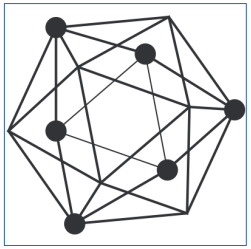
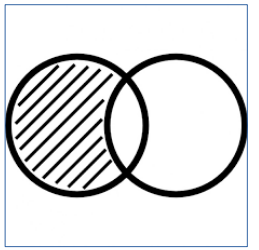
# OSDay 2018

## Построение и верификация отказоустойчивого алгоритма распределенной блокировки



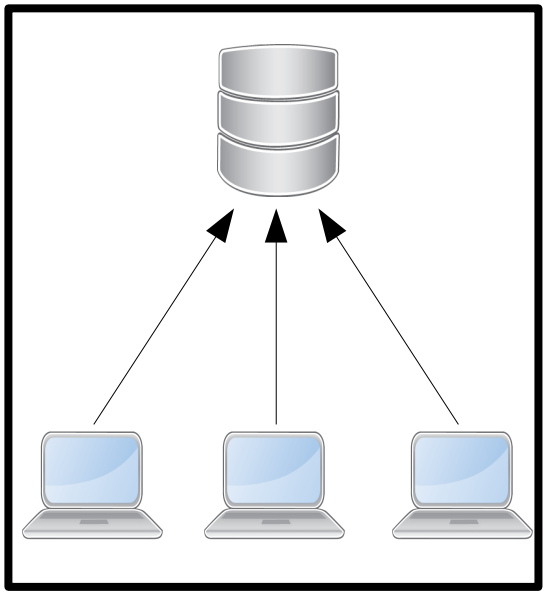

Евгений Шишкин
ИнфоТеКС
2018



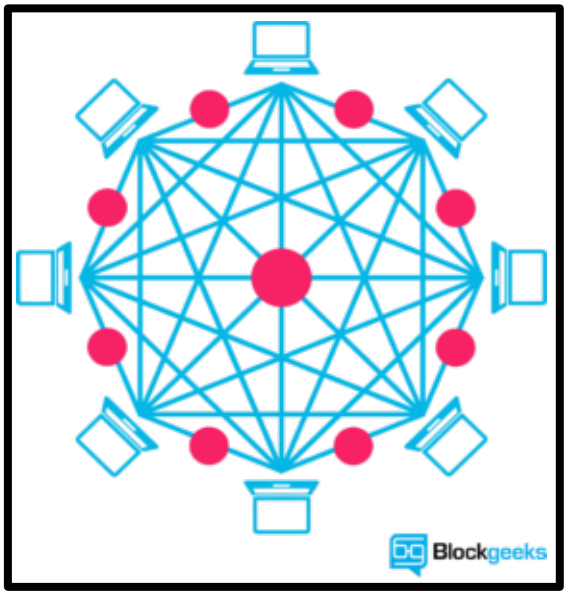

OS:DAY

infotecs®

# Распределённая система

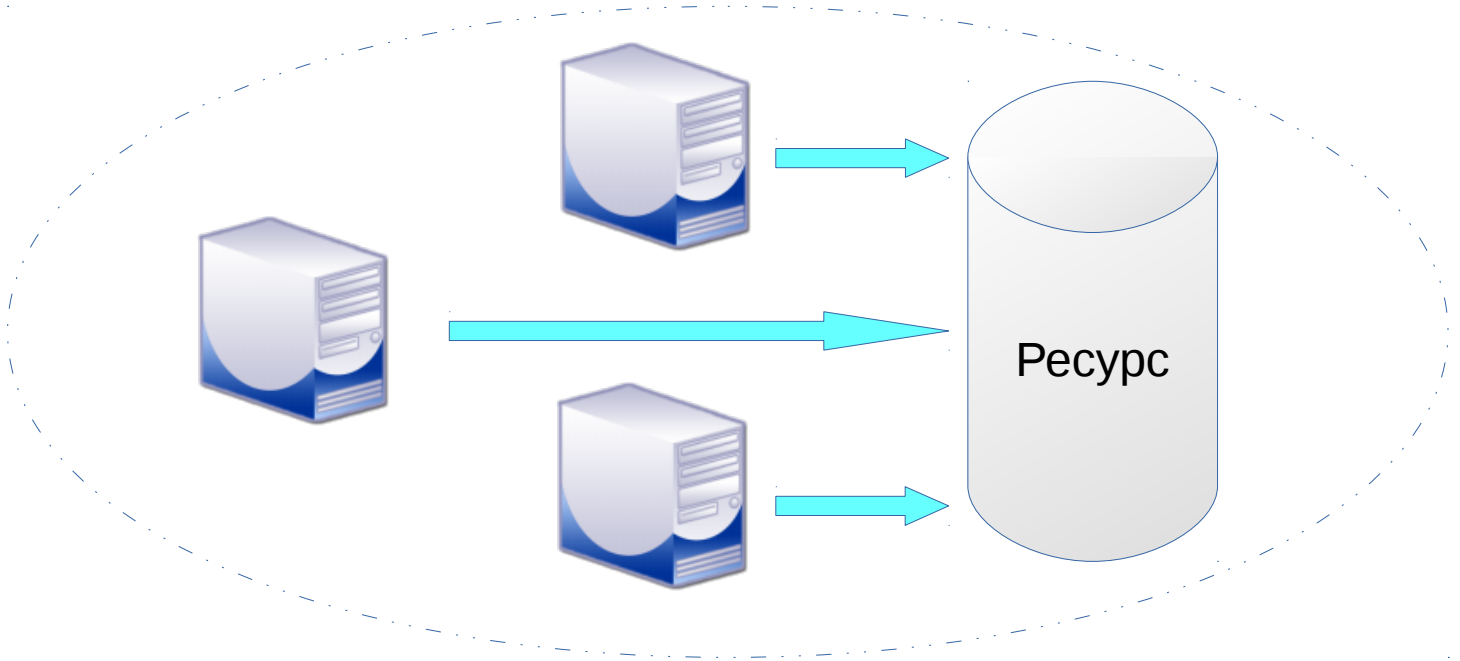


Клиент-Серверная архитектура




Peer-To-Peer архитектура

# Алгоритм распределённой блокировки



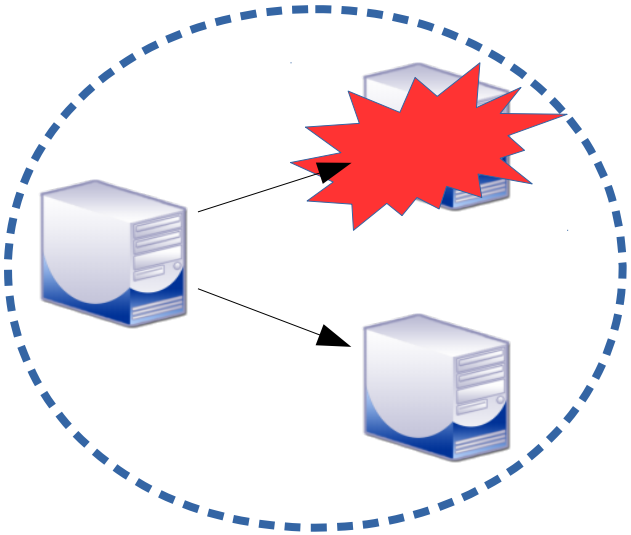

