Выбор лидера

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Распределенная обработка информации»
Сибирский государственный университет телекоммуникаций и информатики
Осенний семестр, 2019

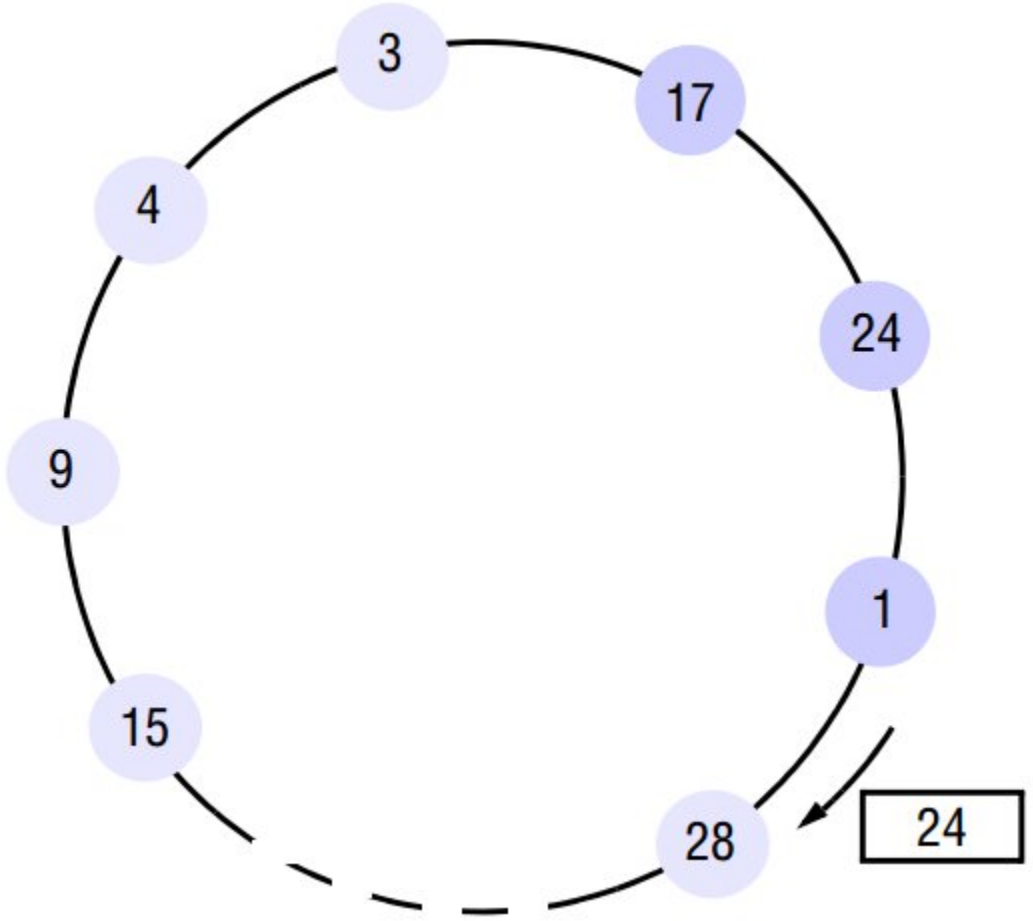
Выборы (Elections, leader election)

- **Необходимо выбрать (elect) один процесс из *n* для реализации заданной роли в распределенной системе**
  - ❑ Все процессы должны прийти к одному решению
  - ❑ Выборы могут быть инициированы любым процессом в любое время
  - ❑ *n* процессов могут одновременно инициировать *n* выборов
- **Например**, выбрать процесс для синхронизации часов, процесс для логирования, управления доступом к критической секции, ...
- Считаем, что выбранный процесс должен иметь наибольший «идентификатор»

Требования к алгоритму выборов

- Каждый процесс p_i имеет переменную $elected_i$
  - В начале участия в выборах устанавливается в *elected* = NULL
- Безопасность
  - Переменная *elected* у участвующего в выборах процесса может принимать значения NULL и P, где P – выбранный процесс с наибольшим идентификатором
- Живучесть
  - Каждый процесс в конечном итоге принимает участие в выборах и устанавливает значение elected != NULL или отказывает (в случае сбоя)
- Производительность
  - Количество сообщений
  - Время выборов

Кольцевой алгоритм выборов (Chang and Roberts, 1979)

