

Сбои, иерархия отказов, консенсус



Илья Кокорин

kokorin.ilya.1998@gmail.com

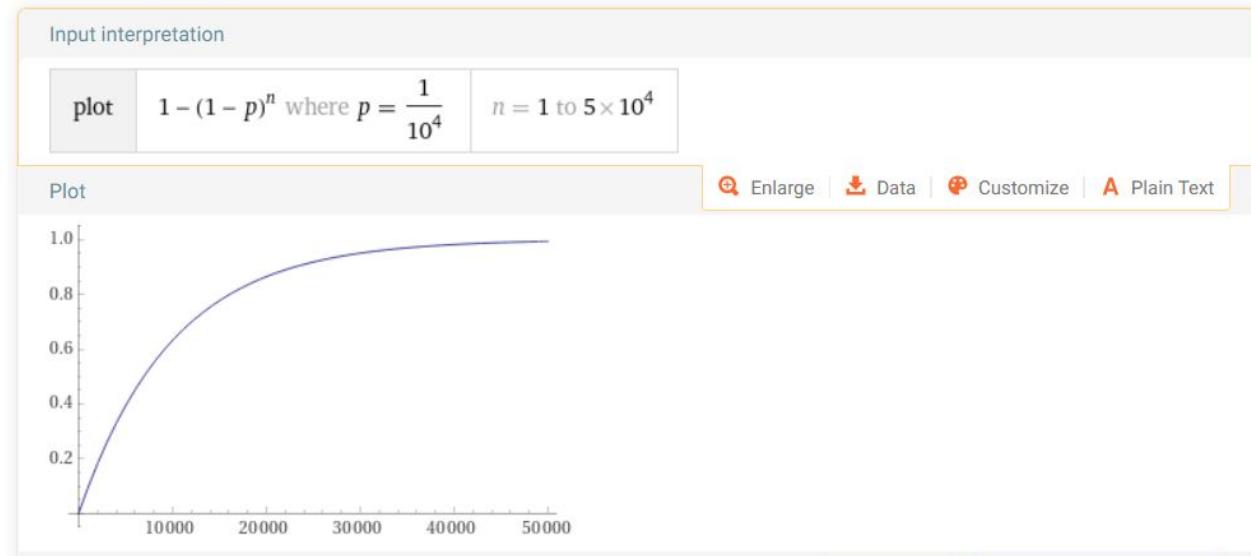
Частичные отказы

Вероятность безаварийной работы одного узла равна $1 - p$

Вероятность безаварийной работы всех узлов равна $(1 - p)^n$

Вероятность сбоя хотя бы одного узла равна $1 - (1 - p)^n$

- Система должна продолжать работать при частичных отказах



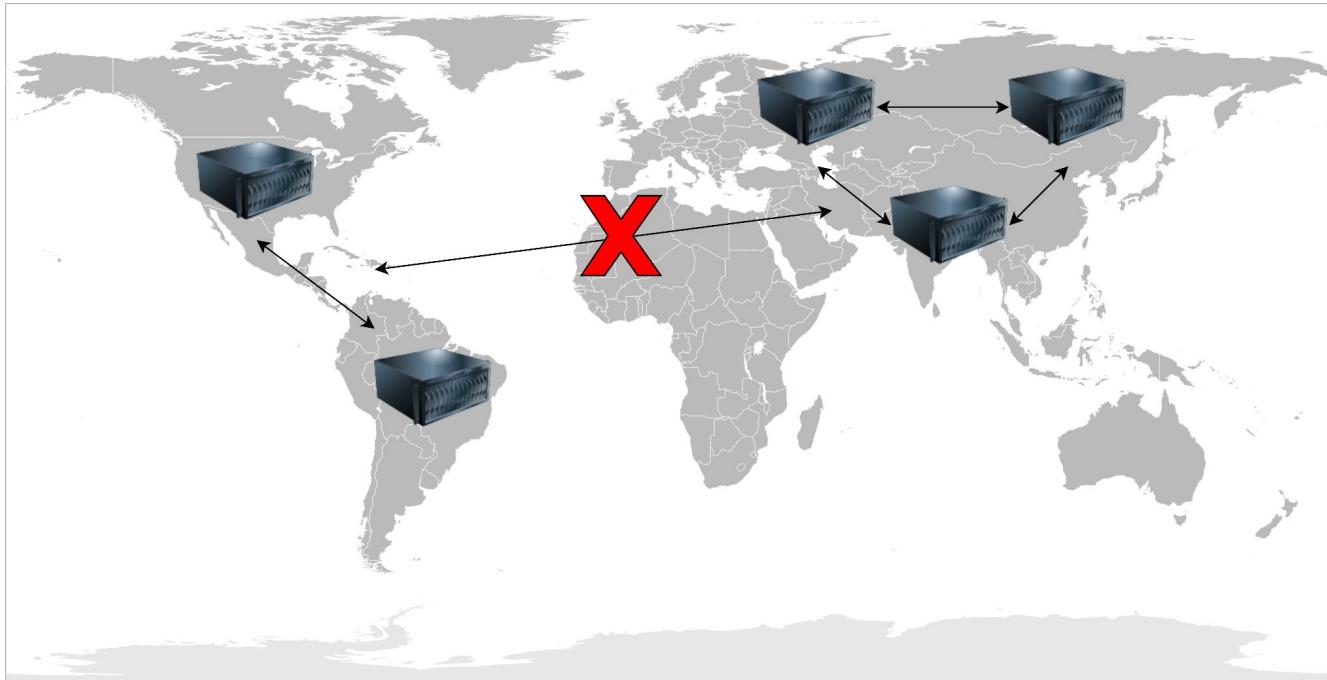
Частичные отказы

- *First, component failures are the norm rather than the exception. ... The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures.*
 - The Google File System



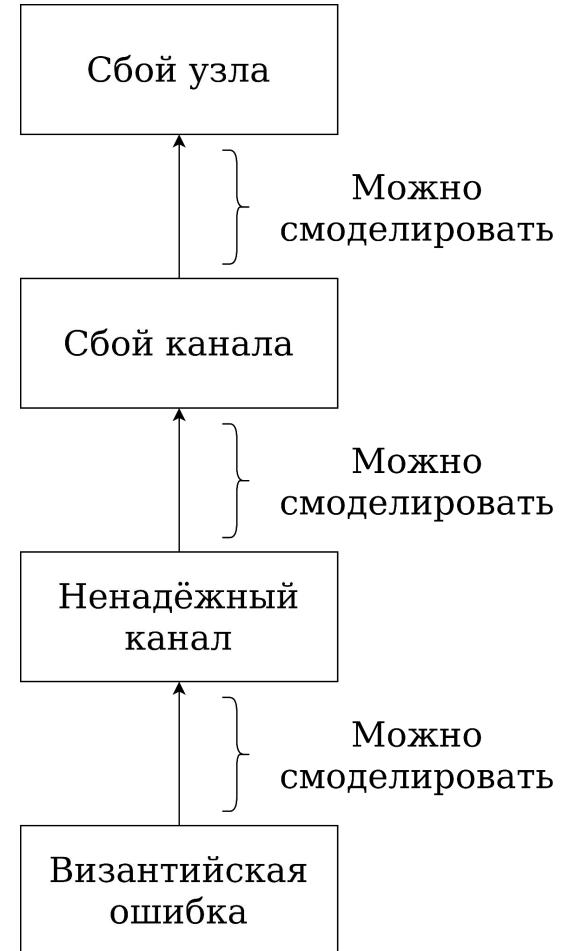
Частичные отказы: обрывы связи

- Система должна работать в условиях ненадёжной доставки сообщений



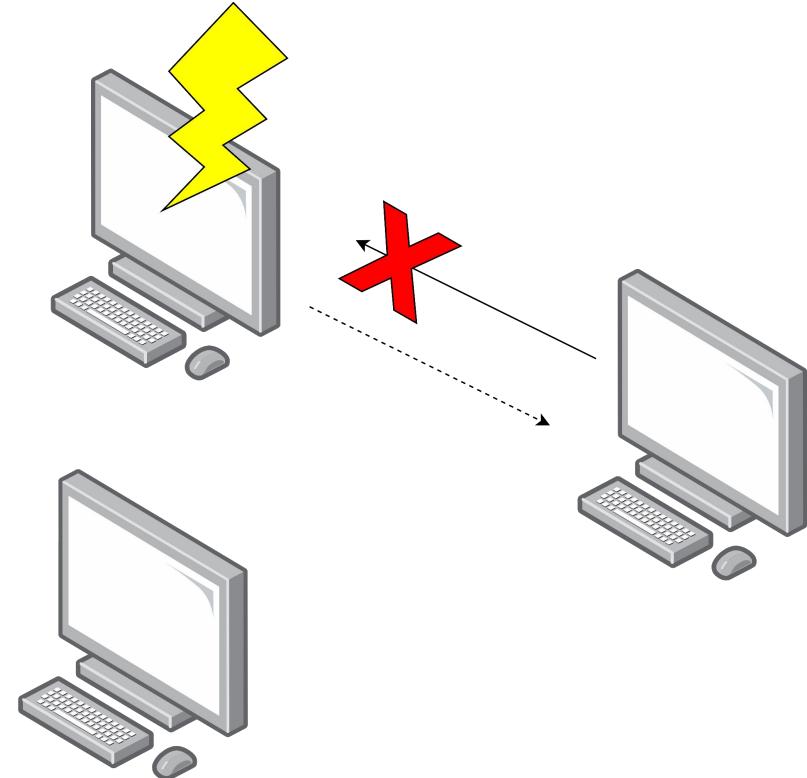
Иерархия отказов

- Выстраиваем отказы в иерархию
- Верхние отказы моделируются с помощью нижних
- Если алгоритм умеет работать при нижнем отказе, значит будет и при верхнем



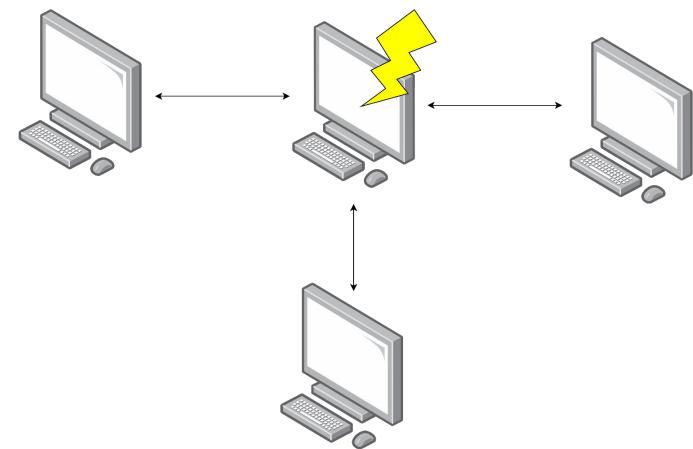
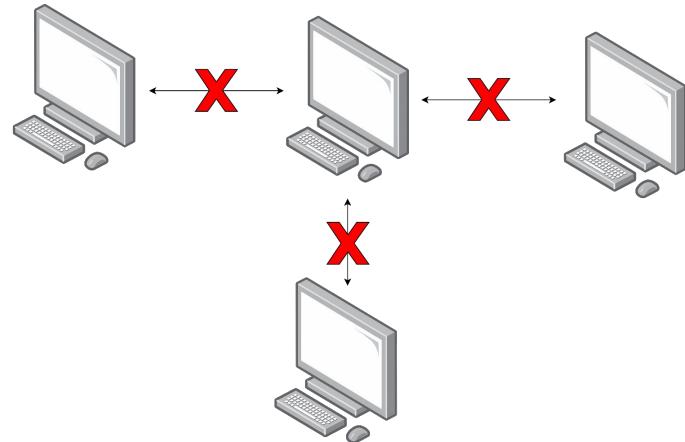
Отказ узла

- Один процесс останавливается
- Не производит вычисления
- Не получает сообщения
- Не посыпает сообщения



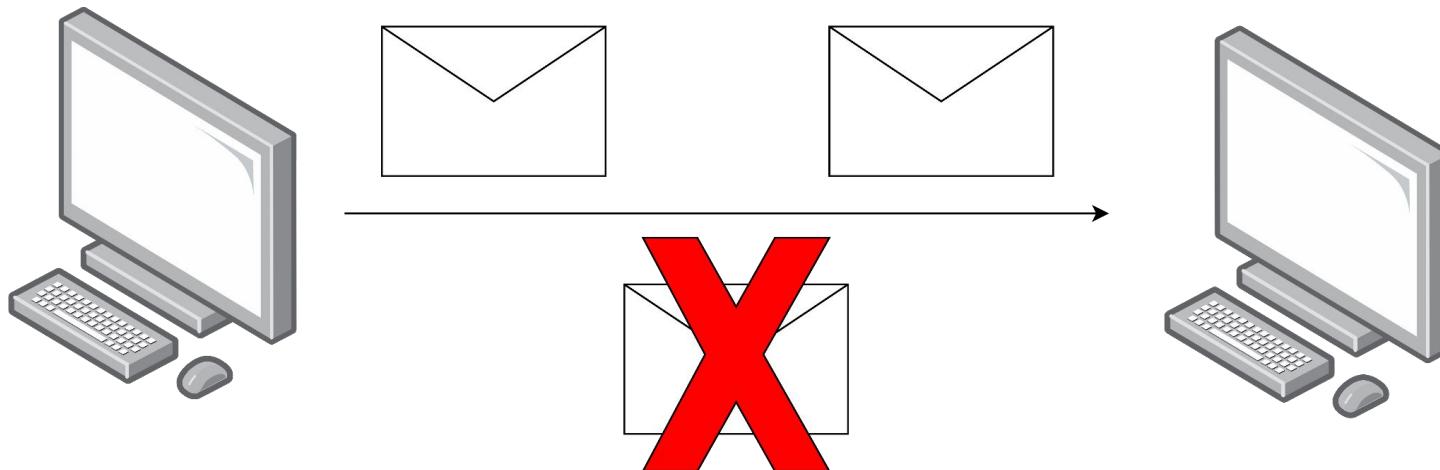
Отказ канала

- Отказ узла можно смоделировать отказом всех каналов этого узла
- Более сильная ошибка



