera	# passos	# mensagens	Complex.
PitTO	síncrono	X	X	3	$(n - 1)^2$	$O(n^2)$
Rodrigues et al.	detector de faltas	Crash	$2f + 1$	2	$(n - 1) + n(n - 1)^2$	$O(n^3)$
Wang et al.	detector de faltas	Crash	$2f + 1$	4	$(2n^3 + 7n^2 + 5n - 2)/2$	$O(n^3)$
<b>PRaft</b>	<i>eventual synchronous</i>	Crash	$2f + 1$	3	$4(n - 1)$	$O(n)$

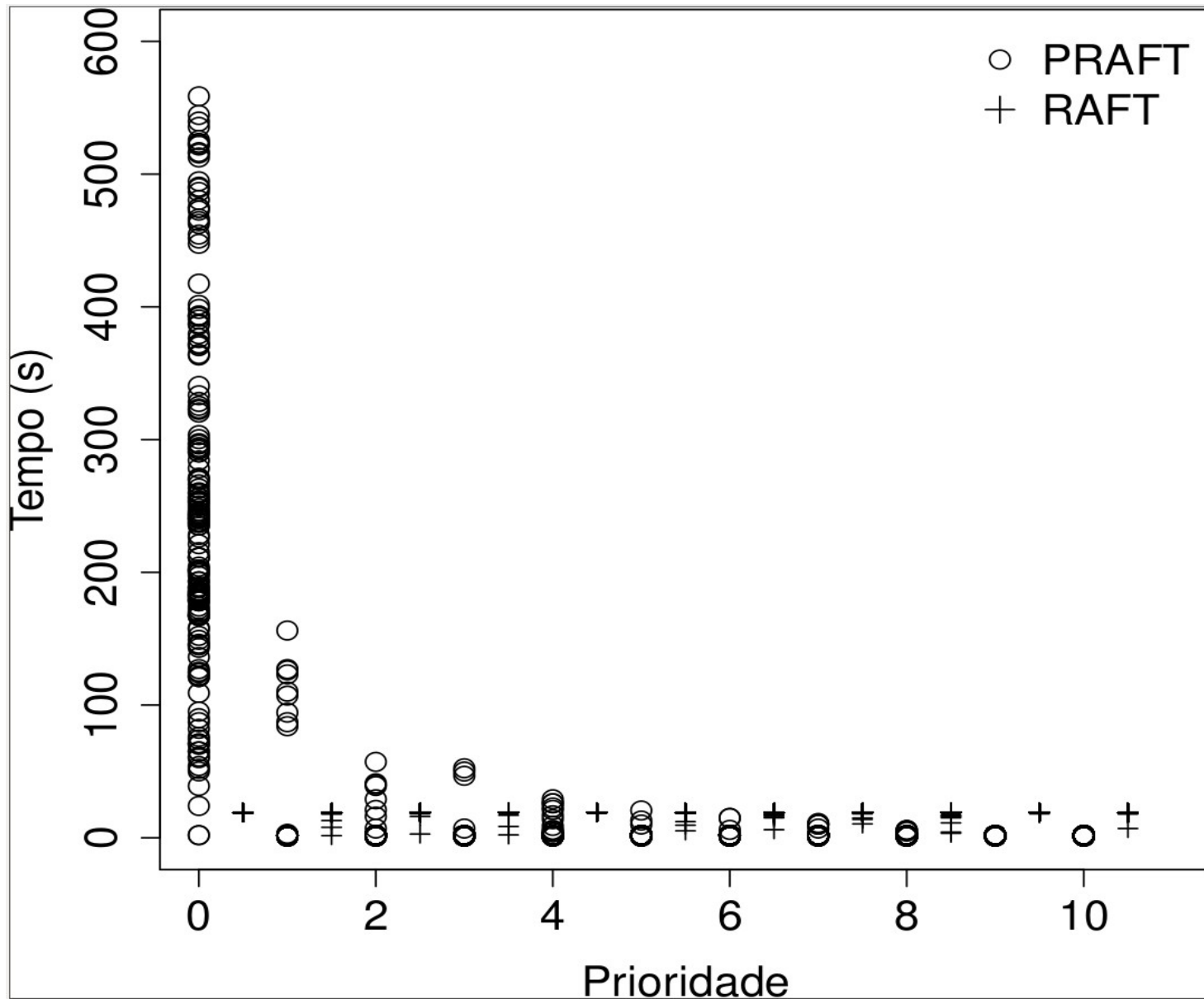
- Custo para adaptação do Raft: n mensagens