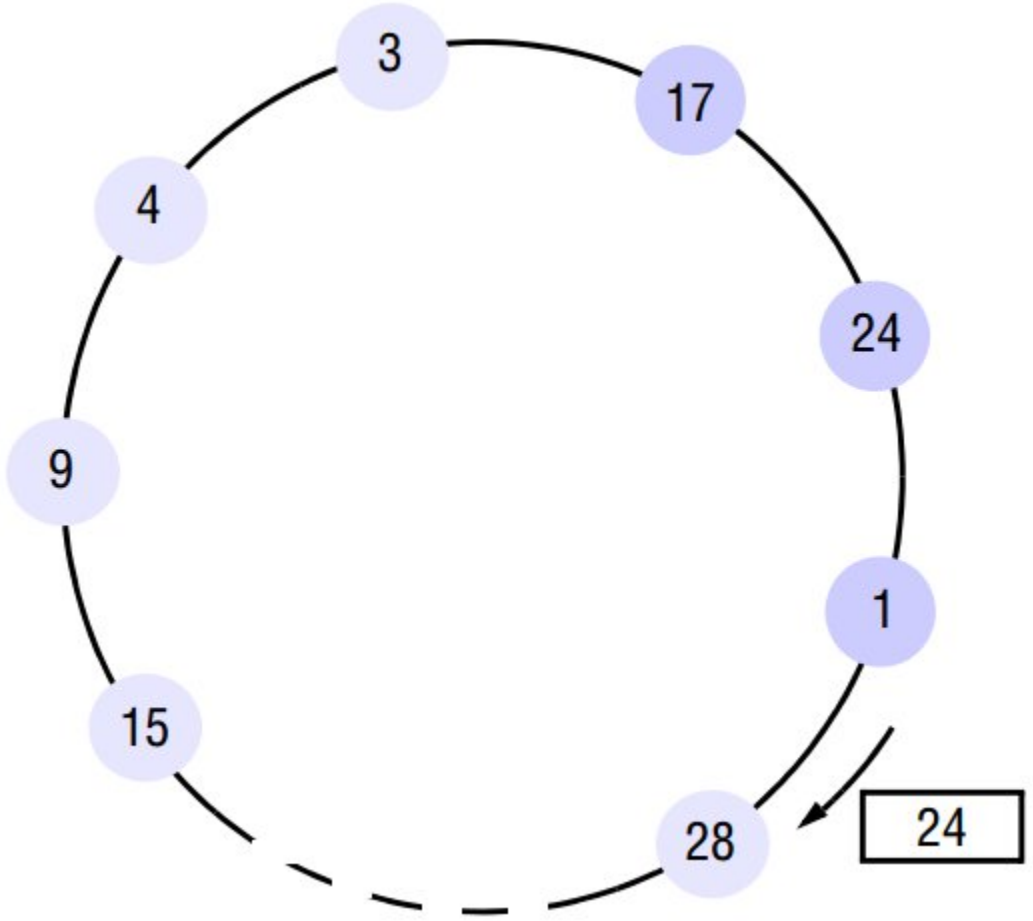


Выборы начаты процессом 17

- Процесс, желающий участвовать в выборах переводит себя в состояние “участник” и отправляет соседу (по часовой стрелке) свой id
- Когда процесс *i* получает сообщение, он проверяет:
  - ❑ Если он “участник”, алгоритм завершается
  - ❑ Выбирается максимум из принятого id и id текущего процесса: id = max(id_{recv}, id_i)
  - ❑ Вычисленное значение id передается дальше по кольцу, а текущий процесс становится “участником”

AllReduce(id, MAX)

Кольцевой алгоритм выборов (Chang and Roberts, 1979)