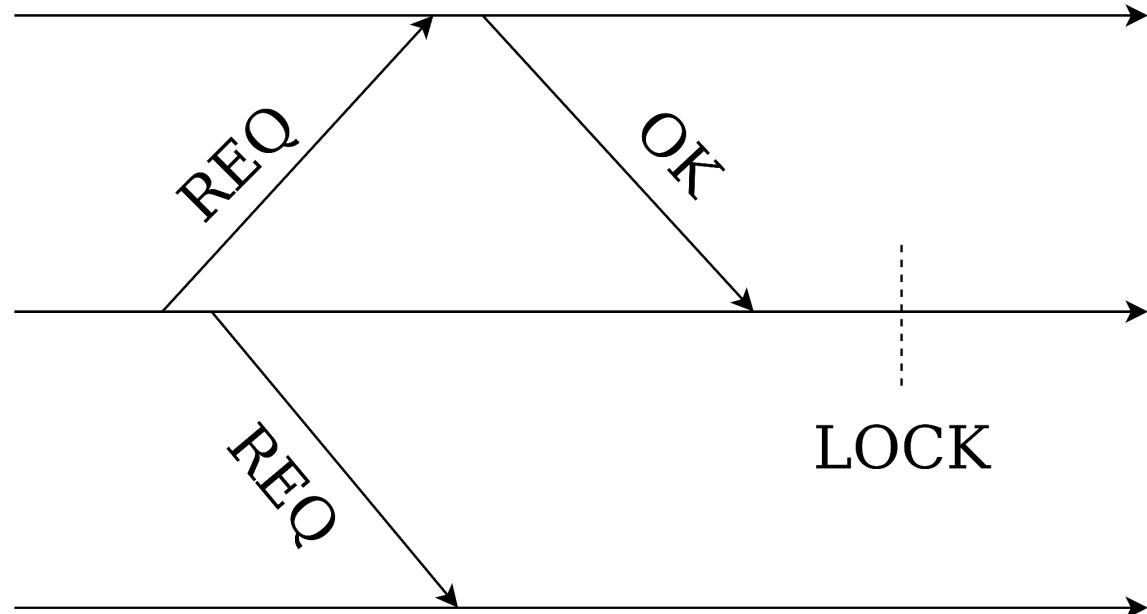
Ненадёжная доставка

- Некоторые сообщения теряются в пути
- Можно смоделировать отказ канала с помощью ненадёжной доставки
 - $P(\text{drop}) = 1$



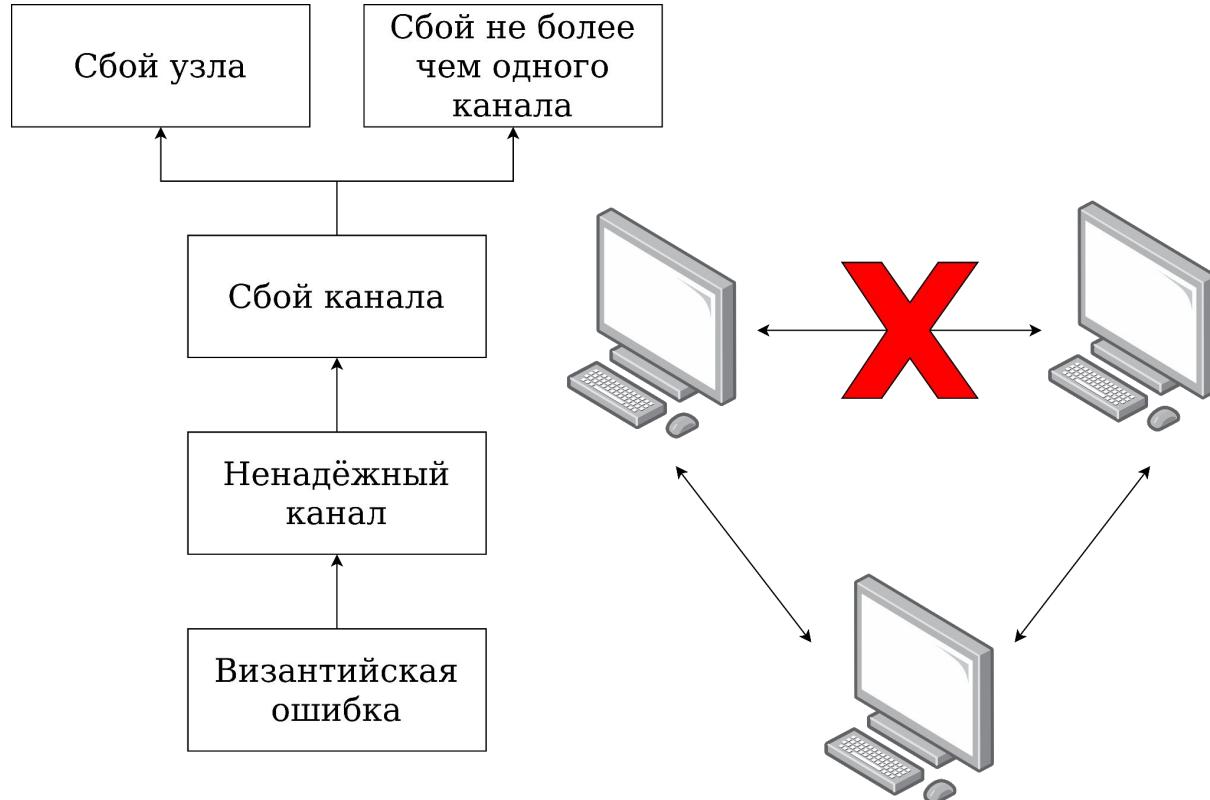
Византийская ошибка

- Процесс действует по произвольному алгоритму
 - Потенциально не по тому, который мы написали
- Злоумышленник захватил контроль над сервером и запустил на нём свой код



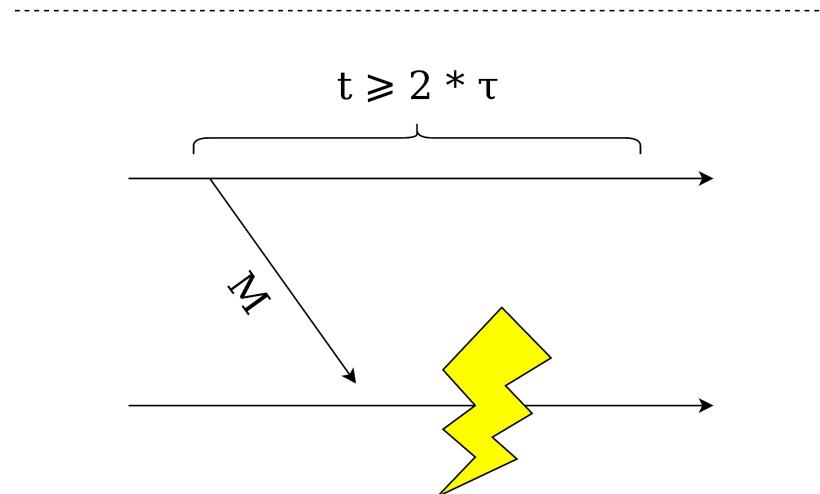
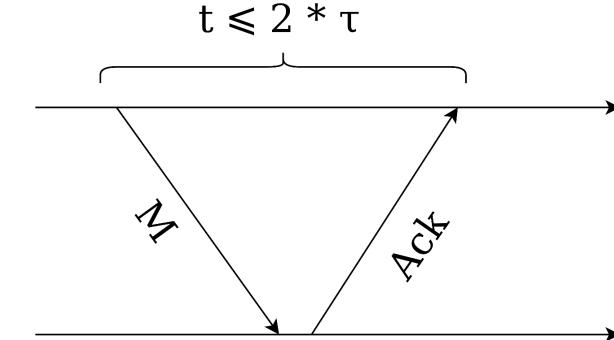
Более сложные иерархии

- Сбой узла невозможно промоделировать сбоем не более чем одного канала



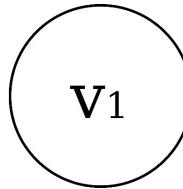
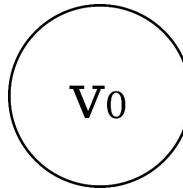
Синхронные и асинхронные сети

- В синхронной сети сообщение доходит до получателя не более чем за t секунд
- Ответ придет не более чем за $2t$ секунд
- Если ответ не пришёл — делаем вывод, что получатель отказал
- В асинхронной сети время доставки не ограничено сверху

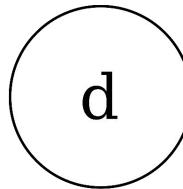
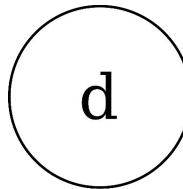
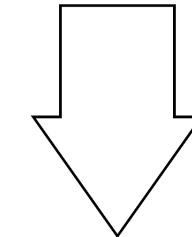
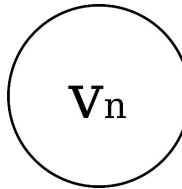


Задача о консенсусе

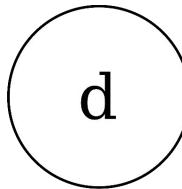
- У каждого процесса есть предложение v_i
- В конце алгоритма все процессы приходят к единому решению d



...

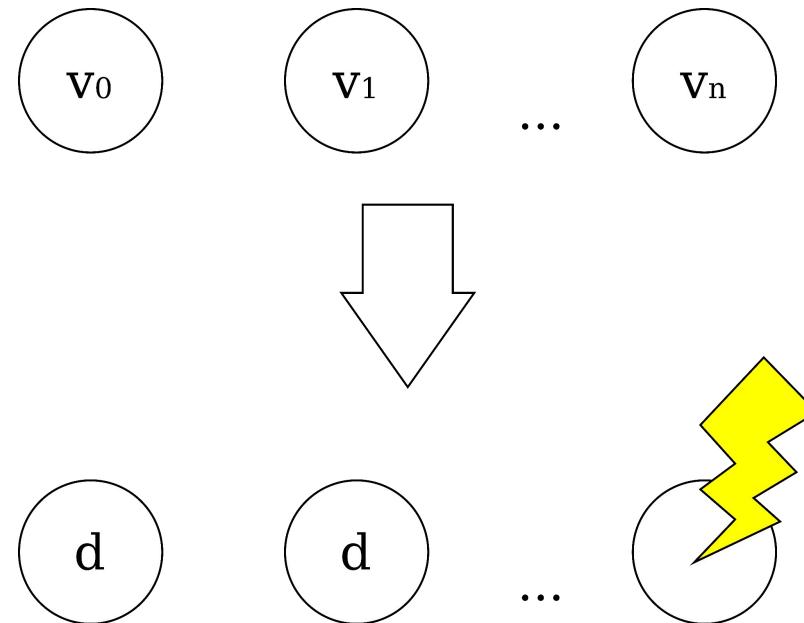


...



Задача о консенсусе: согласие

- Все **несбойные** процессы приходят к одному решению
- Сбойные могут не прийти к решению вообще



Задача о консенсусе: нетривиальность

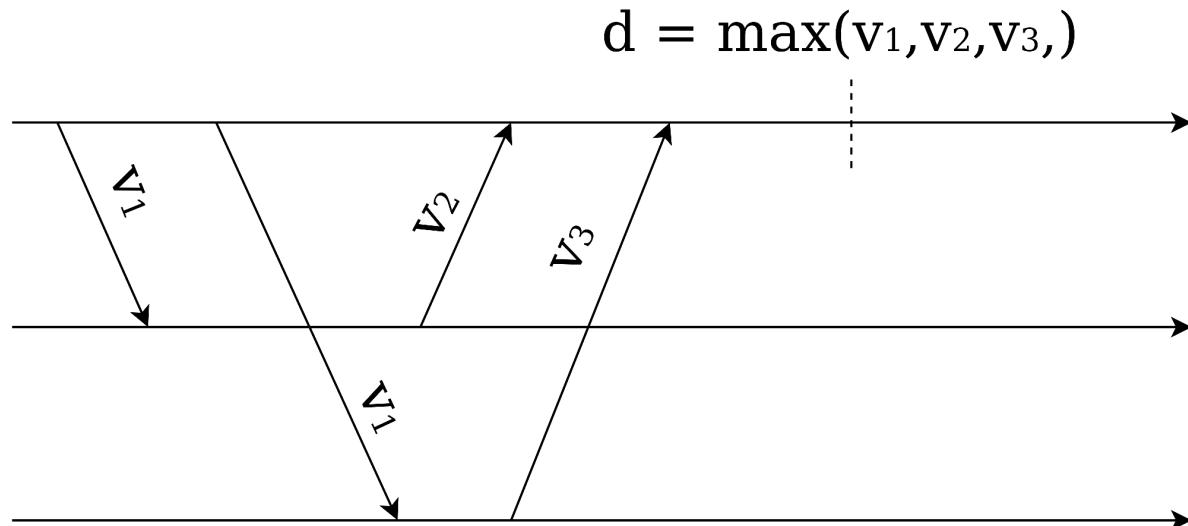
- Должны быть варианты исполнения, приводящие к разным решениям
 - Иначе { `return 0;` } было бы корректным алгоритмом консенсуса
- Обоснованность
 - Более сильное свойство
 - $d \in \{v_1 \dots v_n\}$
 - Решение должно быть предложением одного из процессов