- Несколько процессов в распределенной системе пытаются получить эксклюзивный доступ к разделяемому ресурсу
- Как убедиться в ***эксклюзивности*** доступа? Допускается использовать только обмен сообщениями между процессами
- Алгоритм распределенной блокировки решает именно эту задачу

# Источники недетерминизма

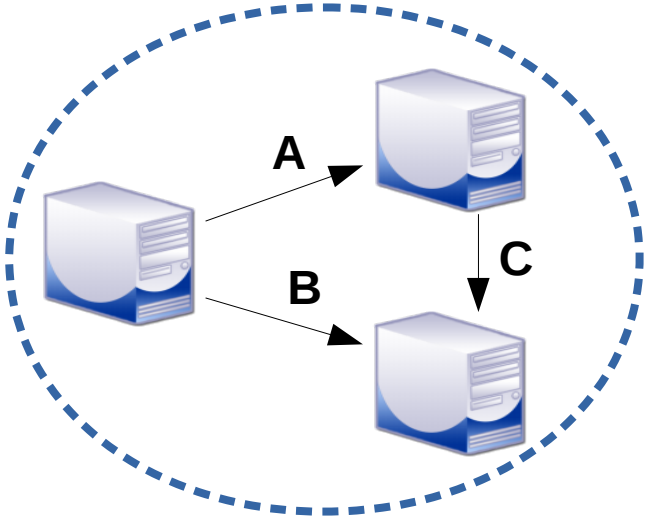
## Планировщик

Process A, Process B, Process C
Process A, Process C, Process B
Process B, Process A, Process C
Process B, Process C, Process A
Process C, Process A, Process B
Process C, Process B, Process A