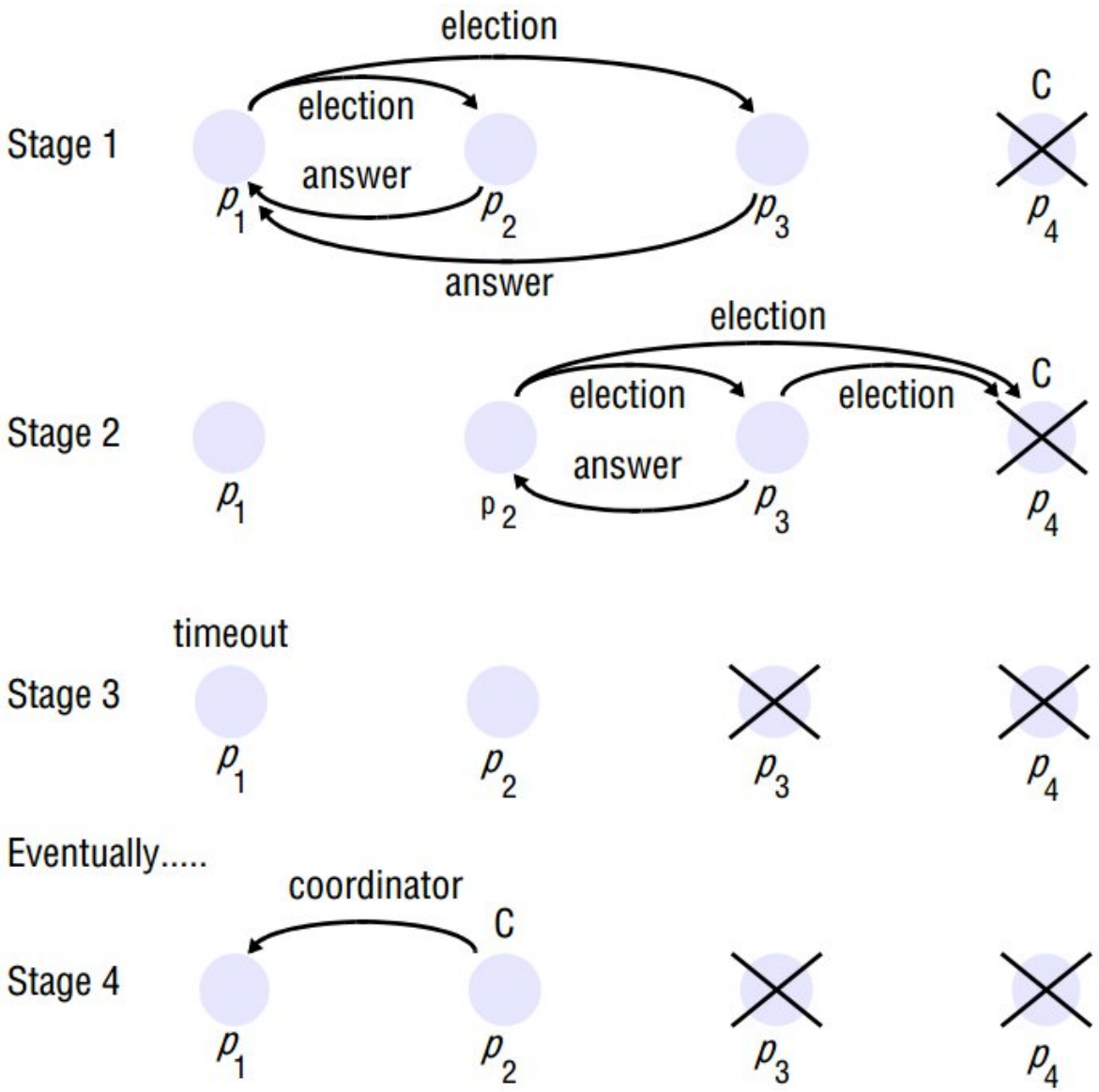
Выборы начаты процессом 17

Выбрать среди *n* процессов наименее загруженный

- Идентификатор процесса *i* = <1 / *load*, *i*> – номер процесса используется в сравнении, если загрузка процессов совпадает

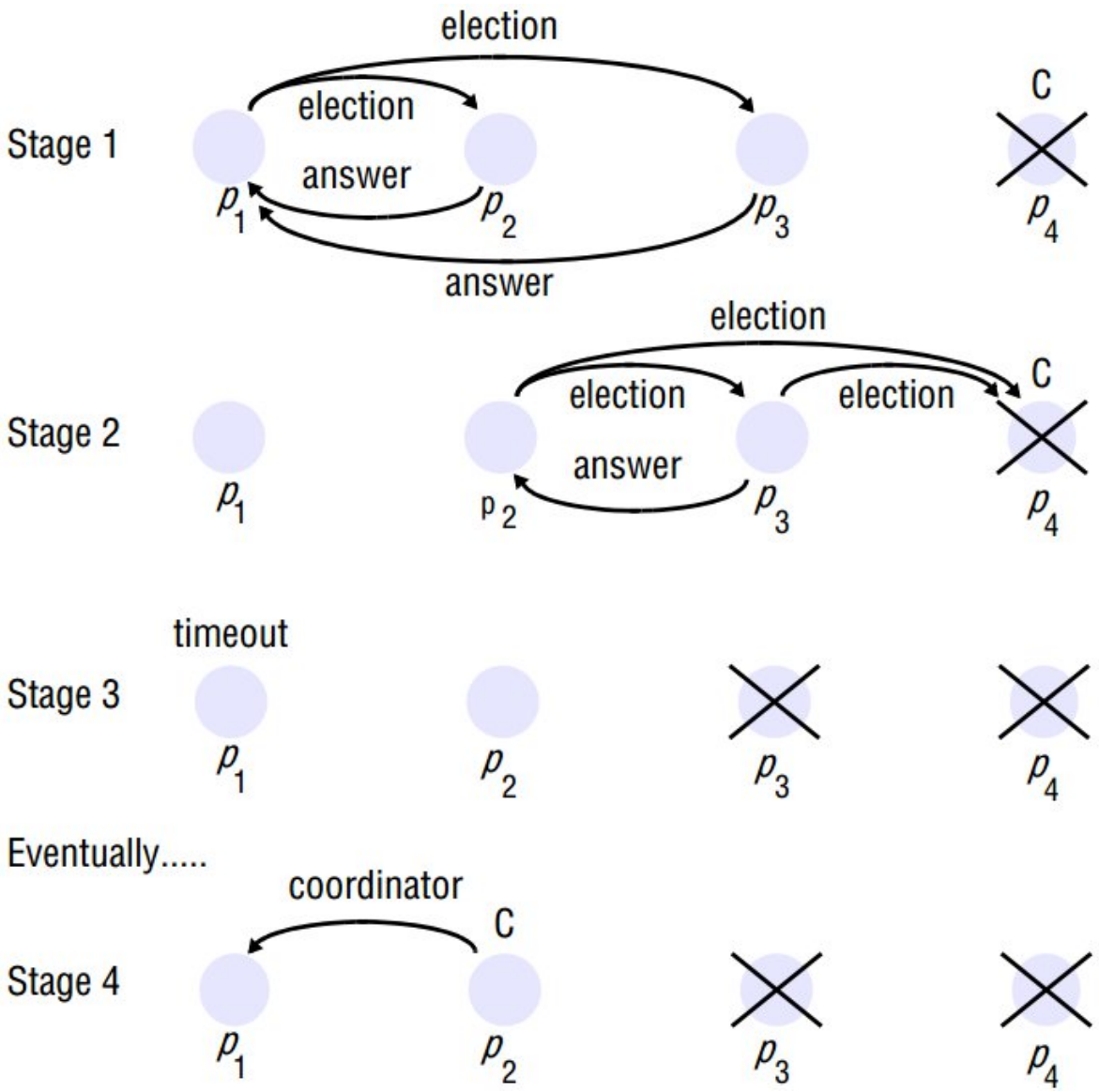
```
#include <stdlib.h>

double loadavg[3];
getloadavg(loadavg, 3);
double load_1min = loadavg[0];
```

Bully algorithm (Garcia-Molina, 1982)