Задача о консенсусе: завершение

- Несбойные процессы должны приходить к решению за конечное время
 - Сбойные процессы к решению могут не приходить
 - Иначе `while (true) {}` было бы корректным алгоритмом консенсуса

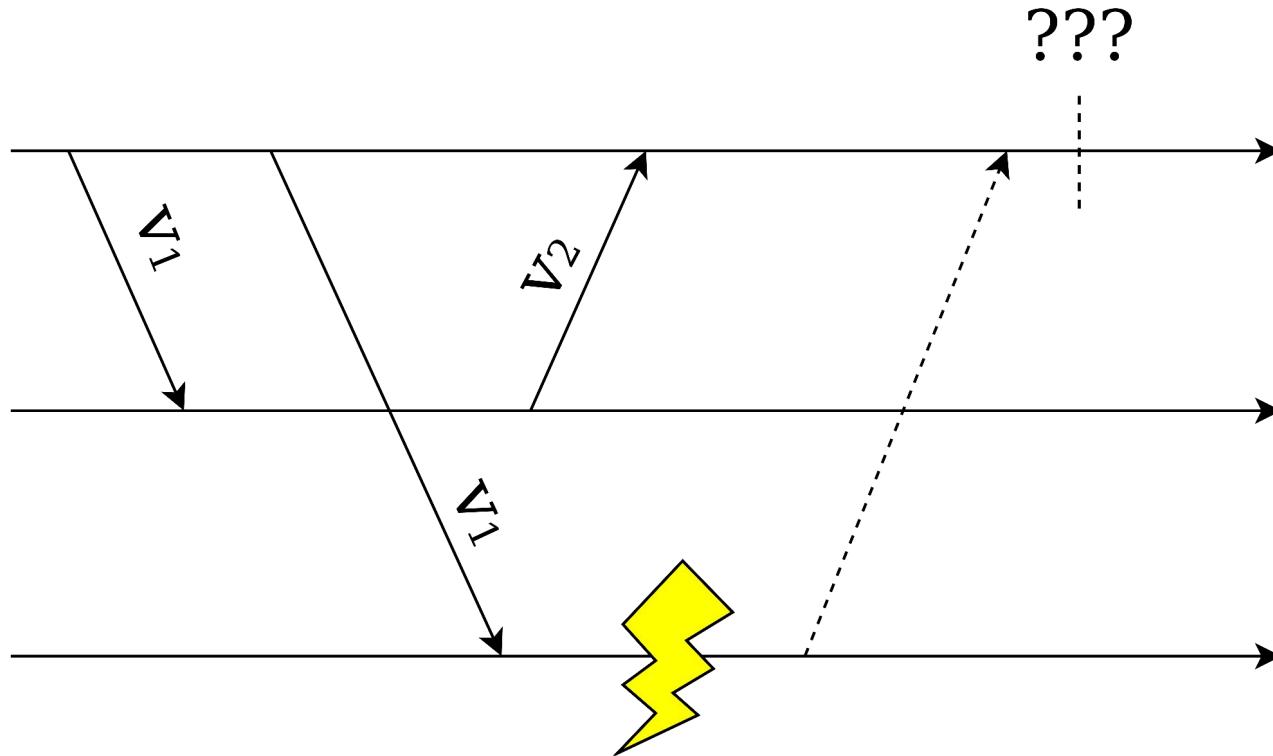
Консенсус в асинхронной системе без отказов

- Рассыпаем своё предложение всем
- Дожидаемся от каждого процесса его предложения
- Детерминированной функцией выбираем из полученного множества
- Множества одинаковые на всех процессах
- Значит, и решения одинаковые



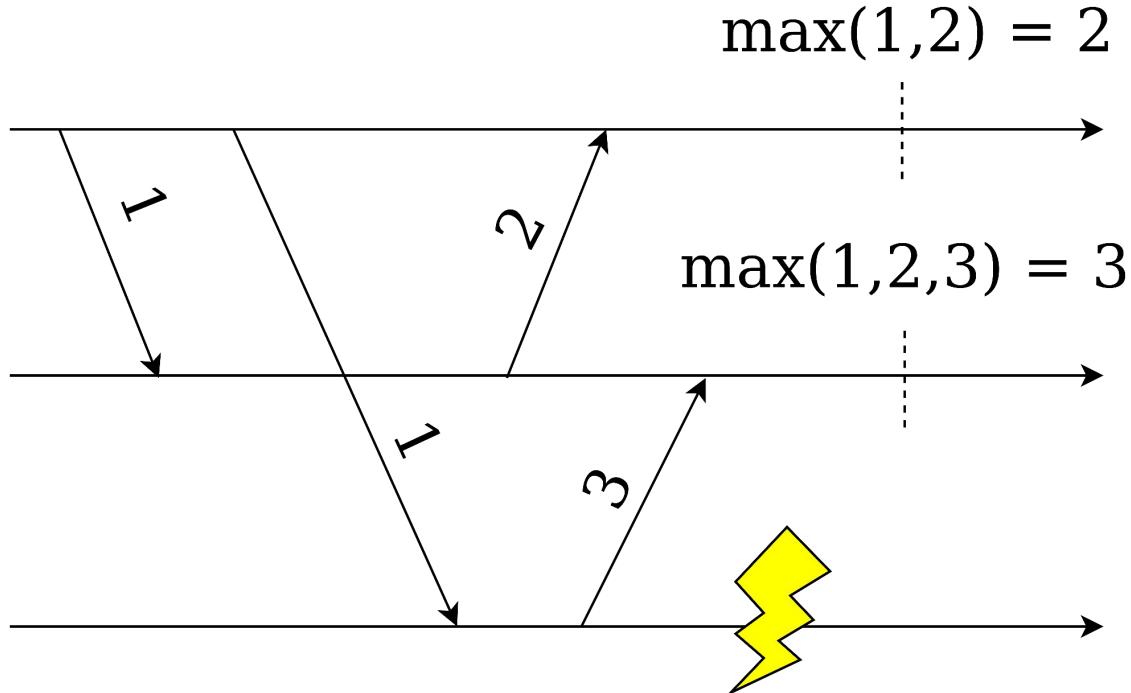
Консенсус в системе с отказами

- Никогда не дождёмся предложения от сбояного процессса



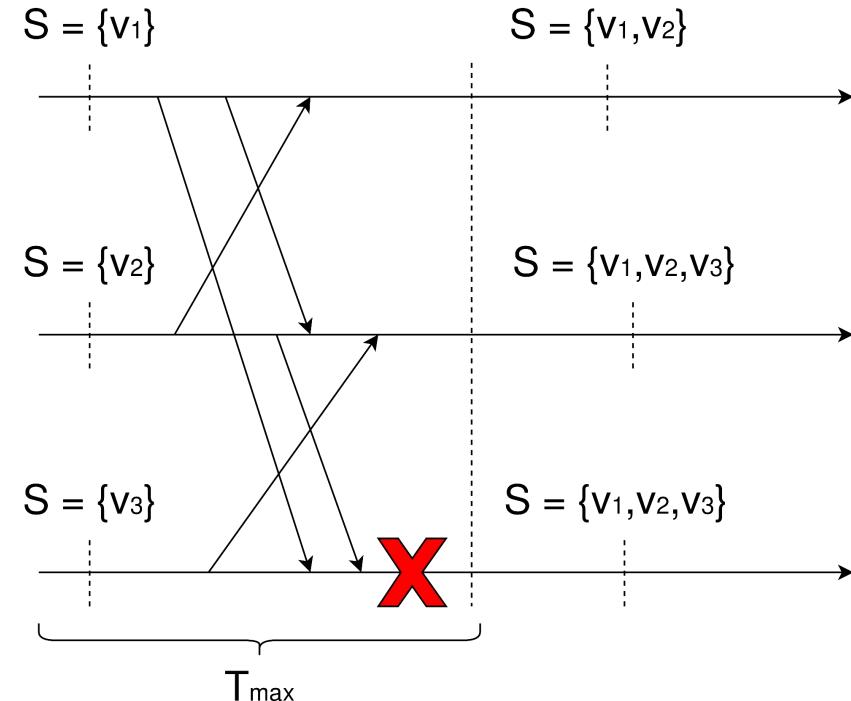
Консенсус в системе с отказами

- Нельзя просто дождаться не всех
- Сбойный процесс может отправить сообщение не всем до сбоя
- Не работает даже в синхронной сети



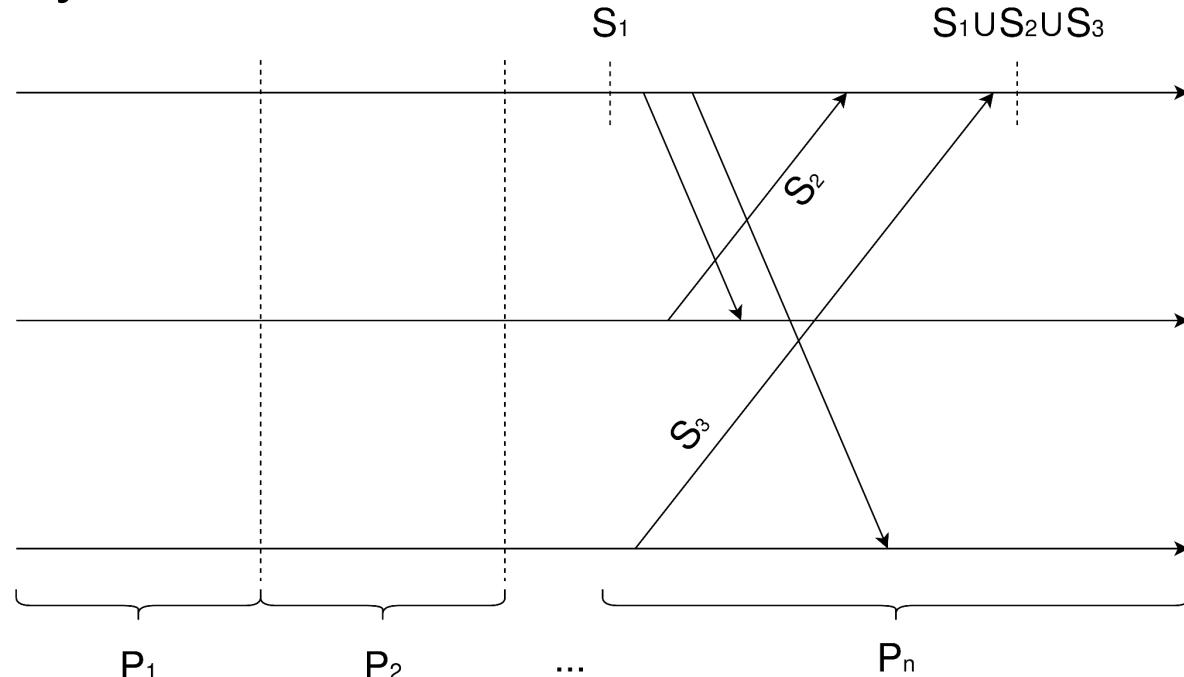
Консенсус в синхронной системе с отказами

- Алгоритм разбит на фазы
- У каждого процесса есть множество известных ему предложений S
- Изначально $S = \{v_i\}$
- Рассылаем S всем процессам
- Получив множество S' от другого процесса, объединяем S и S'
- Можем рассчитать максимальное время, нужное для завершения фазы



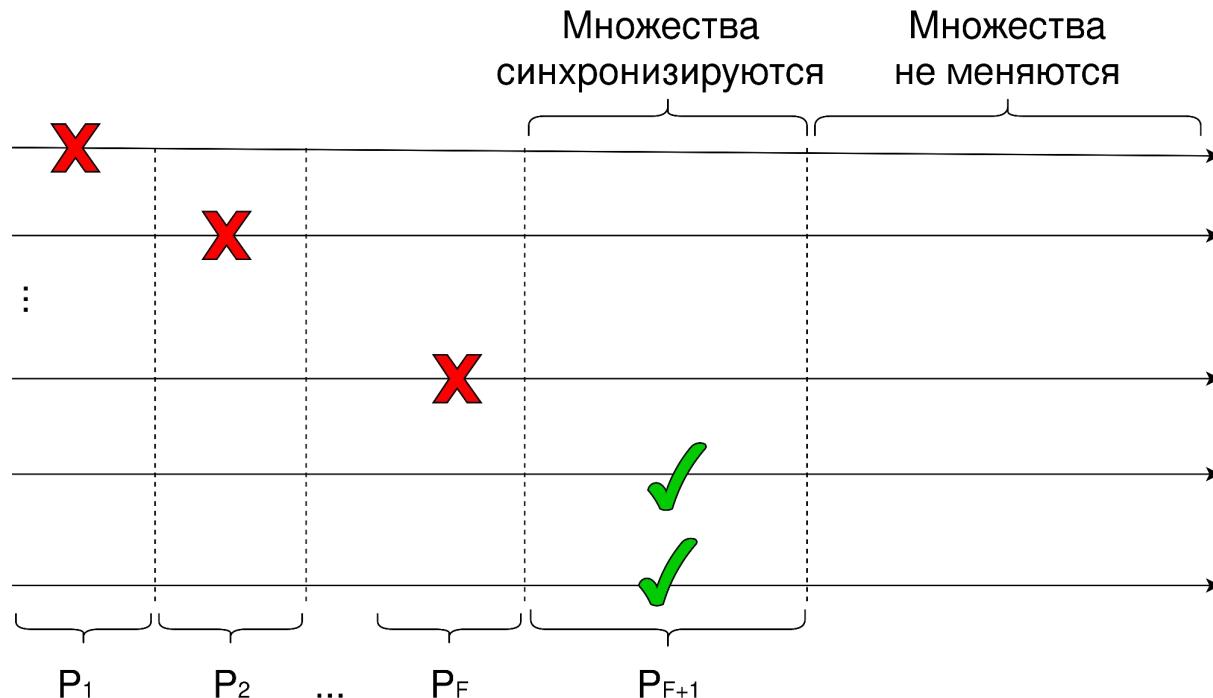
Консенсус в синхронной системе с отказами