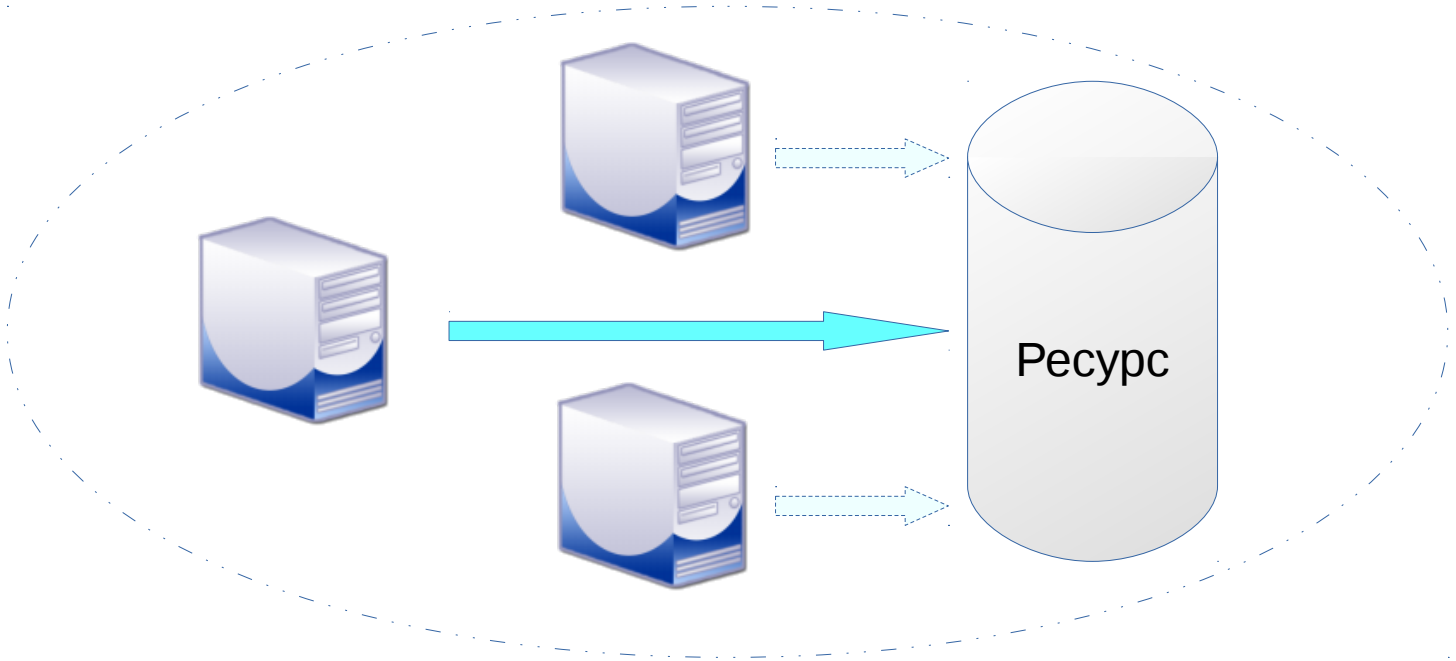
## Сбои



## Асинхронная доставка сообщений



# Свойство взаимного исключения



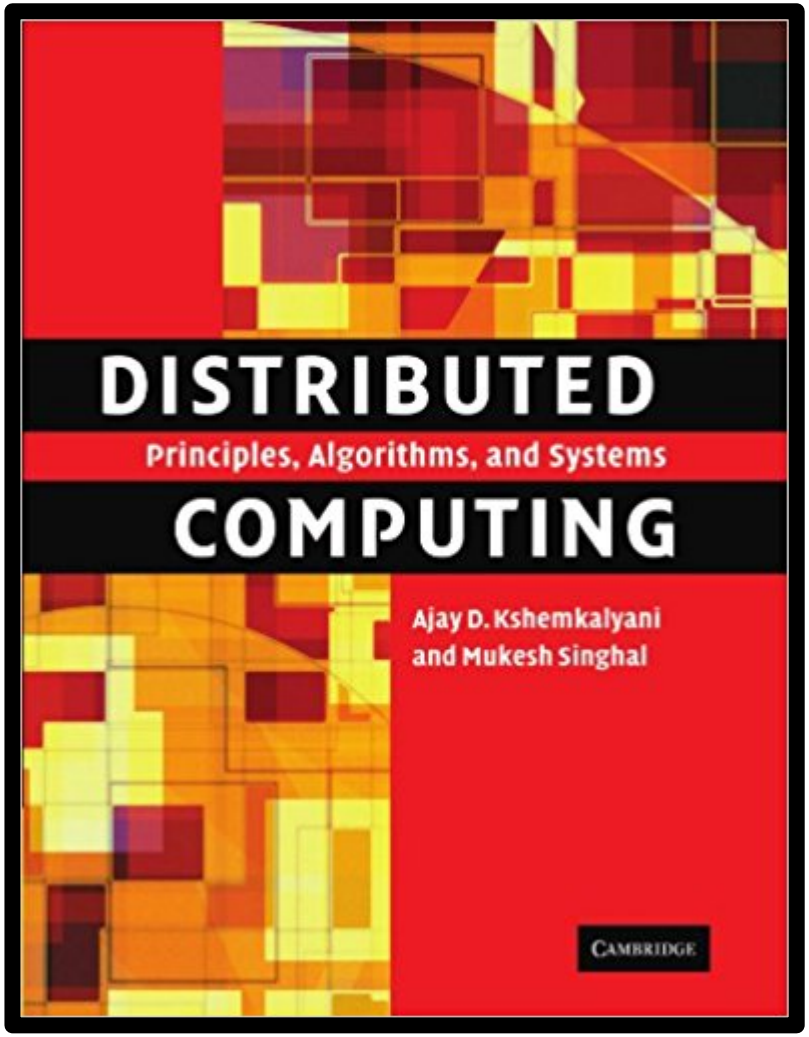

В любой момент времени доступ к разделяемому ресурсу имеет ***не более одного*** процесса.

# Почему не взять готовое?

- Повышенные требования к надежности разрабатываемой системы
- Отсутствие компонент с заданной семантикой в инфраструктуре Erlang
- Экспертиза в области проверки корректности распределенных алгоритмов

# Почему не взять алгоритм “из книжки”?

- Не отказоустойчивые
- Не способны менять группу участников в процессе работы алгоритма
- Отсутствие формальных доказательств