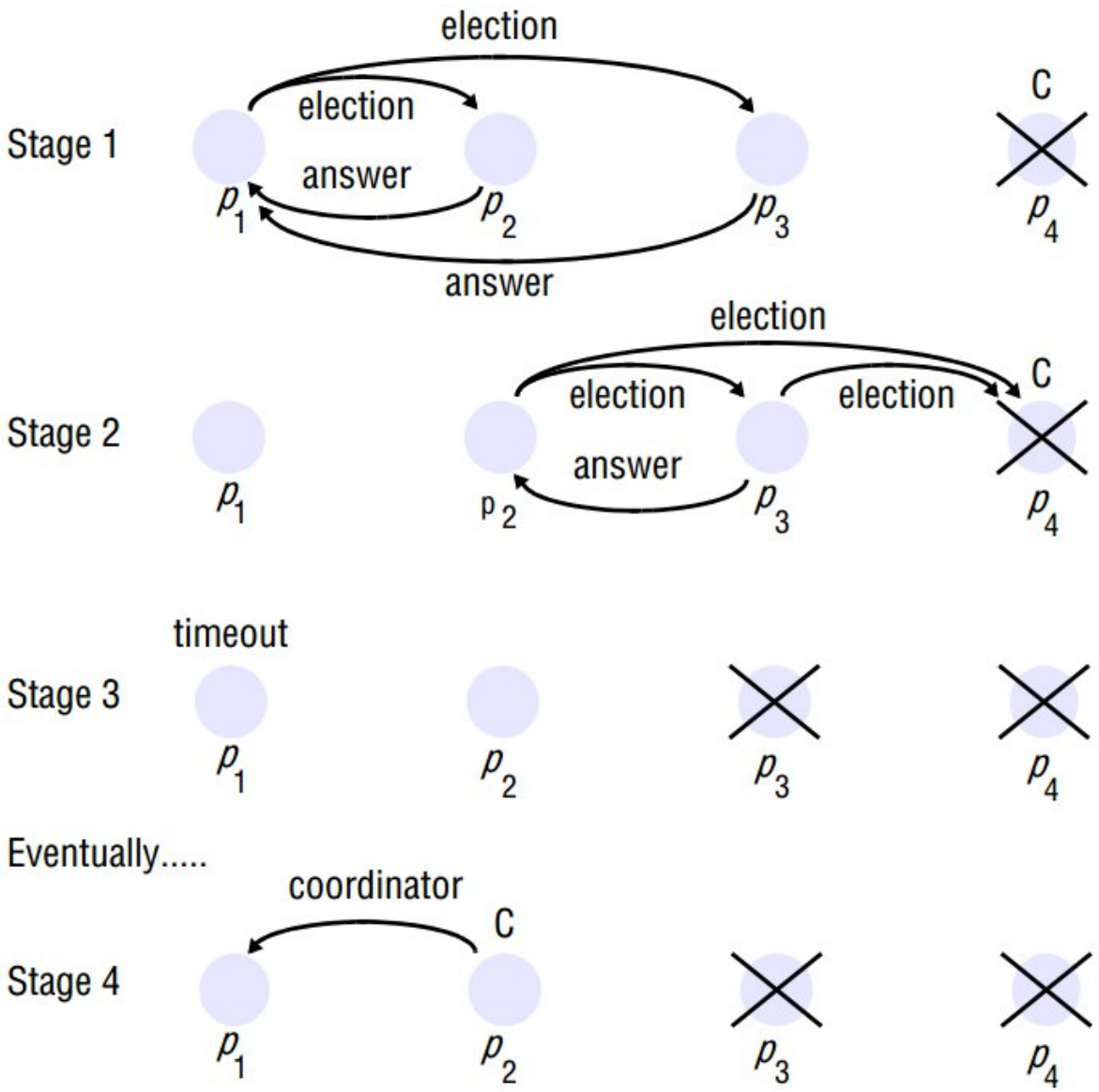
- Синхронная распределенная система
- Допускается отказ процессов, но не каналов
- Процесс отказал, если ответ не пришел в течении времени $T = T_{RTT} + T_{MessageProcess}$
- Все процессы знают, какой процесс имеет максимальный id
- В худшем случае $O(n^2)$ сообщений

Bully algorithm (Garcia-Molina, 1982)



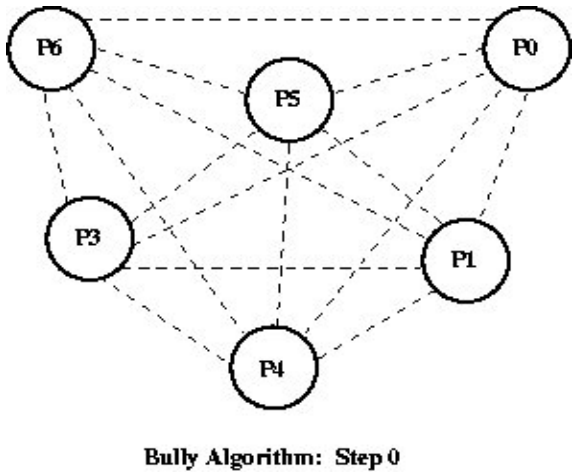
- **Типы сообщений**
  - *Запуск выборов* (election message) – инициация процесса выборов
  - *Ответ* (answer, alive) – ответ на сообщение <election>
  - *Координатор* (coordinator) – отправляется лидером

Bully algorithm (Garcia-Molina, 1982)