- В каждой фазе рассылаем всем множество известных нам предложений
- Ждём не более T секунд
- Объединяем
наше
множество с
полученными
- Переходим к
следующей
фазе



Консенсус в синхронной системе с отказами

- Чтобы пережить сбой F узлов нужна как минимум $F + 1$ фаза
- По принципу Дирихле, как минимум в одной фазе не будет сбоев
- После не множества могут совпадать



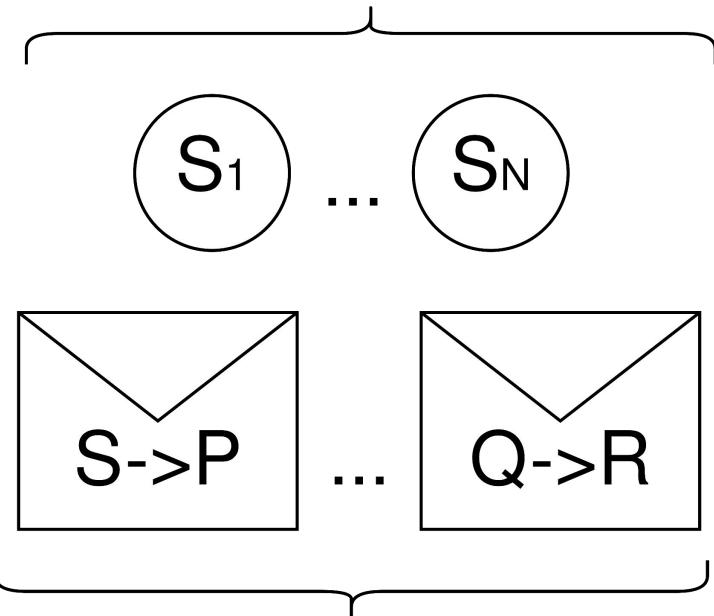
Невозможность консенсуса

- Результат Фишера, Линч и Патерсона
 - Сеть асинхронная
 - Возможен отказ хотя бы одного узла
 - Алгоритм детерминированный
 - Несбойные узлы должны прийти к нетривиальному консенсусу за конечное число шагов
 - Невозможно прийти к консенсусу даже на одном бите (1 или 0)
- Докажем от противного
 - Предположим, что такой алгоритм существует

FLP: модель

- Состояние системы это:
 - Состояния всех процессов
 - Все сообщения в пути
- Все сообщения рано или поздно будут обработаны
 - Если они не предназначены сбийному процессу

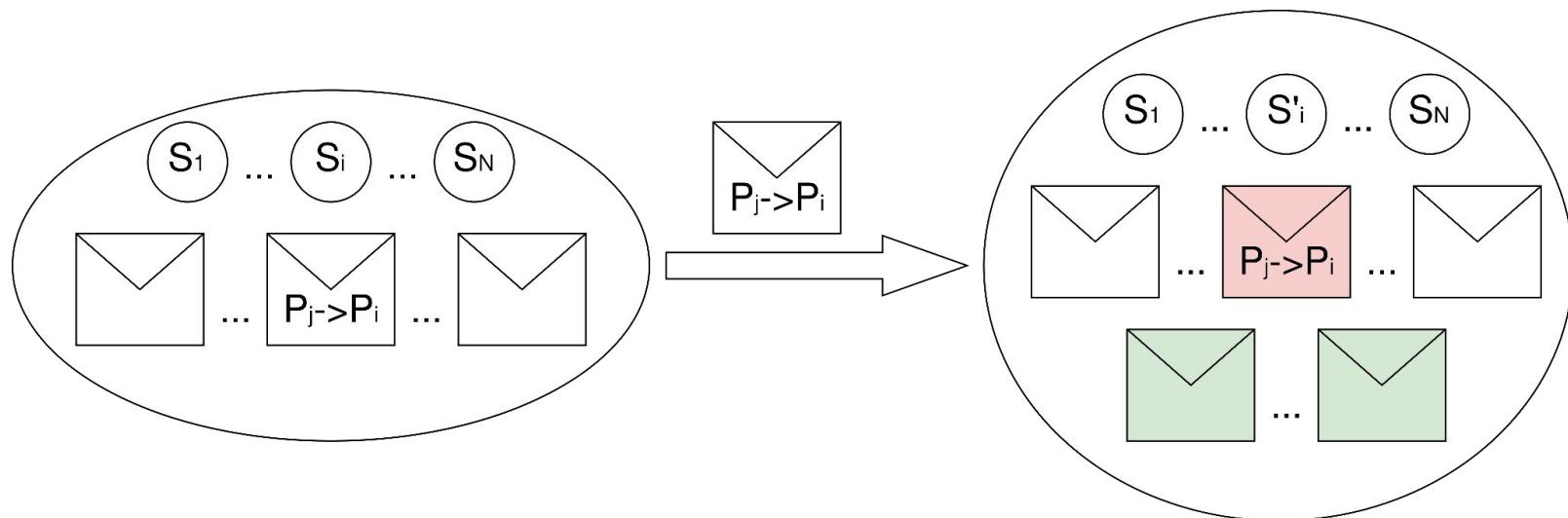
Состояния процессов



Сообщения в пути

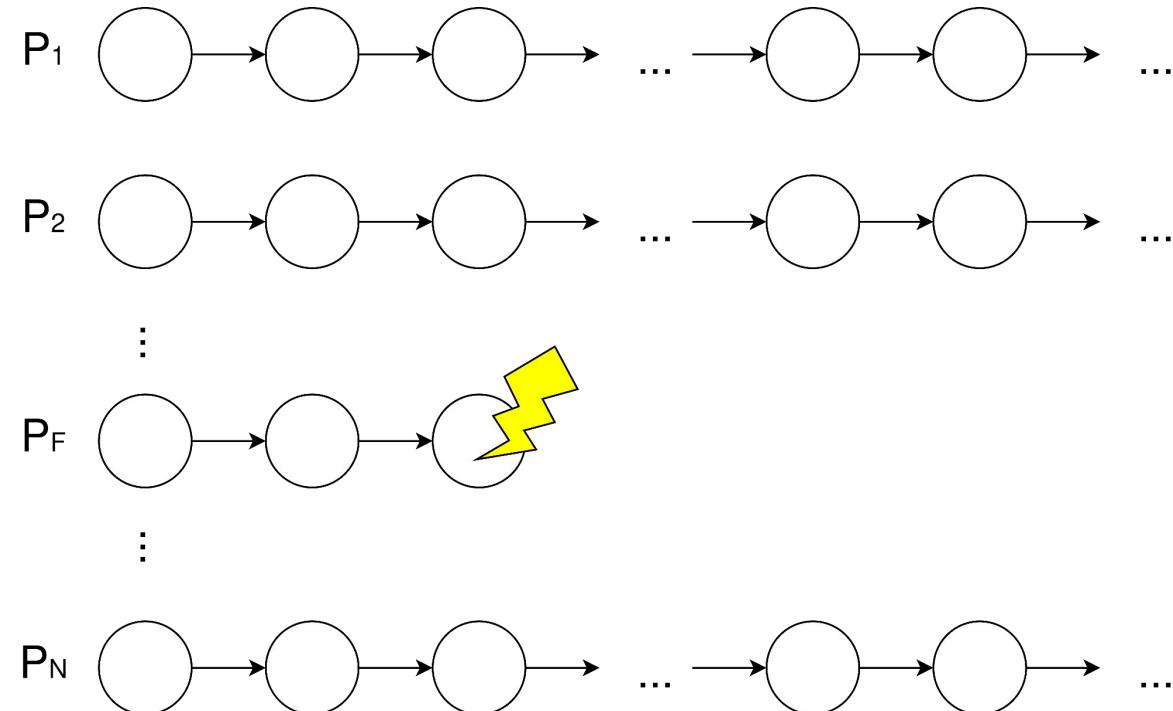
FLP: модель

- Шаг это обработка одного сообщения
- Меняется состояние процесса-получателя
 - Состояние отправителя остаётся прежним
- Обработанное сообщение удаляется
- Добавляется произвольное число новых сообщений



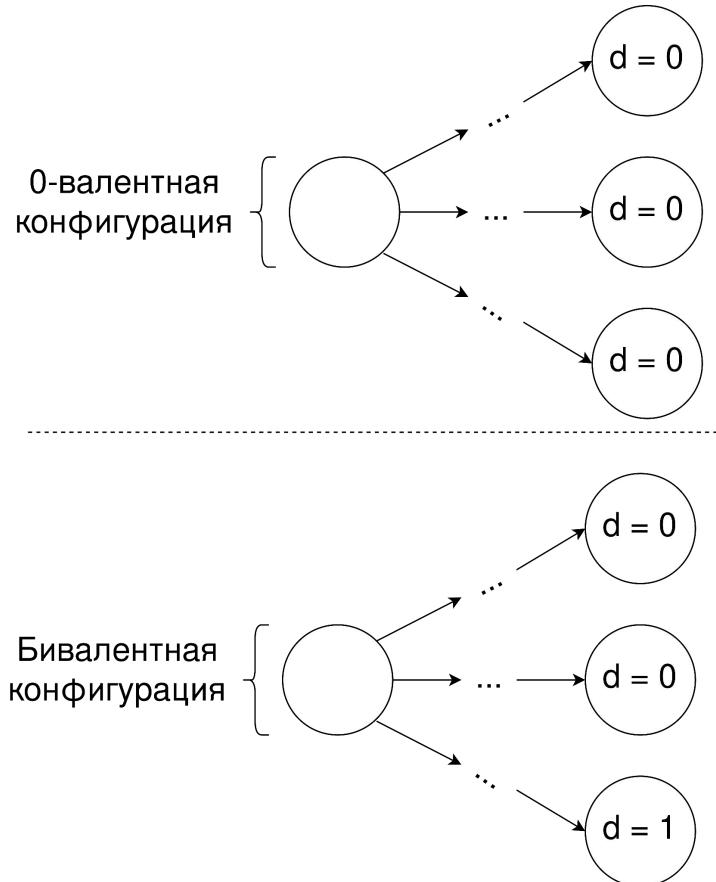
FLP: модель

- Не более одного процесса совершает конечное число шагов
- Несбойные процессы продолжают исполняться и после того, как приняли решение
- Чтобы сообщать о решении другим процессам



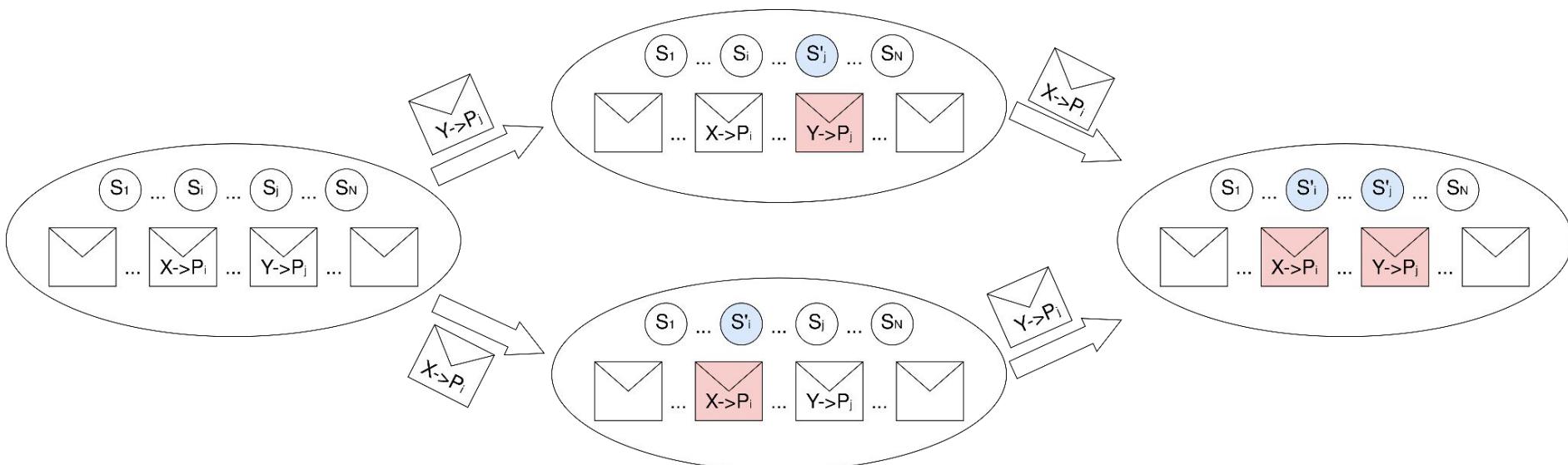
FLP: валентность конфигурации

- Конфигурация называется i -валентной, если все цепочки шагов из неё приводят к решению i
 - $i \in \{0, 1\}$
- Конфигурация называется бивалентной, если из неё существуют цепочки шагов как к решению 0, так и к решению 1