Algoritmo	Modelo de Tempo	Modelo de Falta	Tolera	# passos	# mensagens	Complex.
Raft	assíncrono	Crash	$2f + 1$	3	$3(n - 1)$	$O(n)$
<b>PRaft</b>	<i>eventual synchronous</i>	Crash	$2f + 1$	3	$4(n - 1)$	$O(n)$

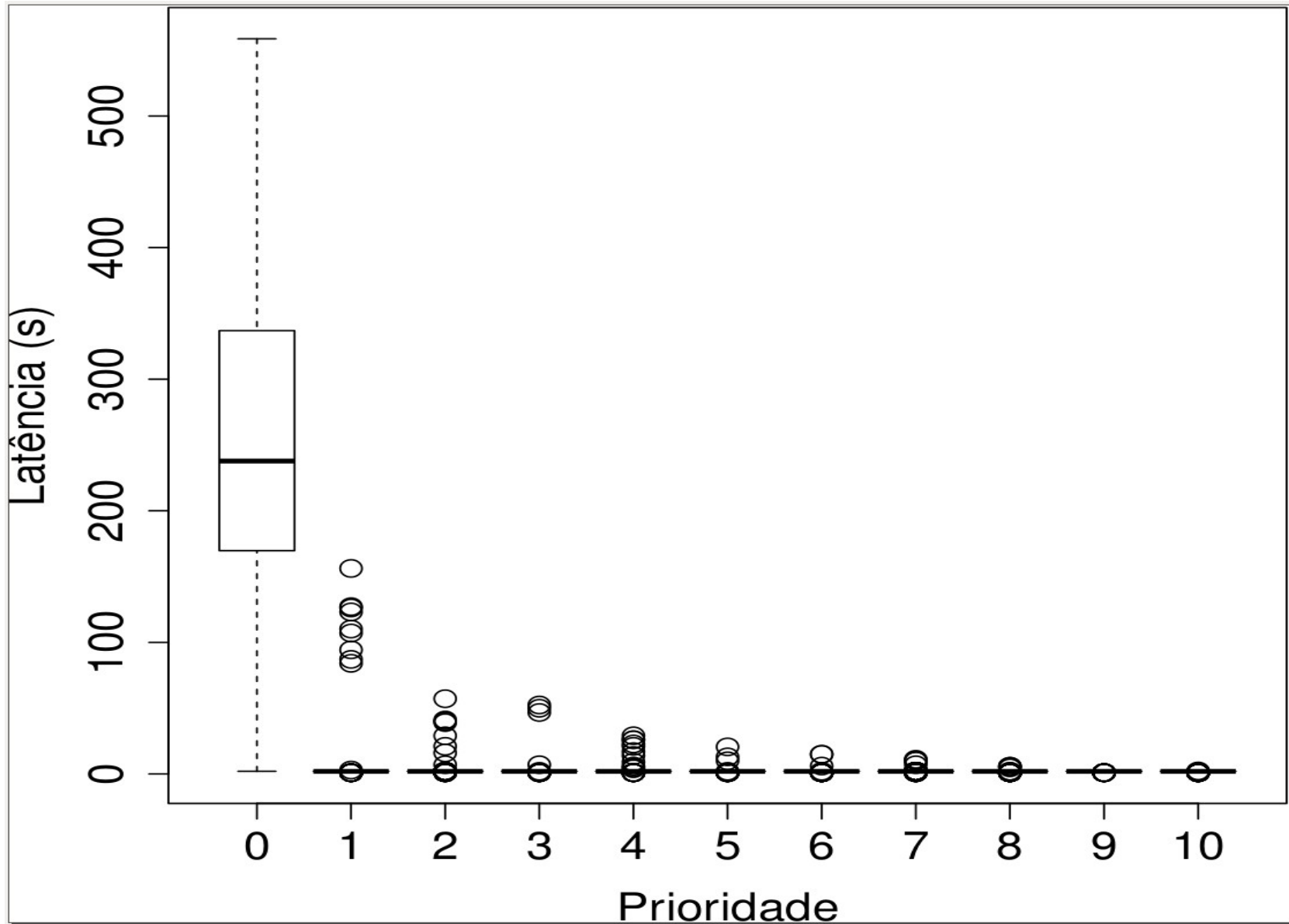
# Testes

- Implementação Python 3
- Comunicação em UDP
- 4 computadores
- 19 Clientes
- 100 mensagens cada
- Prioridades entre 0 e 10
- 1s de execução