# Алгоритм Рикарта-Агравала


Technical Report TR-869 February 1980

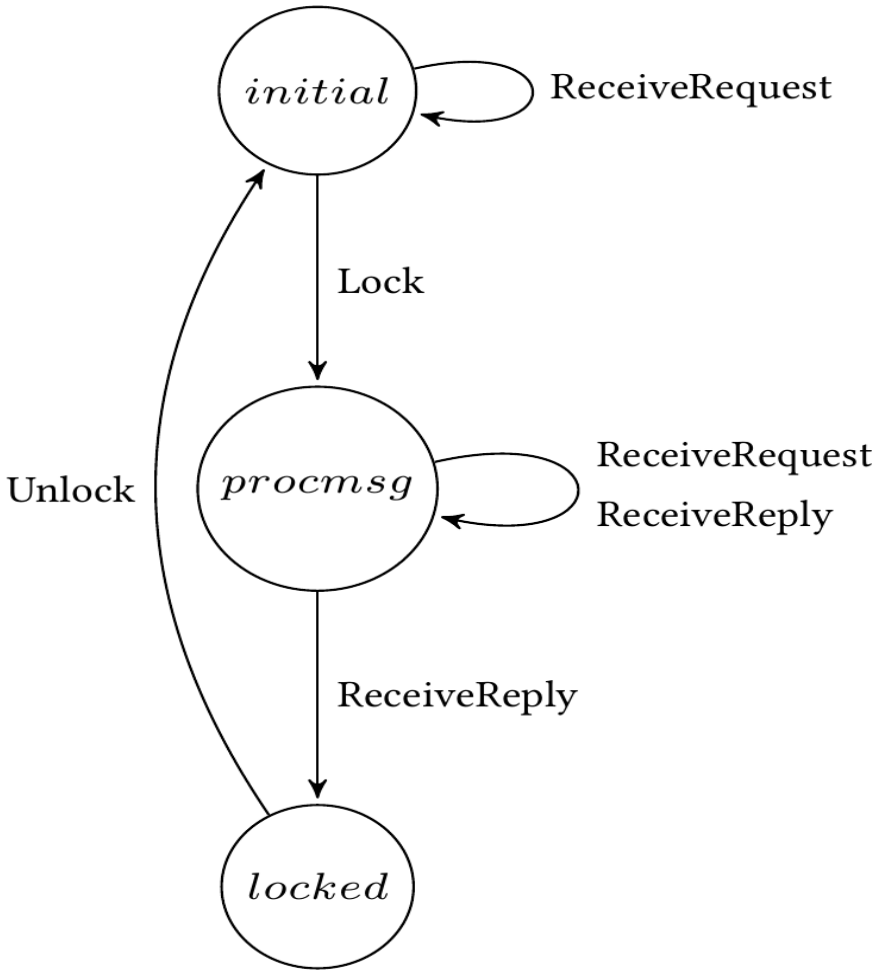
AN ALGORITHM FOR MUTUAL EXCLUSION

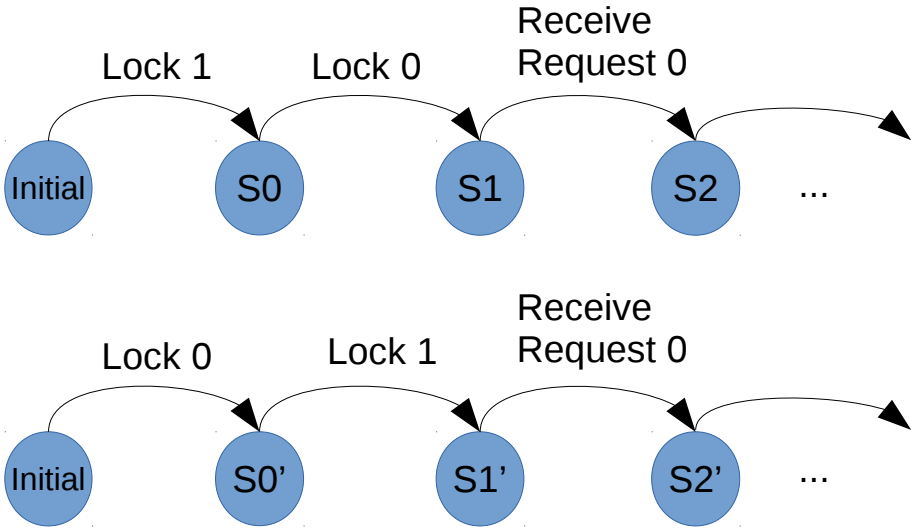
IN COMPUTER NETWORKS

Glenn Ricart
Ashok K. Agrawala

$$(id_1, seq_1) < (id_2, seq_2) \Leftrightarrow seq_1 < seq_2 \vee (seq_1 = seq_2 \wedge id_1 < id\,2)$$

# Представление алгоритма



# Состояние процесса

```
(* A state of a process *)
Record PState :=
  {
    id: ID;                       (* unique id *)
    clock: Clock;                 (* process logical clock *)
    deferred: list RequestSig;    (* list of unresponded requests *)
    await_from: list ID;          (* list of ids we are waiting a reply from *)
    mbox: list Msg;               (* message box of a process *)
    fsm_state: FsmState;          (* abstract machine instruction *)
    deferred_clock : Clock;       (* maximum of all received clocks *)

    (* ************************************************************************ *)
    (* Ghost variables used only for proofs, no control decisions are made using those *)
    (* ************************************************************************ *)
    received_from: list ReplySig; (* list of arrived reply msgs *)