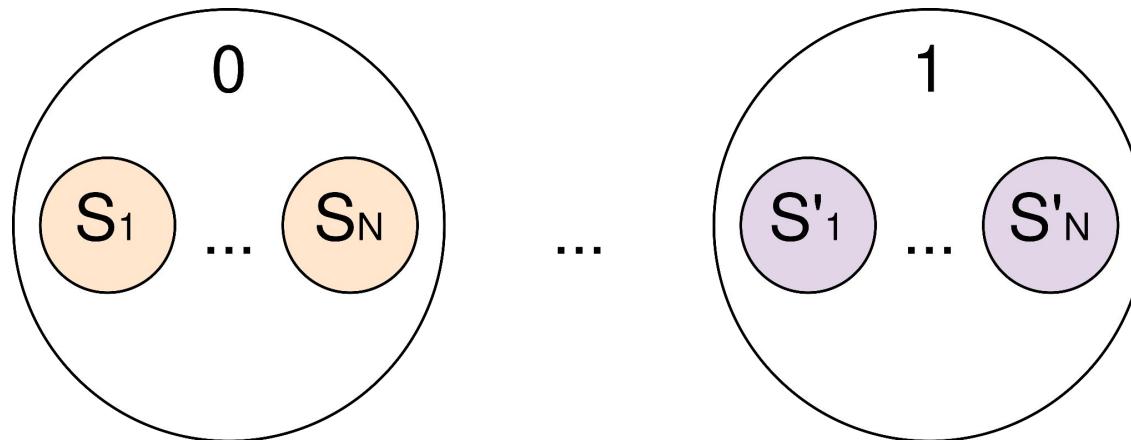
FLP: коммутирующие события

- Есть два сообщения M_1 и M_2 , которые должны быть получены разными процессами
- Порядок обработки не важен
 - Финальный результат будет одинаковым



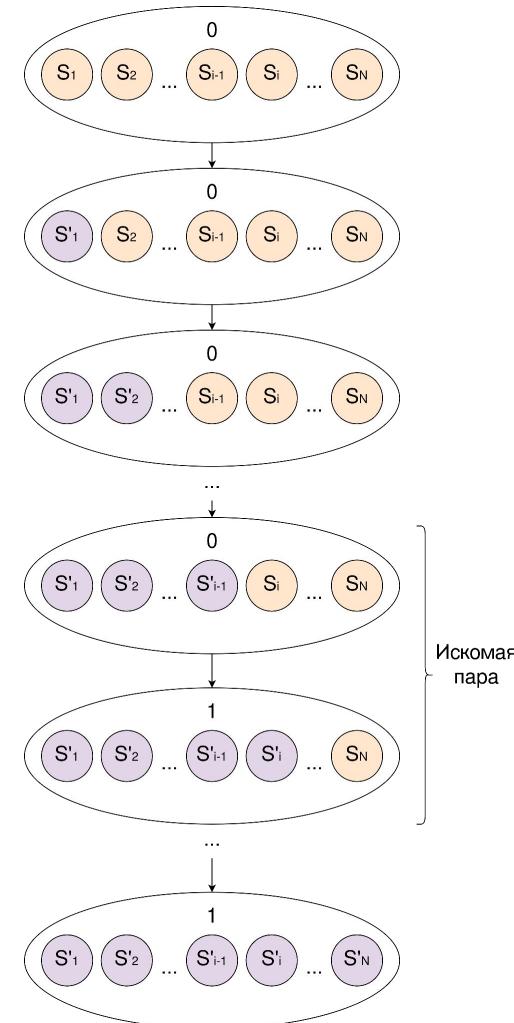
FLP: Лемма

- Существует начальная бивалентная конфигурация
- Предположим, что нет
- Существует пара начальных конфигураций разной валентности
 - Иначе консенсус будет тривиальным



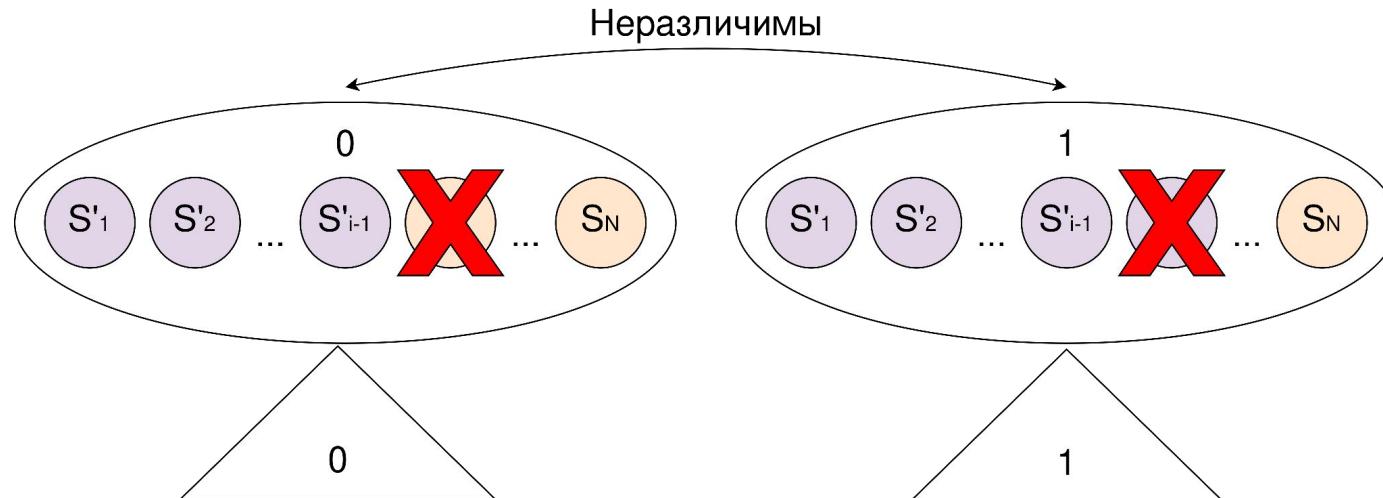
FLP: Лемма

- Находим пару начальные конфигурации разной валентности
 - Различаются начальными состояниями процессов
- Начинаем менять начальные состояния процессов по одному за раз
- Находим две соседние конфигураций разной валентности
 - Отличаются состояниями одного процесса



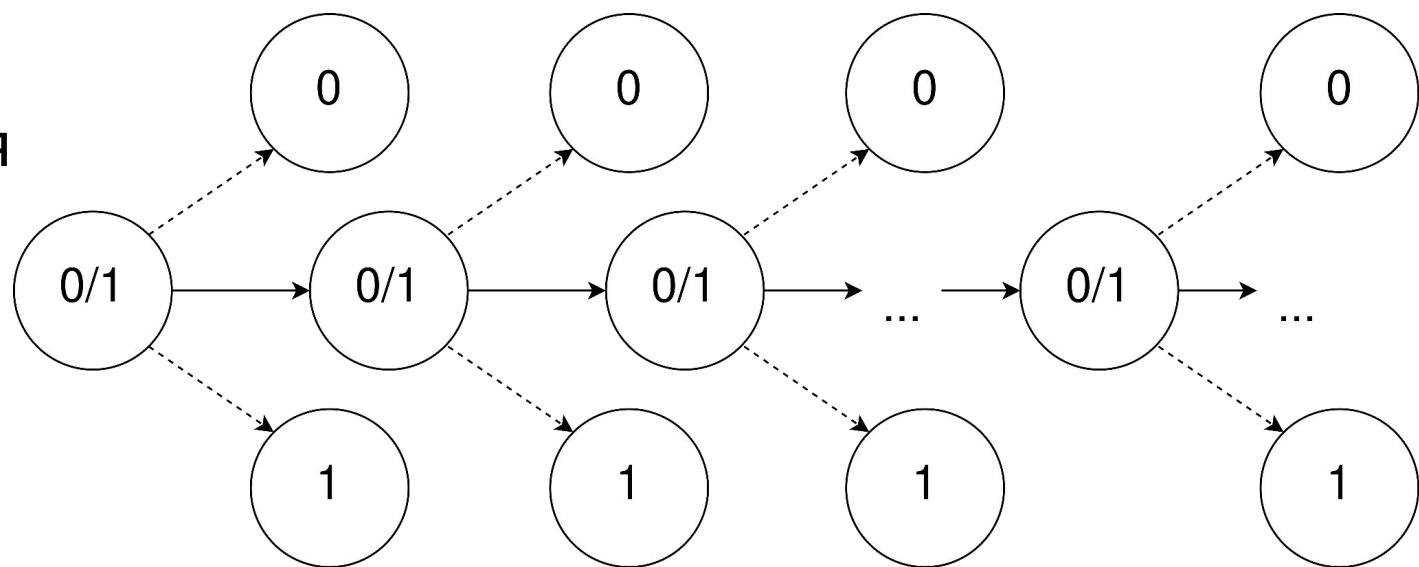
FLP: Лемма

- Существует две соседние конфигурации разной валентности, отличающиеся состояниями одного процесса
- Процесс может отказать с самого начала
 - Конфигурации неразличимы
- Противоречие



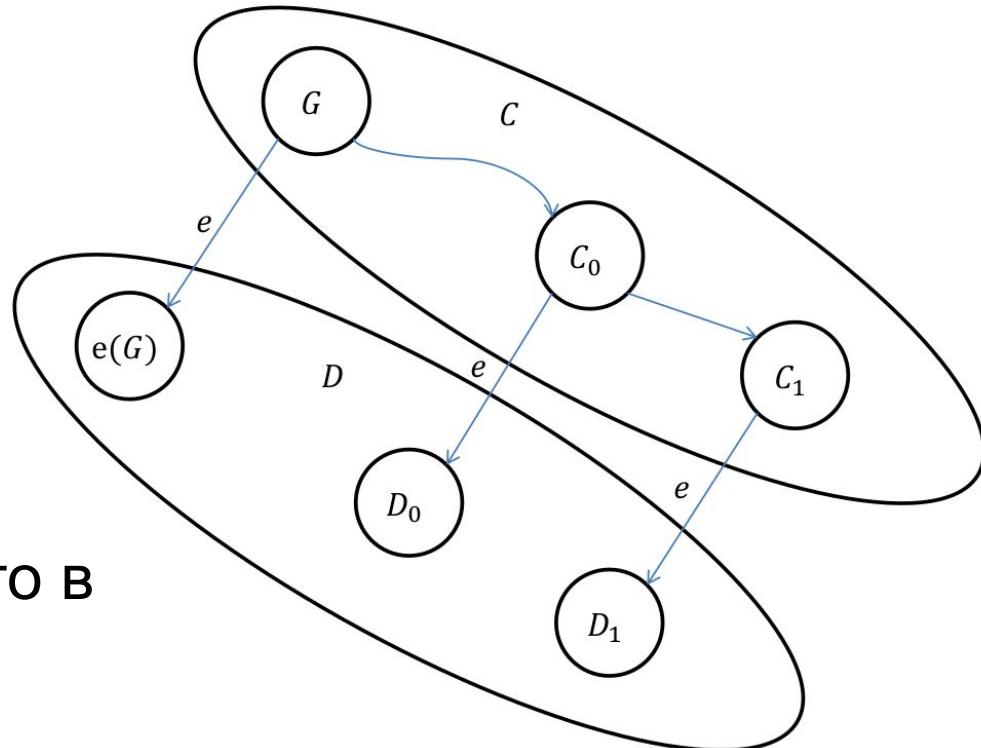
FLP: Доказательство

- Для любой бивалентной конфигурации можно найти следующую за ней бивалентную
 - Управляя порядком доставки сообщений, можно вечно держать систему в бивалентном состоянии
 - Не давая алгоритму завершаться



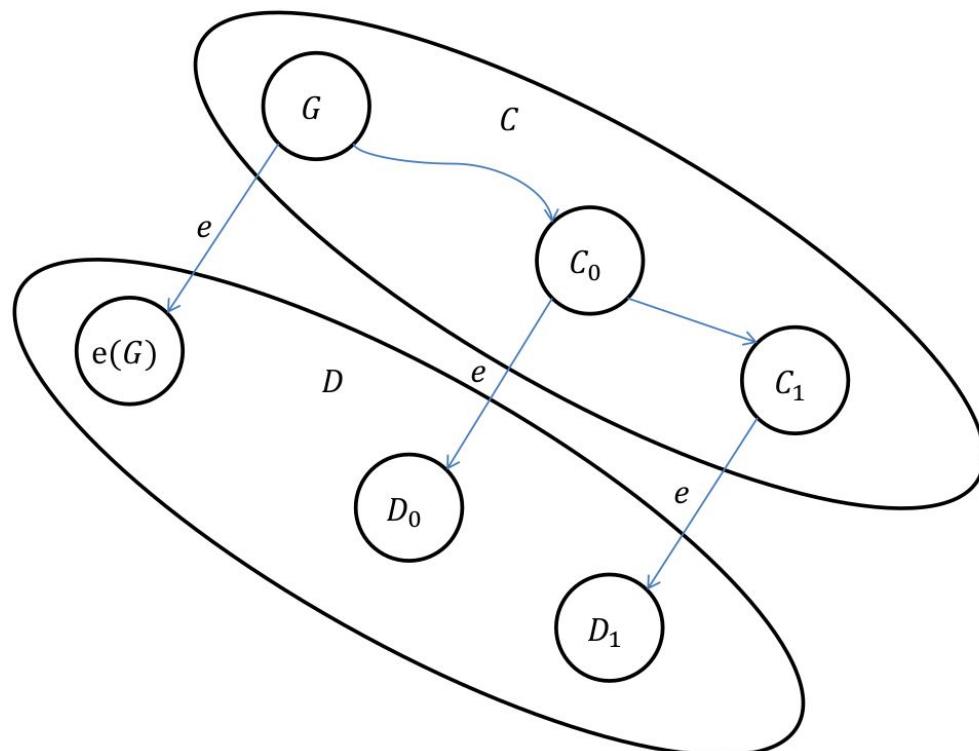
FLP: Доказательство

- Пусть G — бивалентная конфигурация, e — произвольное сообщение
- C — множество конфигураций, достижимых из G без обработки e
- $D = \{e(F) \mid F \in C\}$
- Докажем от противного, что в D есть бивалентная конфигурация