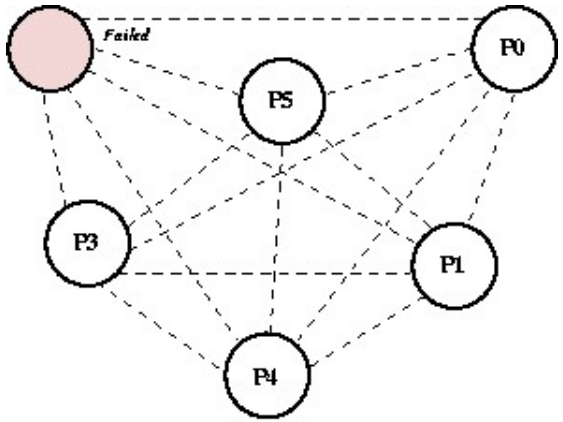
- **Процесс *p* восстановлен, либо установлено, что текущий координатор отказал:**
- Если процесс *p* имеет максимальный id он отправляет сообщение <*coordinator*> всем и становится лидером;
  в противном случае, процесс отправляет всем процессам с id большего его сообщение <*election*>.
- Если *p* не получил <*answer*> после <*election*>, он рассылает всем сообщение <*coordinator*> и становится лидером.
- Если *p* получил <*answer*> от процесса с большим *id*, он не отправляет сообщений и ждет сообщения <*coordinator*>.
  (если сообщение не приходит за T_{fault}, процесс запускает алгоритм заново)
- Если p получил сообщение <*election*> от процесса с меньшим id, он отправляет ему <*answer*> и запускает выборы – рассылает <*election*> процессам с большими id
- Если p получил <*coordinator*>, он считает отправителя лидером

Bully algorithm (Garcia-Molina, 1982)



Bully Algorithm: Step 0

- **Система из 6 процессов**
- Полный граф связей
- Процесс 6 – текущий лидер (наибольший ID)

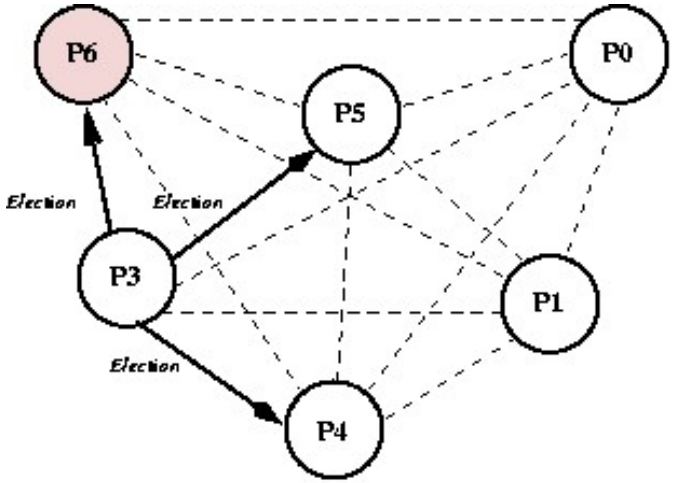


Bully Algorithm: Step 1

- **Процесс 6 отказал**
- Отказ обнаруживает подсистема контроля и диагностики (fault detector)

https://www.cs.colostate.edu/~cs551/CourseNotes/Synchronization/BullyExample.html

Bully algorithm (Garcia-Molina, 1982)

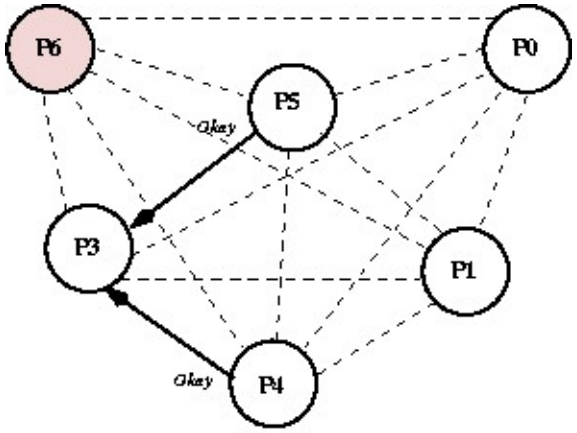


Bully Algorithm: Step 2

- **Процесс 3 обнаружил отказ процесса 6**
- Процесс 3 запускает выборы нового лидера
  - Отправляет сообщение <election> процесса с ID больше 3: 4, 5, 6

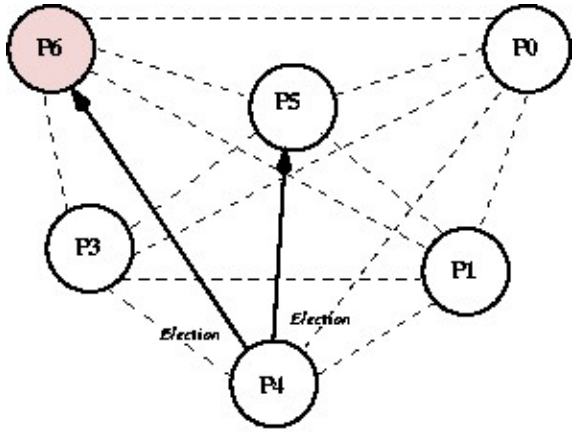
https://www.cs.colostate.edu/~cs551/CourseNotes/Synchronization/BullyExample.html

Bully algorithm (Garcia-Molina, 1982)