    reply_sent : list ReplySig    (* list of replies we sent to others *)
  }.
```

# Отношение Step

```
Inductive step: GState -> FsmInput -> GState -> Prop :=
  | stepInitial s i:
      fsm_state (s i) = Initial
      -> step s (Lock i) (send_all_requests i s)
  | stepProcessRequestInitial s i j j_clk:
      fsm_state (s i) = Initial
      -> hd_error (mbox (s i)) = Some (Request j j_clk)
      -> step s (ReceiveRequest i j j_clk)
          (uncond_reply i j j_clk s)
  | stepProcessRequest s i j j_clk:
      fsm_state (s i) = ProcessMessages
      -> hd_error (mbox (s i)) = Some (Request j j_clk)
      -> step s (ReceiveRequest i j j_clk)
                (process_messages i s)
  | stepProcessReply s i j j_clk i_clk:
      fsm_state (s i) = ProcessMessages
      -> hd_error (mbox (s i)) =
            Some (Reply j j_clk i_clk)
      -> step s (ReceiveReply i j j_clk)
                (process_messages i s)
  | stepLocked s i:
      fsm_state (s i) = Locked
      -> step s (Unlock i) (unlock i s).
```

# Отношение StepStar

```
Inductive step_star:
  State -> list Label -> State -> Prop :=
| refl s:
    step_star s nil s
| steps s1 ls s2 l s3:
    step_star s1 ls s2
    -> step s2 l s3
    -> step_star s1 (ls ++ [l]) s3.
```