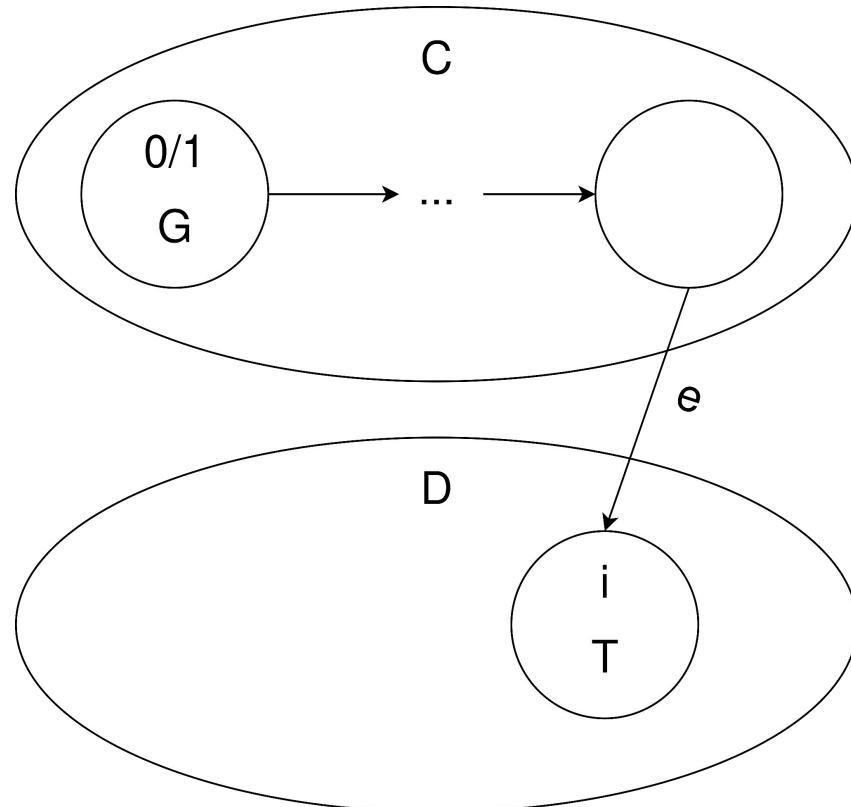
FLP: Доказательство

- Предположим, что в D нет бивалентных конфигураций
- Докажем, что $\forall i \in \{0, 1\}$ в D есть i -валентная конфигурация
- G — бивалентная
- От G можно дойти до i -валентной конфигурации T



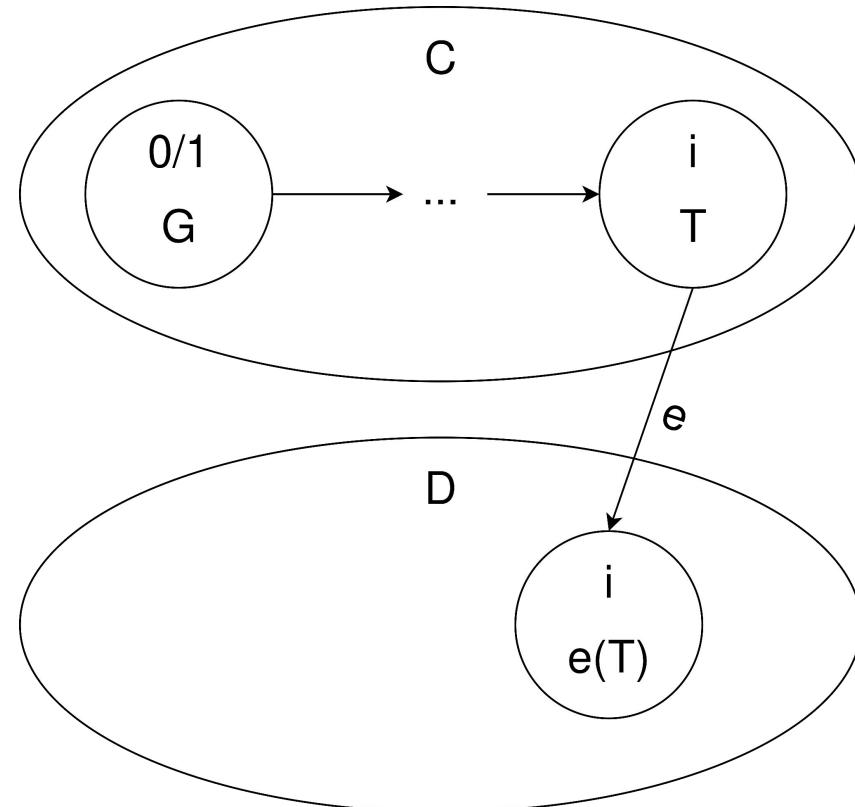
FLP: Существование i -валентной конфигурации

- Докажем, что $\forall i \in \{0, 1\}$ в D есть i -валентная конфигурация
- G — бивалентная
- От G можно дойти до i -валентной конфигурации T
- Пусть $T \in D$
 - Тривиально



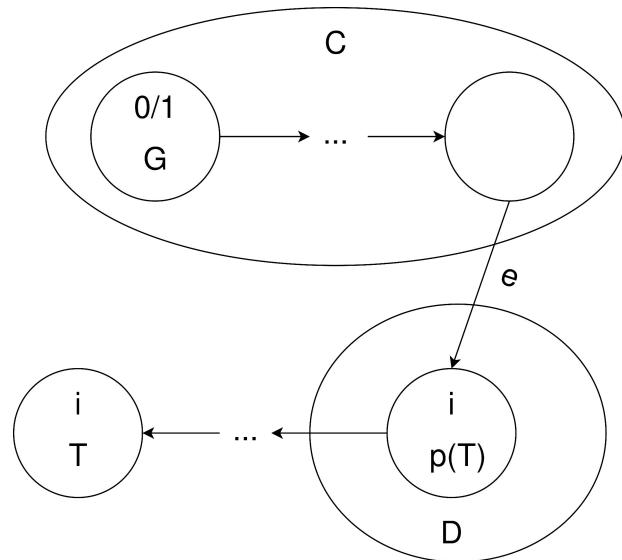
FLP: Существование i -валентной конфигурации

- Докажем, что $\forall i \in \{0, 1\}$ в D есть i -валентная конфигурация
- G — бивалентная
- От G можно дойти до i -валентной конфигурации T
- Пусть $T \in C$
- От G до T можно дойти не обрабатывая e
- Тогда $e(T) \in D$
- $e(T)$ i -валентная
 - Так как T i -валентная



FLP: Существование i -валентной конфигурации

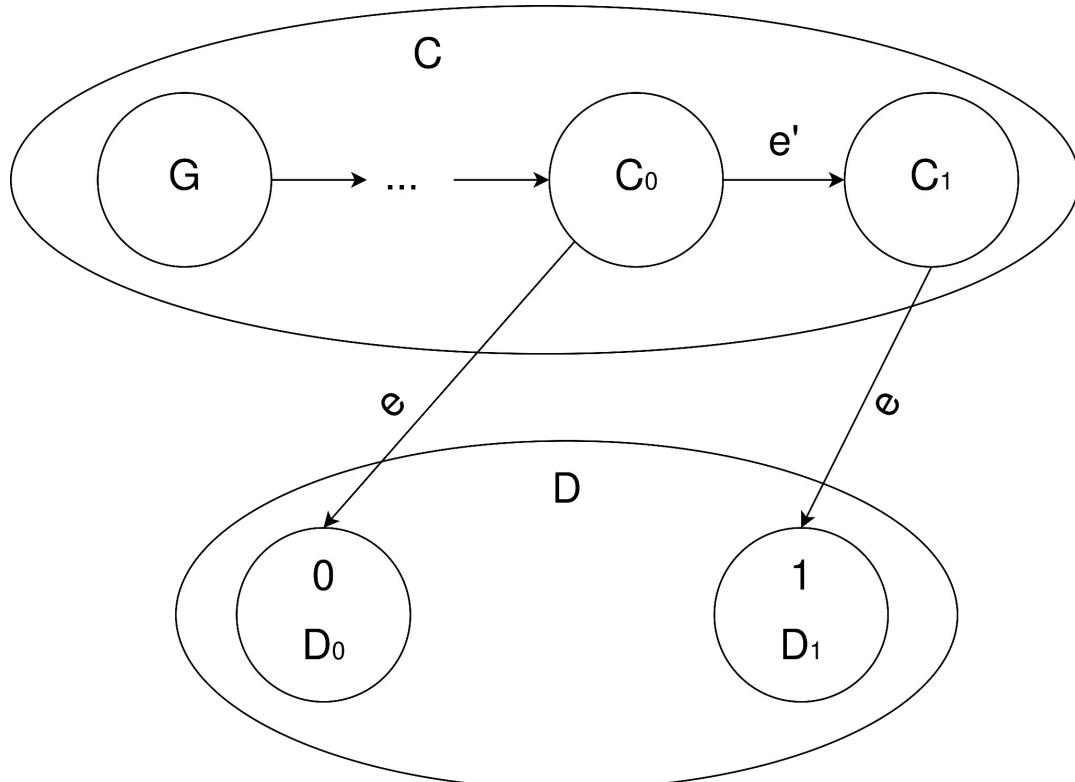
- От G можно дойти до i -валентной конфигурации T
- Пусть T не лежит ни в C , ни в D
- Раз T не лежит в C , на пути от G до T встречается ребро e
- Значит, этот путь проходит через D
- Обозначим $p(T)$ конфигурацию из D на пути от G до T
- В D нет бивалентных конфигураций
- Значит, $p(T)$ либо i -валентная, либо $1-i$ валентная
- От $p(T)$ можно дойти до i -валентной T
- Значит, $p(T)$ тоже i -валентная



FLP: Доказательство

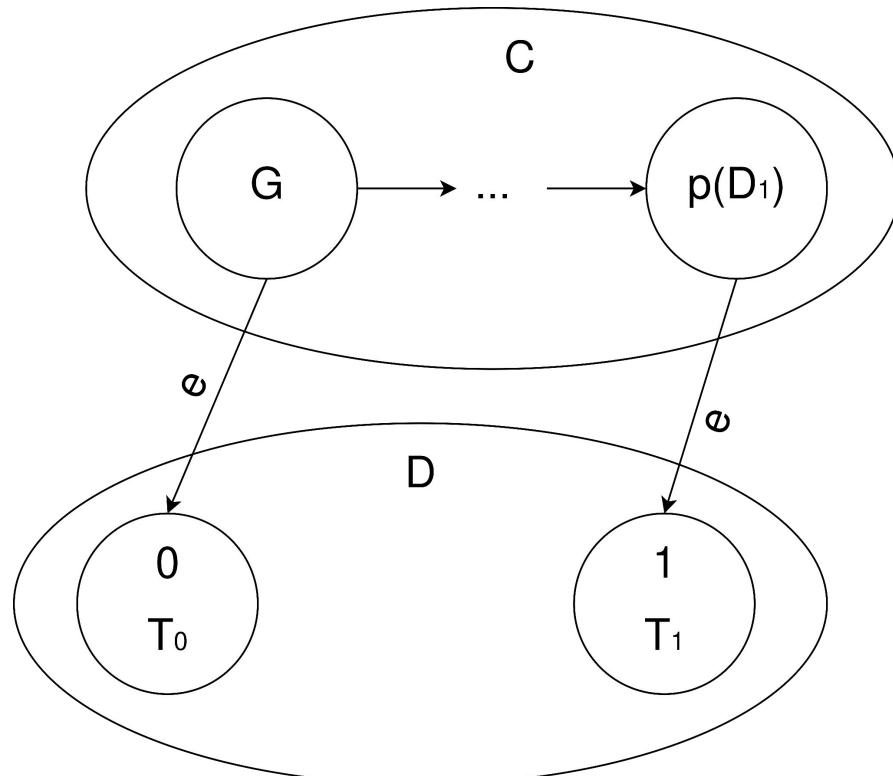
- Найдём такие $C_0, C_1 \in C$, что:

- C_0 отличается от C_1 одним шагом e'
- $D_0 = e(C_0) \in D$ — 0-валентная конфигурация
- $D_1 = e(C_1) \in D$ — 1-валентная конфигурация