Bully Algorithm: Step 3

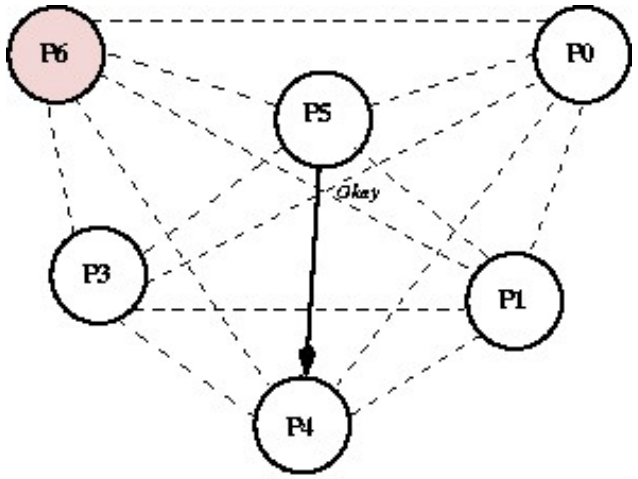
- Процессы 4 и 5 отправляют процессу 3 ответ <answer>
- Процесс 3 получает ответы и ждет сообщения <coordinator>



Bully Algorithm: Step 4

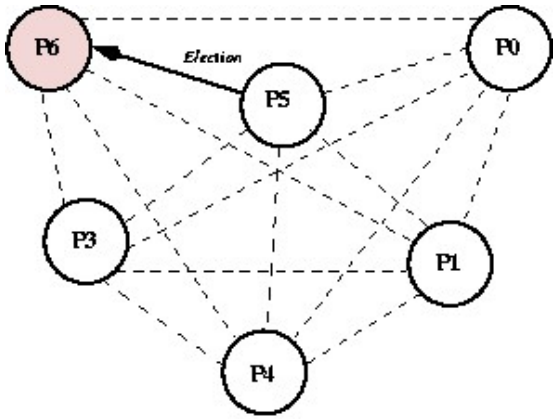
- Процесс 4 отправляет <election> процессам 5 и 6

Bully algorithm (Garcia-Molina, 1982)



Bully Algorithm: Step 5

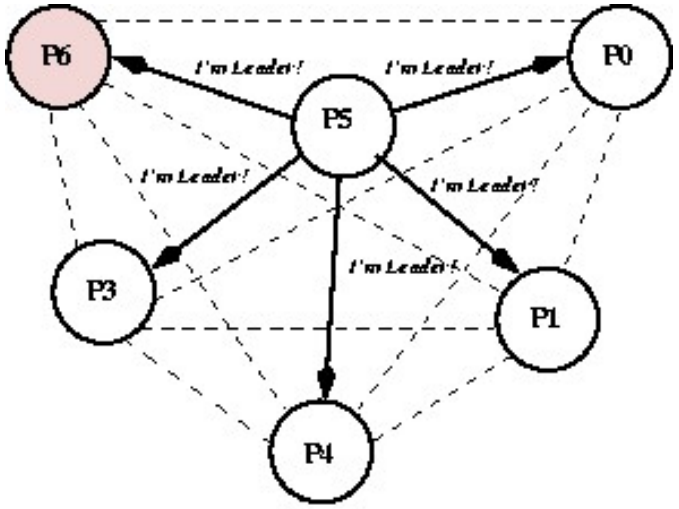
- Процесс 5 отвечает процессу 4 на сообщение <election> сообщением <answer>



Bully Algorithm: Step 6

- **Процесс 5 запускает выборы** – передает сообщение <election> процессам с большим id: 6

Bully algorithm (Garcia-Molina, 1982)

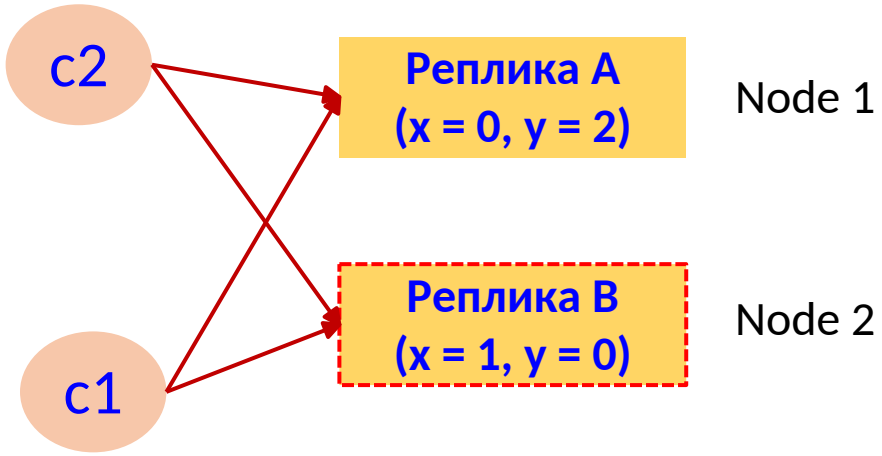


Bully Algorithm: Step 7

- Процесс 6 не ответил на сообщение <election>
- Процесс 5 объявляет себя лидером и рассылает всем процессам сообщение <coordinator>

Репликация (replication)

- Клиент обращается к локальному менеджеру реплик (RM)
- Если локальный RM не доступен, то обращается ко второму