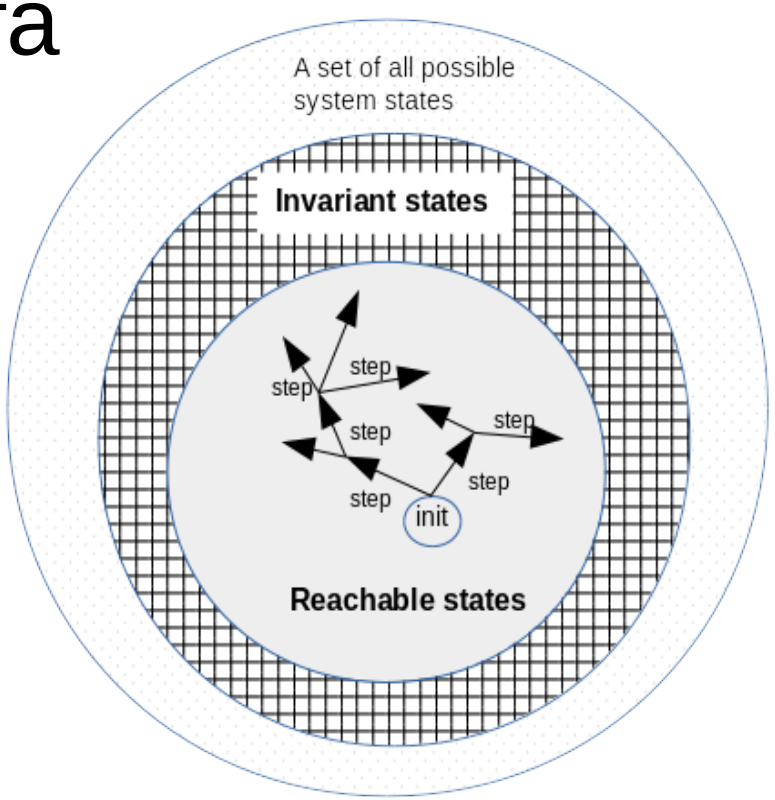
# Основная теорема надежности

```
Theorem ra_mutex_safety:
  forall st t i j,
    Clients <> []
      -> i <> j
      -> step_star Ginit_state t st
      -> fsm_state (st i) = Locked
      -> ~ fsm_state (st j) = Locked.
```

*Для любого достижимого состояния и для любой пары процессов справедливо то, что если в данном состоянии один процесс обладает блокировкой, то другой процесс не может обладать блокировкой в этом же состоянии.*

# Техника доказательств

- Нахождение инварианта
- Переписывание
- Индукция
- Теневые переменные
- Теория StepStar
- Автоматизация доказательств (ltac)




Invariant P <=> P init /\
(step st op st' /\ P st → P st')

Кол-во строк спецификаций: 1915, количество строк доказательств: 2156

# Инварианты

```
Definition locked_implies_two_things (st: GState) :=
  forall i,
    (fsm_state (st i) = Locked
       \/ fsm_state (st i) = Initial)
    -> await_from (st i) = [].

Definition rf_plus_af_equ_inv (st: GState) :=
  forall i j,
    fsm_state (st i) <> Initial
    -> In j (known_to i)
    -> In j (get_id (received_from (st i)) ++
                await_from (st i)).
```

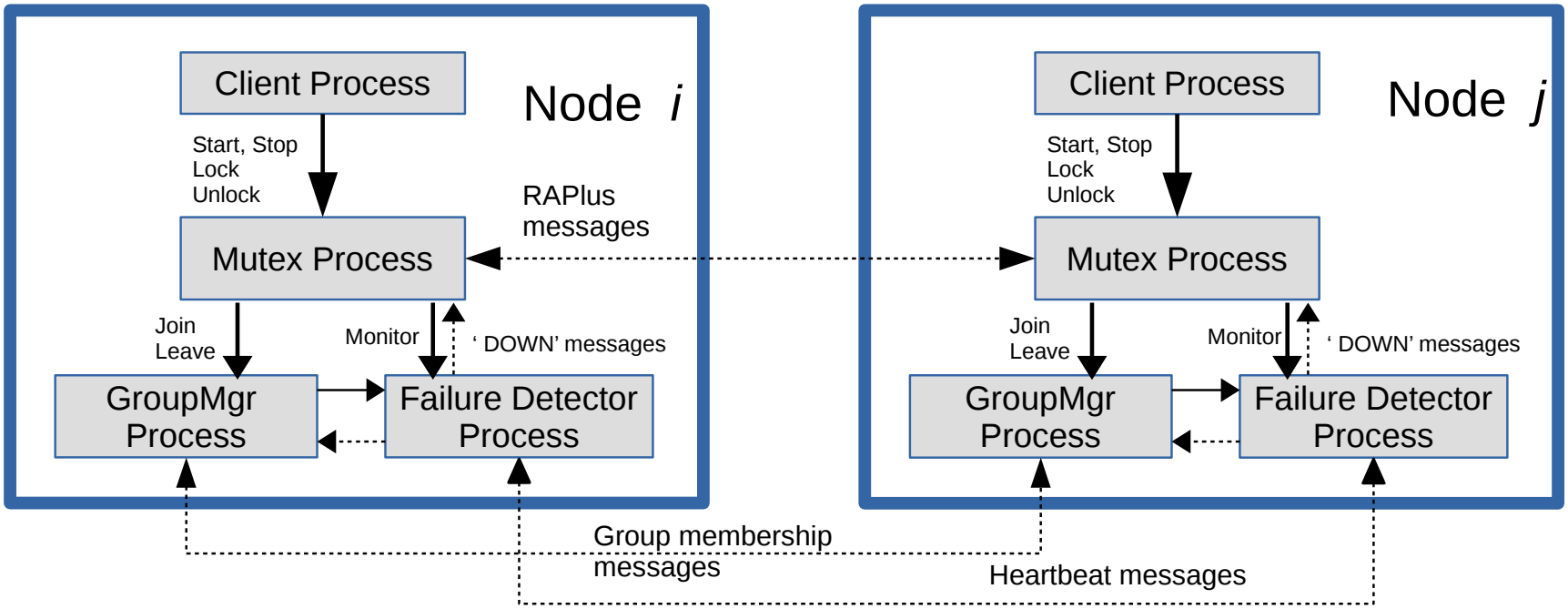
- 10 инвариантов потребовалось для успешного доказательства основной теоремы