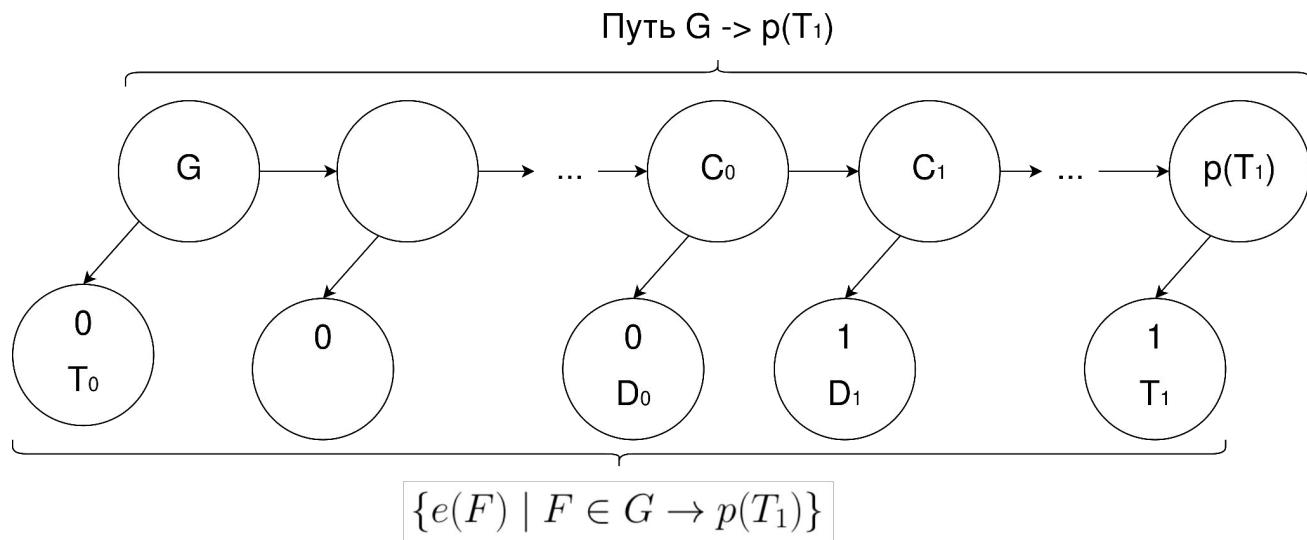
FLP: Поиск C_0, C_1, D_0, D_1

- Не умаляя общности, предположим, что $e(G)$ — 0-валентная конфигурация
- Найдём в D 1-валентную конфигурацию T_1
- Найдём предка T_1 в C
- Обозначим его $p(T_1)$
- От G до $p(T_1)$ существует путь в C
- На этом пути нет события e



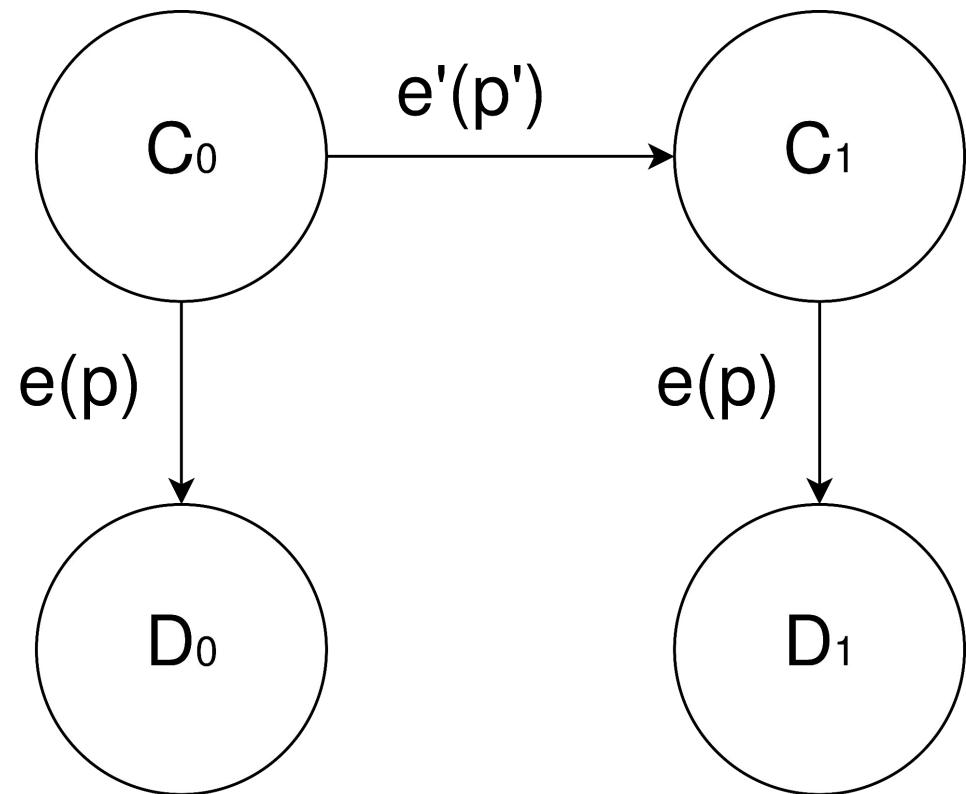
FLP: Поиск C_0, C_1, D_0, D_1

- Рассмотрим множество $\{e(F) \mid F \in G \rightarrow p(T_1)\}$
- $e(G)$ — 0-валентная конфигурация
- $e(p(T_1))$ — 1-валентная конфигурация
- Где-то на пути происходит переход с 0 на 1



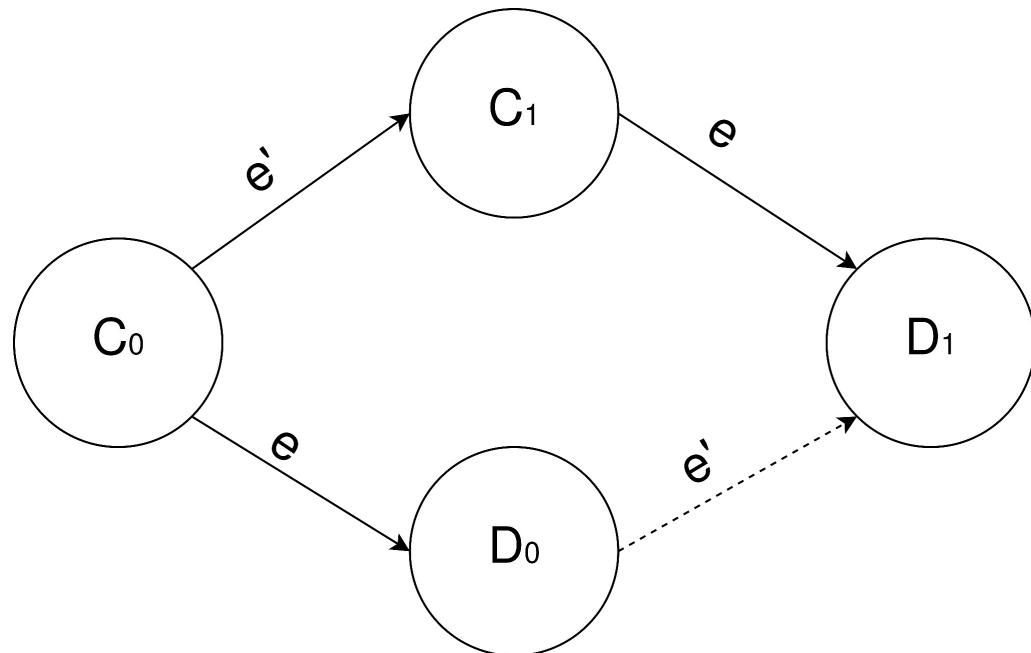
FLP: Доказательство

- C_0 и C_1 отличаются событием e' на процессе p'
- Не умоляя общности, представим что $C_1 = e'(C_0)$
- Но может быть и наоборот
- $D_0 = e(C_0)$ — 0-валентная конфигурация
- $D_1 = e(C_1)$ — 1-валентная конфигурация
- e произошло на процессе p



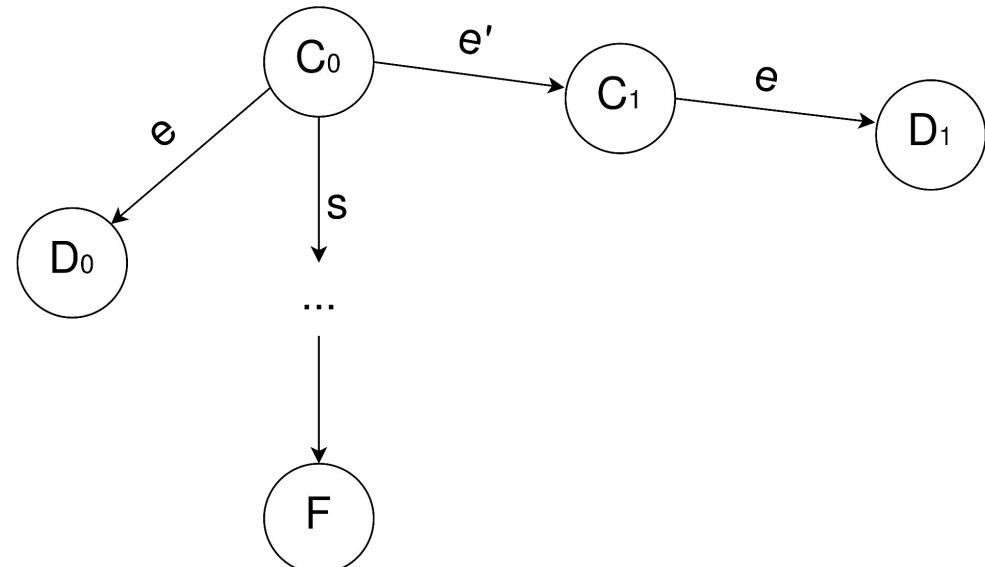
FLP: Доказательство

- Пусть e и e' произошли на разных процессах
 - Значит, они коммутируют
- Не важно, в каком порядке их применять — в итоге получим D_1
- D_1 1-валентна по условию
- D_1 достижима из D_0
 - Значит, D_1 0-валентна
- Противоречие!



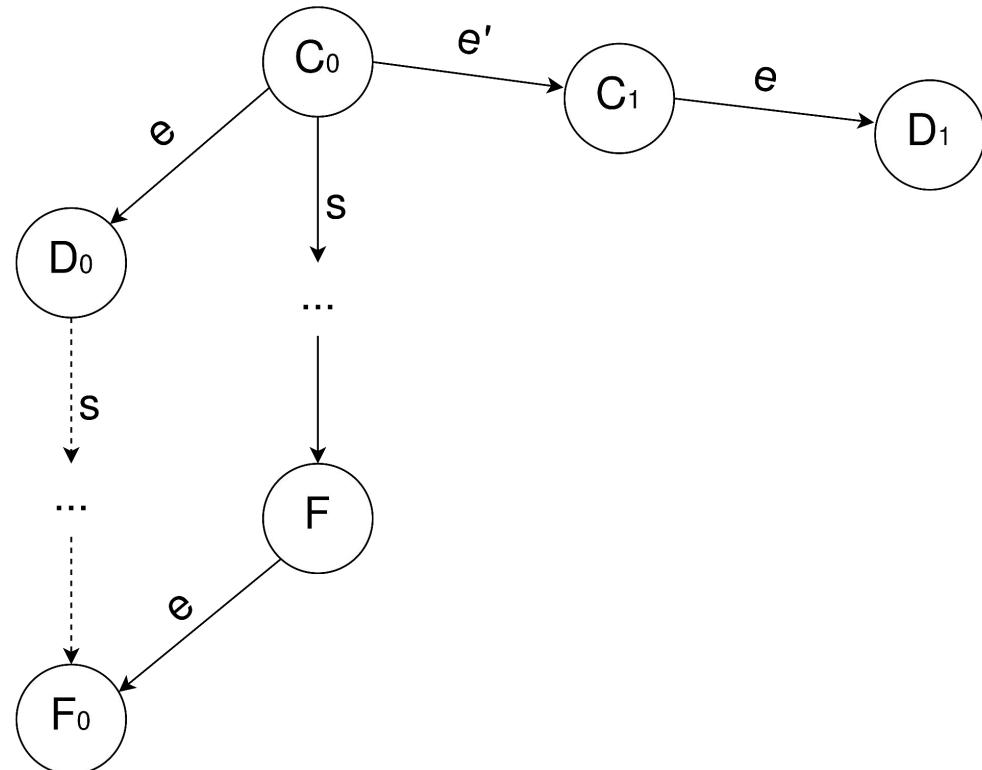
FLP: Доказательство

- Пусть e и e' произошли на одном процессе
 - Не коммутируют
- Представим, что этот процесс отказал в конфигурации C_0
- Остальные процессы пришли к консенсусу с помощью цепочки шагов s
- В s нет шагов отказавшего процесса
- F не может быть бивалентной