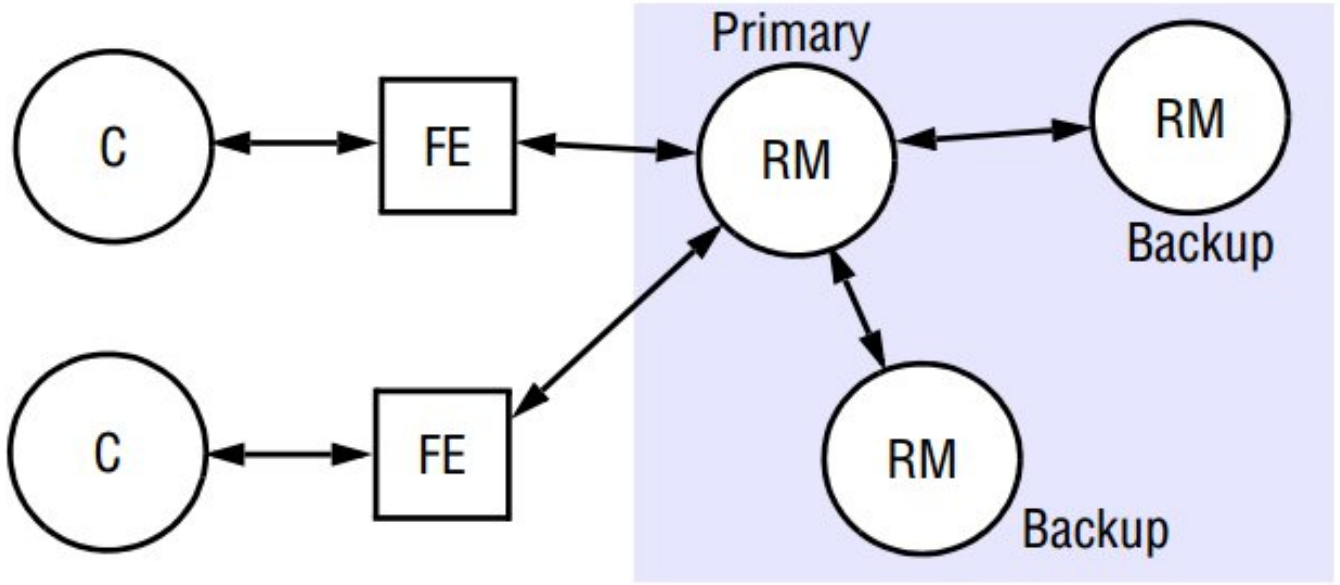


Client 1:	Client 2:
$setBalance_B(x, 1)$	
$setBalance_A(y, 2)$ – *B fault*	
	$getBalance_A(y) \rightarrow 2$
	$getBalance_A(x) \rightarrow 0$

B отказал, A не обновлена

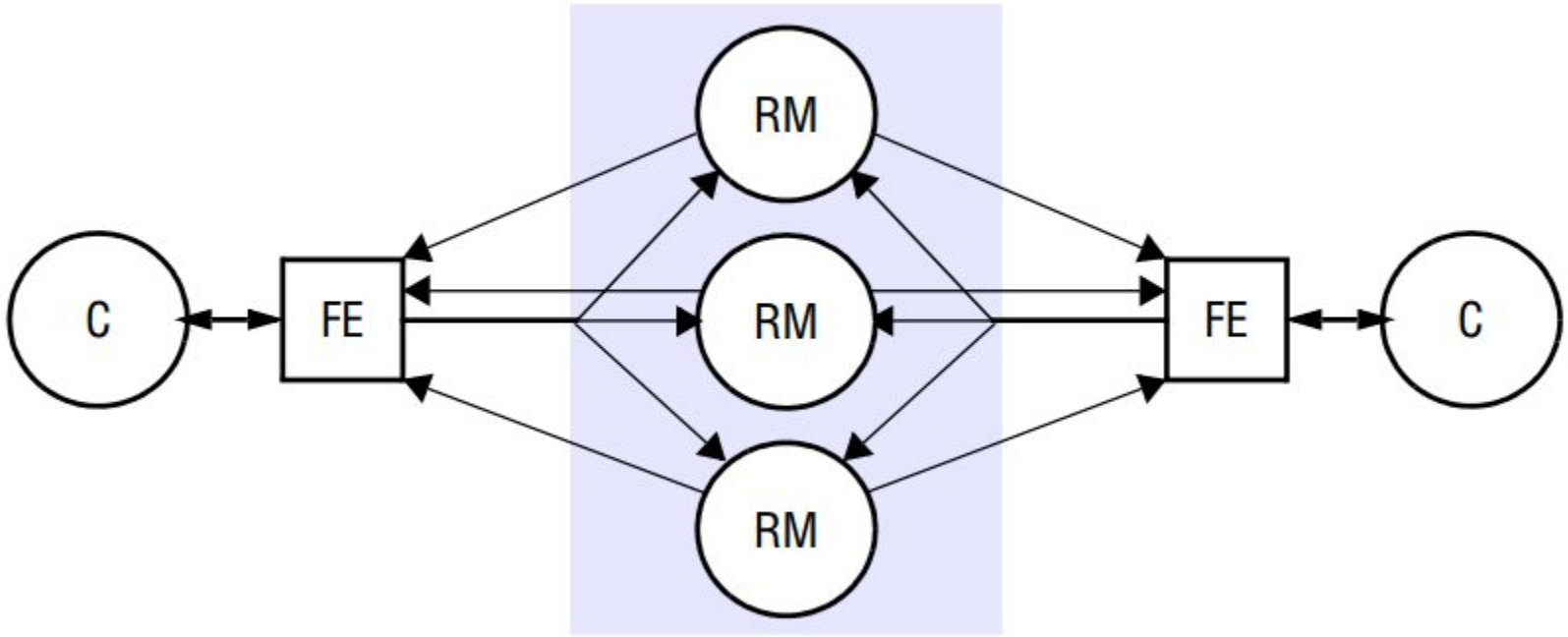
Client 1:	Client 2:
$setBalance_B(x, 1)$	
	$getBalance_A(y) \rightarrow 0$
	$getBalance_A(x) \rightarrow 0$
$setBalance_A(y, 2)$	