  (safety)

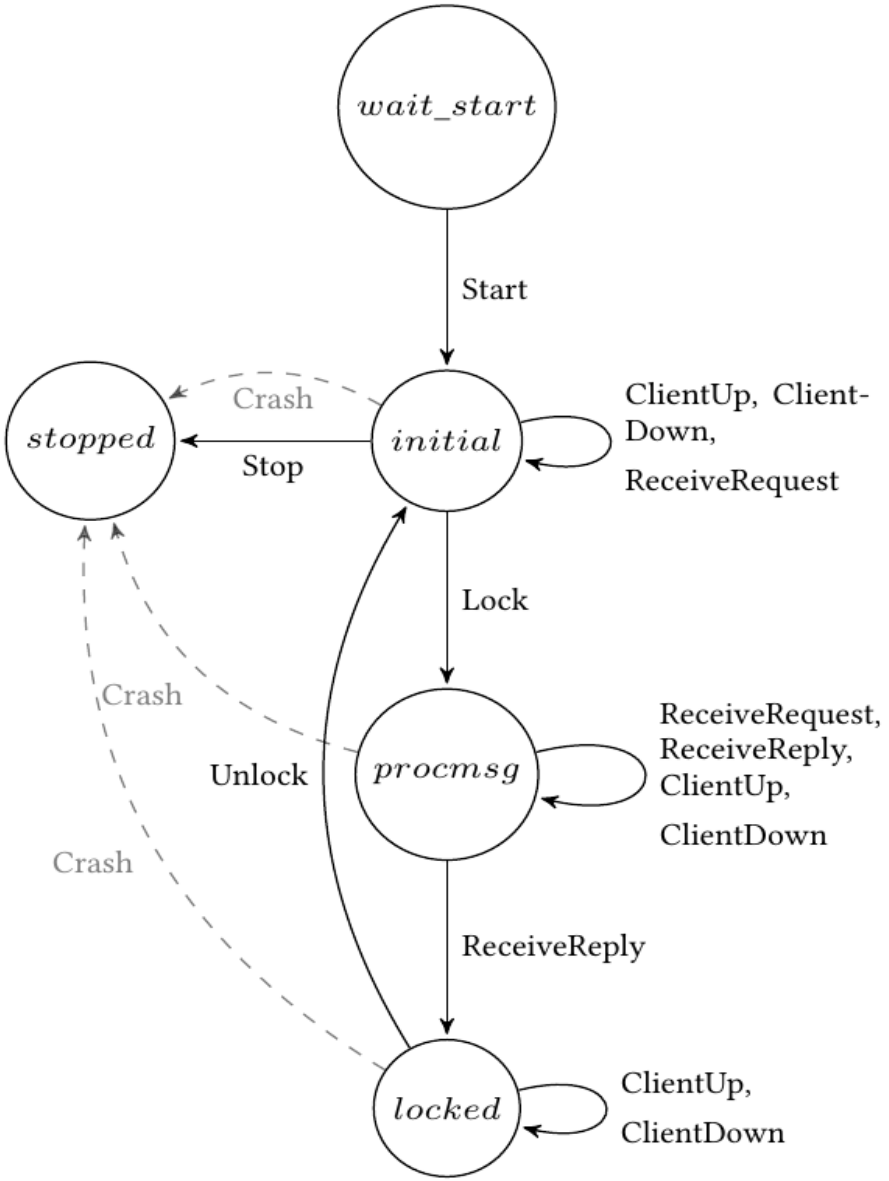
# Недостатки классического алгоритма

- Не способен переживать отказы отдельных процессов
- Не способен принимать новые процессы в группу претендентов за ресурс
- Процесс не способен покинуть группу претендентов за ресурс; даже если он не претендует за доступ, ему по-прежнему необходимо отвечать остальным процессам на их запросы
- Как устранить эти недостатки?