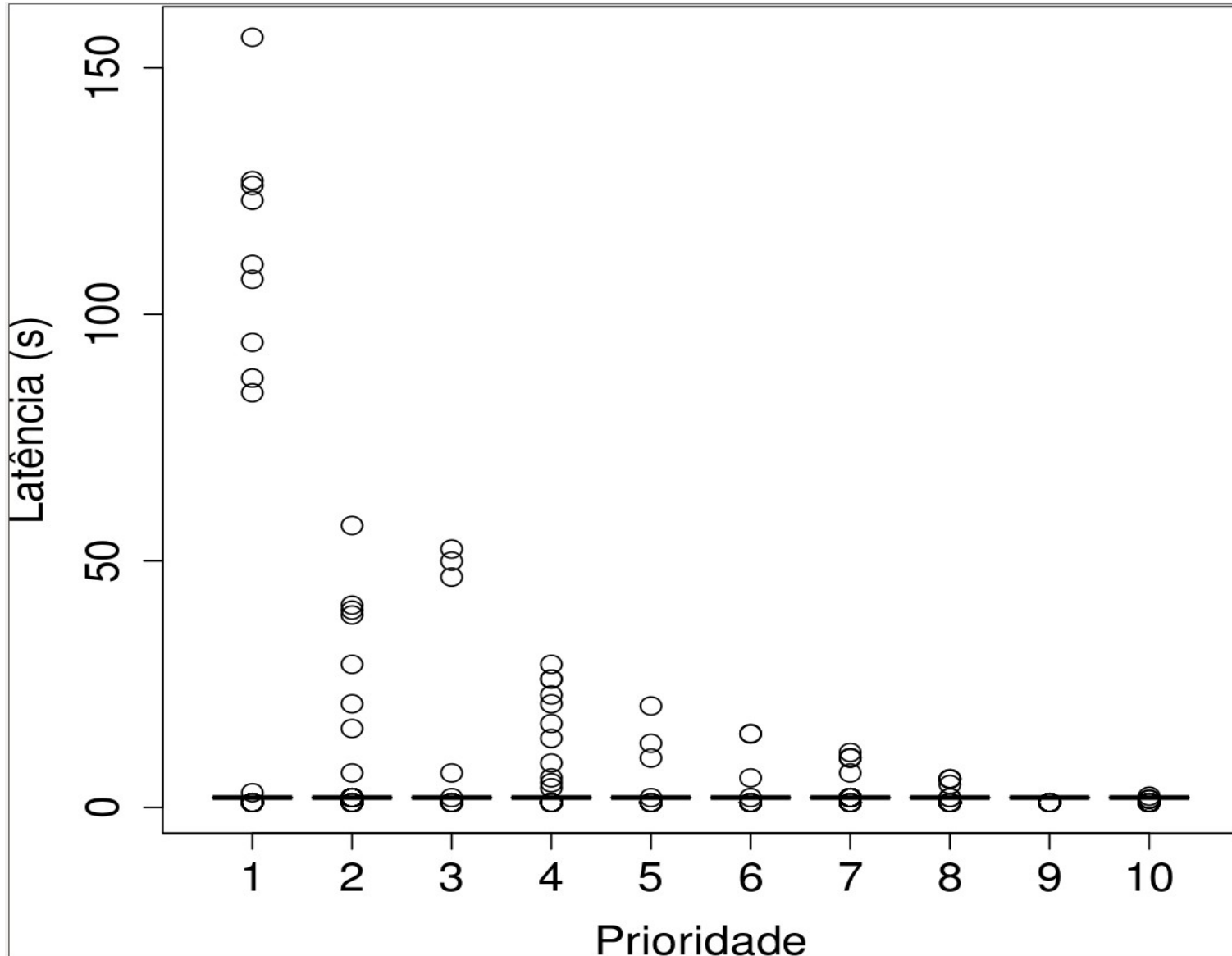
# Testes: Comparação



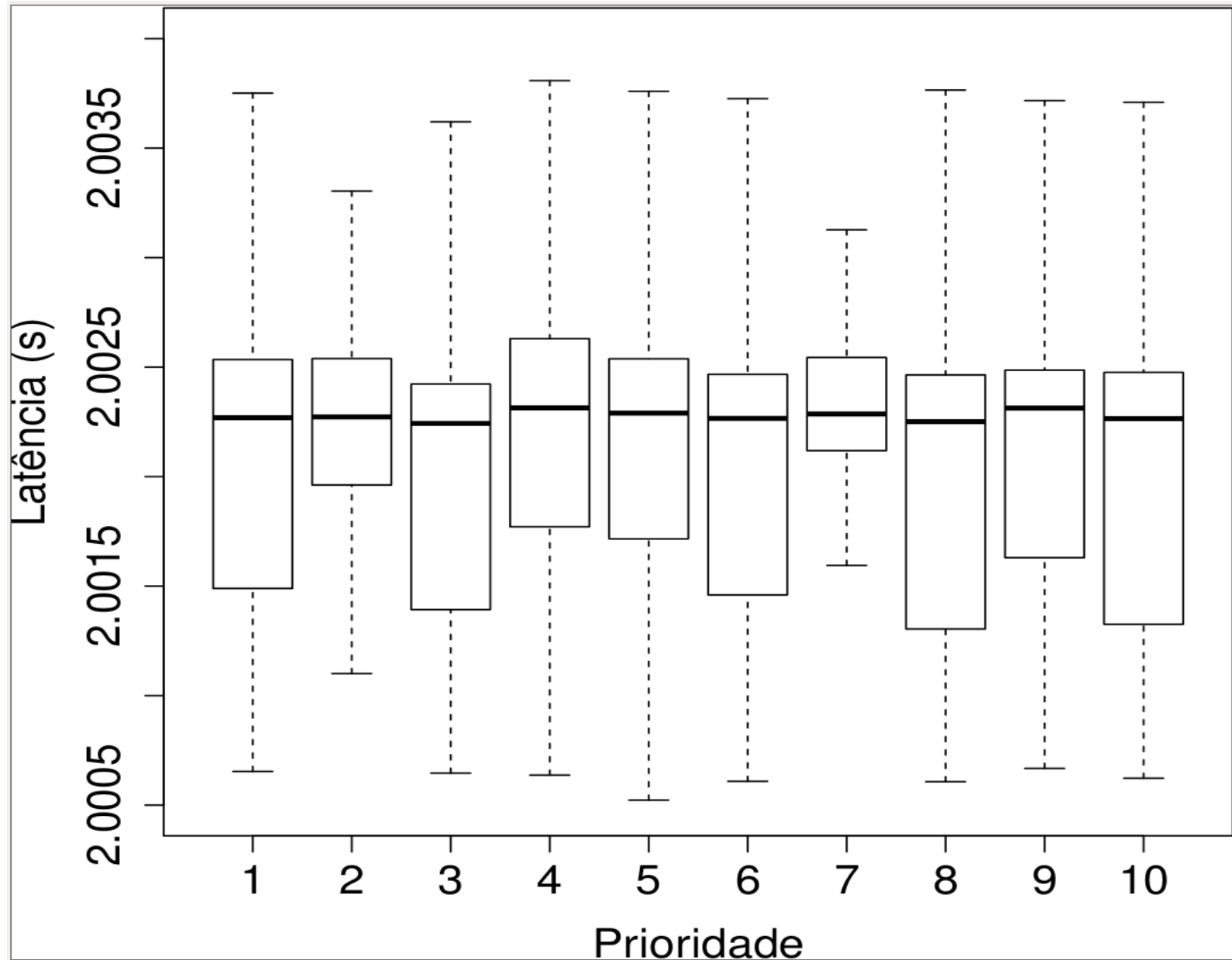
# Testes: PRAFT



# Testes: PRaft sem 0



# Testes: PRaft sem 0



# Contribuições

- Algoritmo P Raft – PB-SMR
- Requisições de alta prioridade têm latência mais baixa no P Raft do que as requisições têm em média no Raft.

# Trabalhos Futuros

- Outras políticas de escalonamento
- Mais testes
- Comparação com os trabalhos relacionados

# Agradecimentos

- CNPq 455303/2014-2
- Fundação para a Ciência e a Tecnologia (FCT UID/CEC/50021/2013).
- CAPES/Brasil (project LEAD CLOUDS)
- Fapemig