Пассивная модель репликации (primary-backup)



- Одна реплика активна (primary)
- Остальные реплики пассивные (backups, slaves)
- Активная реплика отправляет в подчиненные обновления данных
- В случае отказа активной реплики, одна из подчиненных становится главной

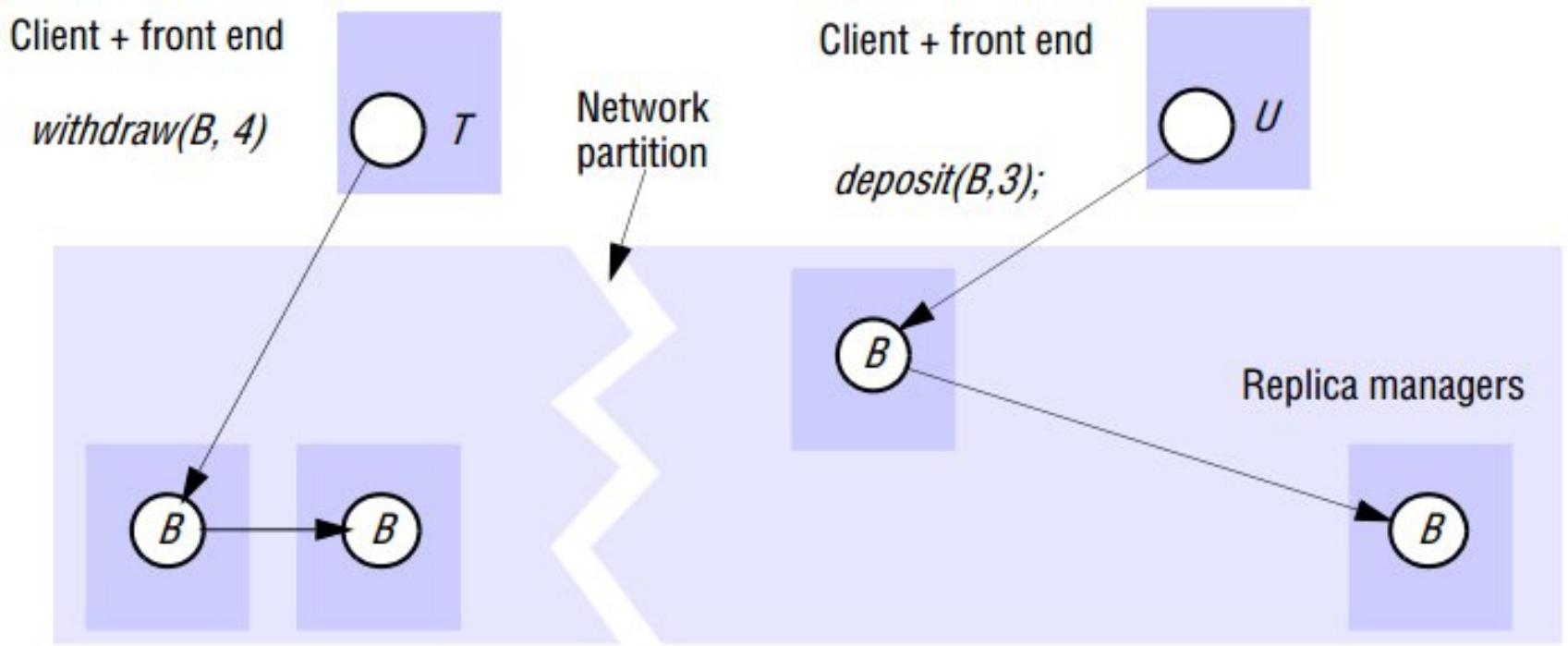
Активная модель репликации (Active replication)



- Пограничные узлы (FE) передают запросы всем репликам (RM)
- Все менеджеры реплик обрабатывают запросы одинаково

Разделение сети (Network partition)

- Возможна ситуация, когда между репликами будет потеряна связь (сетевое соединение)
- Несогласованность данных реплик разных частей



Теорема CAP

- **Теорема CAP** (**теорема Брюера, Brewer, 2000**) – эвристическое утверждение
- В распределенной системе невозможно одновременно обеспечить свойства
  - ❑ согласованности данных (Consistency)
  - ❑ доступности (Availability)
  - ❑ устойчивости к разделению (Partition tolerance) – отклик всегда корректный
- Из трех свойств одновременно можно обеспечить не более двух

http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

Распределенные системы хранения данных

- Выбор реализуемых свойств на уровне архитектуры системы
  - ❑ CA: реляционные СУБД, LDAP
  - ❑ CP: Google BigTable, HBase
  - ❑ AP: веб-кэши, DNS, Amazon Dynamo
- Возможность выбора свойств пользователем на уровне отдельных операций
  - ❑ Apache Cassandra

Домашнее чтение

- Задача византийских генералов (Byzantine generals problem)
- Теорема CAP