# Расширенная модель системы



# Система переходов RAPlus



# Свойства надежности и прогресса

**Definition 4.4** (Safety property).

$$\forall i, j \in Process : \Box(i \neq j \Rightarrow$$
$$(fsm\_state[i] = Locked \Rightarrow fsm\_state[j] \neq Locked)) \quad (4)$$

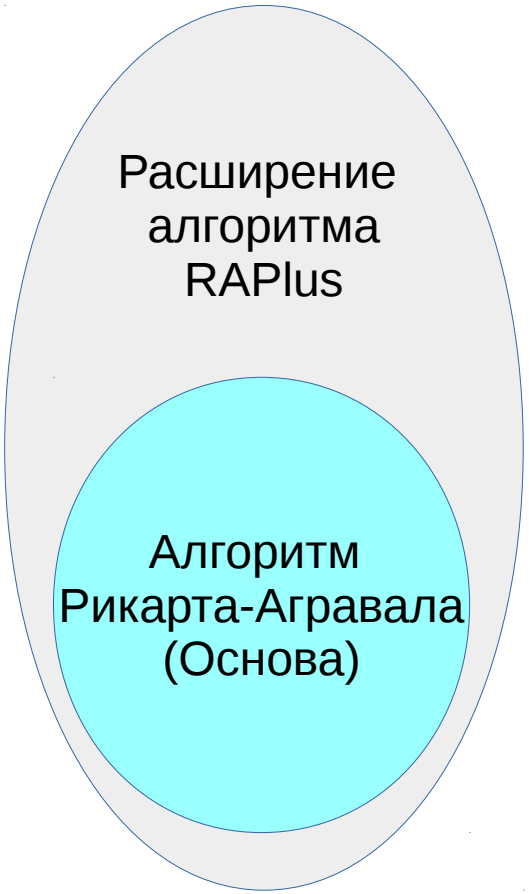
**Definition 4.5** (Progress property).

$$\forall i \in Process : fsm\_state[i] = ProcessMessages \rightsquigarrow$$
$$(fsm\_state[i] = Locked \vee fsm\_state[i] = Stopped) \quad (5)$$

N = макс. кол-во процессов в системе
L = макс. кол-во запросов на доступ к ресурсу
C = макс. кол-во процессов, которым разрешается “умереть”

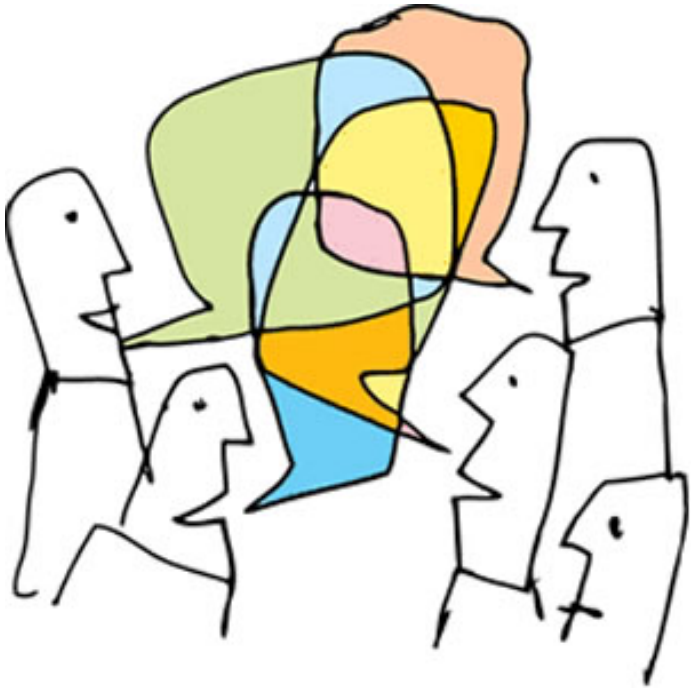
TLA+ модель данного расширенного алгоритма была проверена на таких параметрах:
(N = 3, L = 3, C = 1), (N = 3, L = 2, C = 2)
Для (N = 4, L = 2, C = 2) проверка не завершилась в разумные сроки (более суток работы)
Железо: 32-core Intel Xeon, 1TB

# Вклад работы




- Свойство надежности алгоритма Рикарта-Агравала формализовано и доказано в системе Coq, используя модель процессов с почтовыми ящиками
- Представлено расширение алгоритма (новый алгоритм назван RAPlus), устраняющее недостатки исходного алгоритма
- Свойства надежности и прогресса нового алгоритма были проверены на ограниченной модели в TLA+
- Артифакты доступны для изучения (исходный код на Erlang, формализация в Coq, модель TLA+)

# Дискуссия



**Email**: evgeniy.shishkin@gmail.com

**Repo**: https://bitbucket.org/unboxed_type/ra_mutex/src