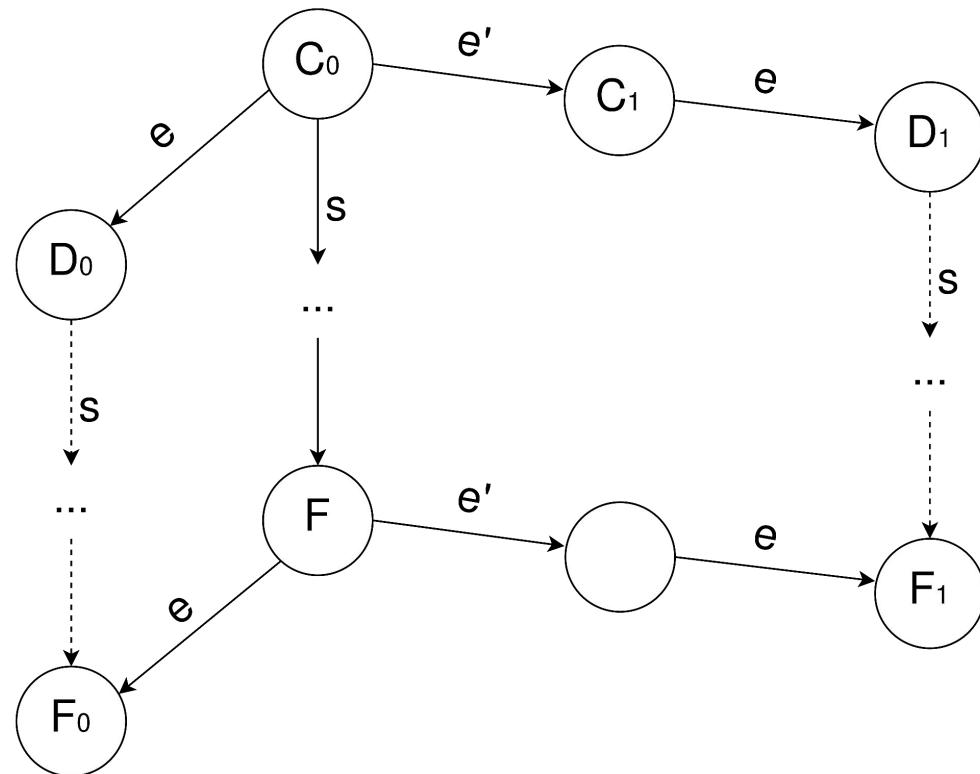
FLP: Доказательство

- Представим, что процесс r не отключился, а просто долго спал, пока остальные пришли к консенсусу
- После пробуждения должен тоже прийти к консенсусу
- s и e коммутируют
- F_0 — 0-валентная конфигурация
 - Достижима из C_0
- Значит, F — 0-валентная конфигурация



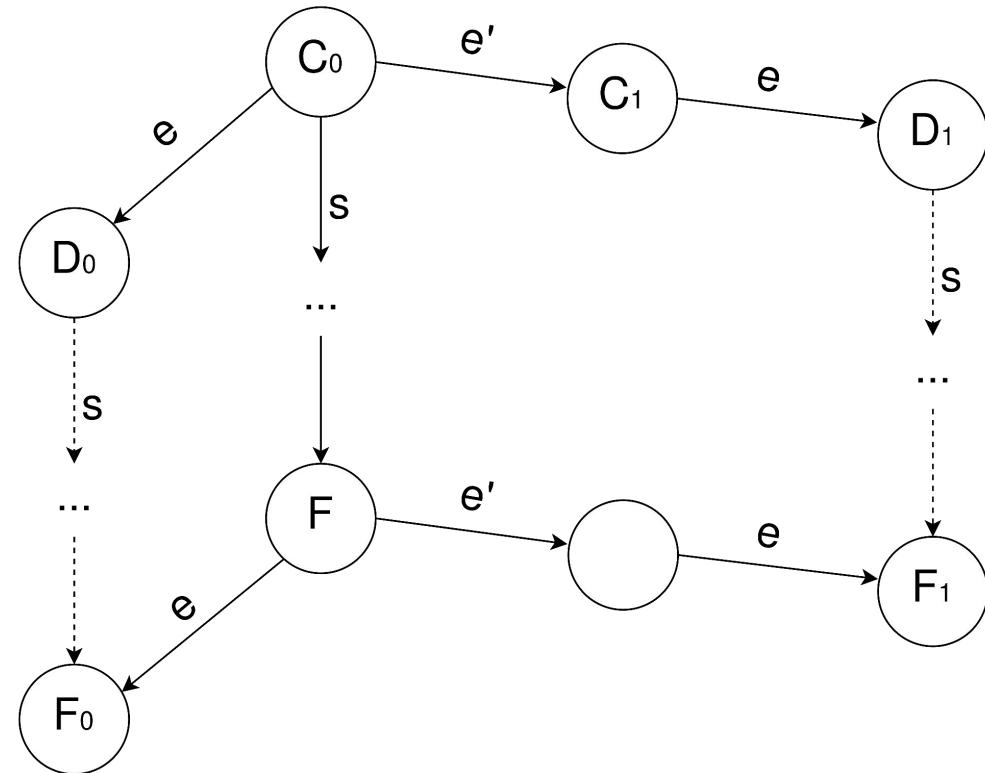
FLP: Доказательство

- Аналогично, s и (e', e) коммутируют
- F_1 — 1-валентная конфигурация
 - Достижима из D_1
- F — 1-валентная конфигурация
- Противоречие



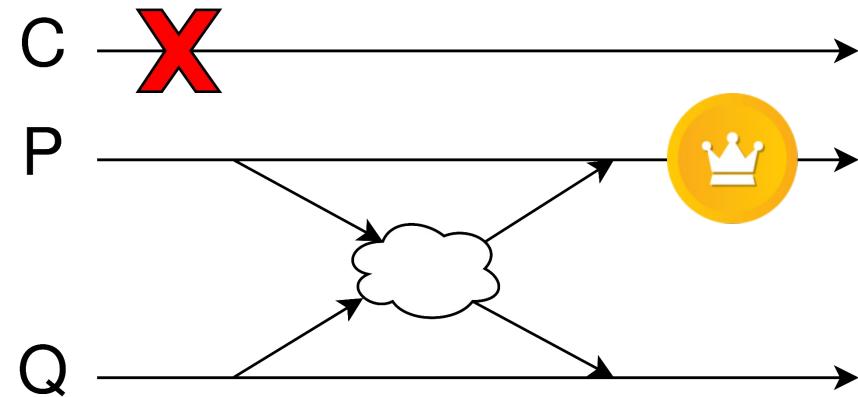
FLP: Доказательство

- Невозможно понять, отказал процесс r или просто задержка доставки большая
- Вынуждены приходить к консенсусу без него (F)
- Но действия этого процесса могут детерминировано привести как к результату 0 (D_0), так и 1 (D_1)



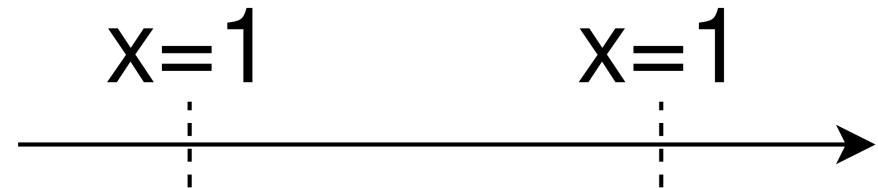
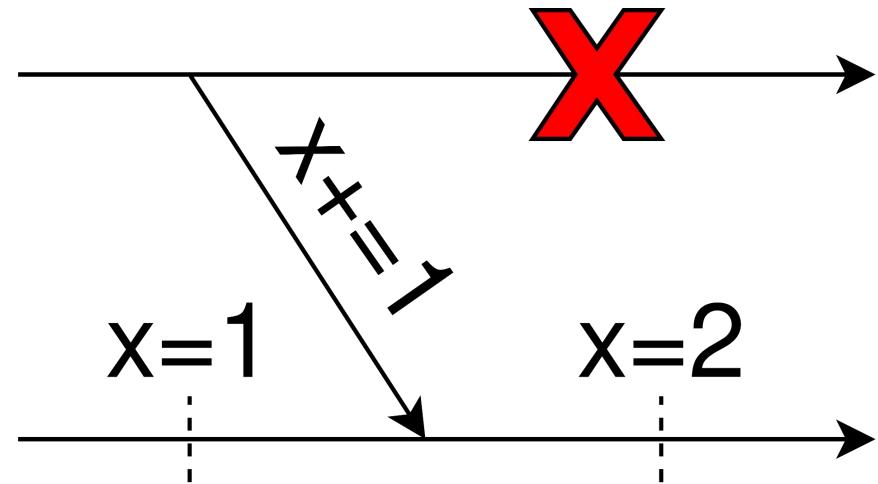
Эквивалентные задачи: выбор лидера

- Консенсус => выбор лидера
 - Каждый предлагает себя в качестве лидера
 - Приходим к консенсусу
- Выбор лидера => консенсус
 - Выбираем лидера
 - Каждый процесс посыпает ему своё предложение
 - Лидер выбирает одно предложение
 - Рассыпает его всем живым



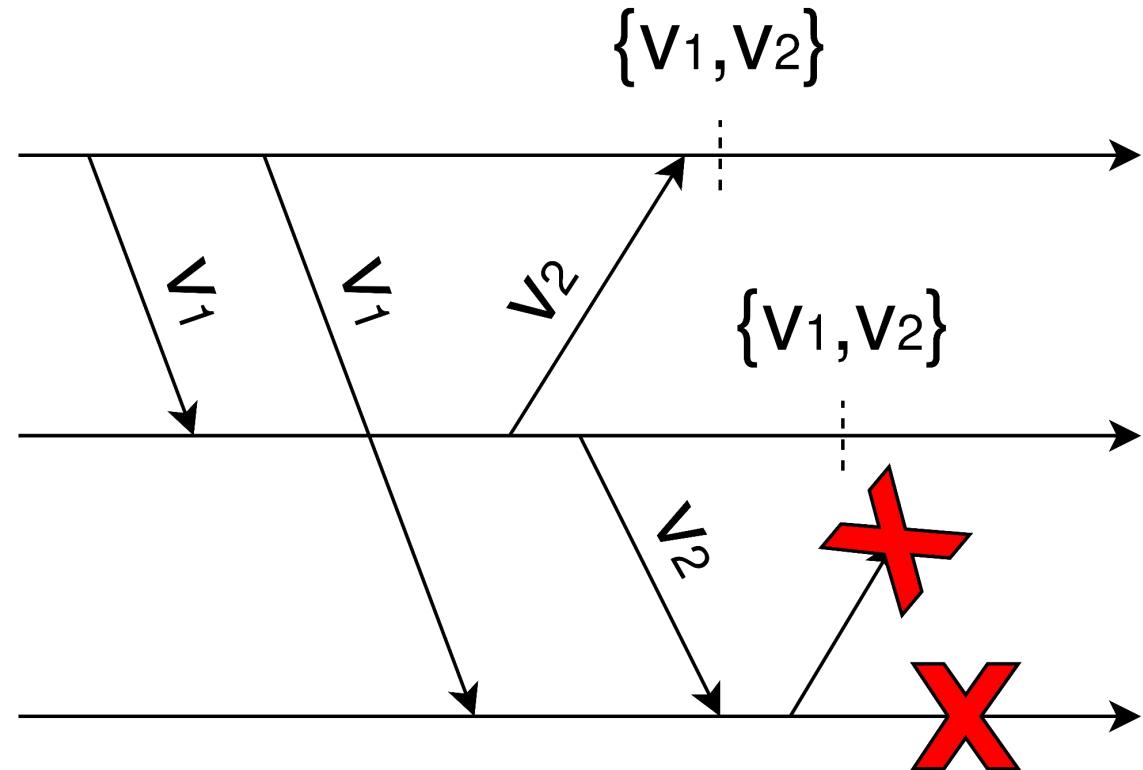
Terminating Reliable Broadcast

- Сообщение обрабатывается либо всеми, либо никем
- Иначе состояния могут разойтись



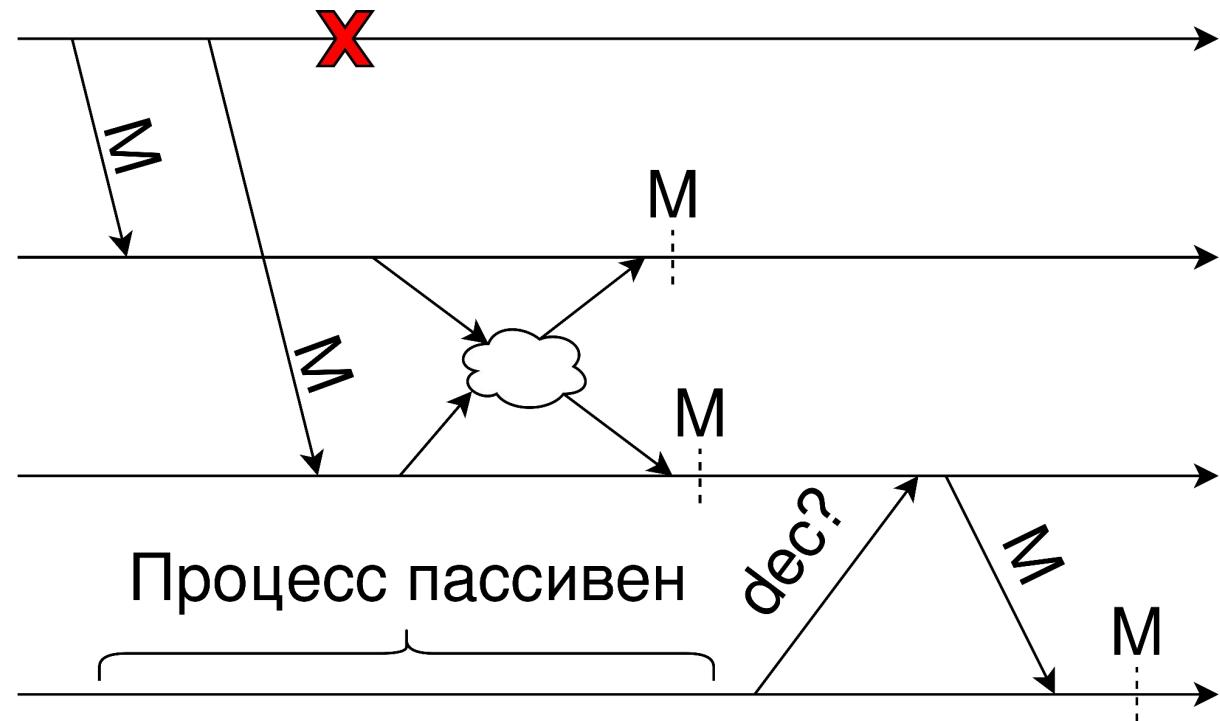
TRB => консенсус

- Каждый процесс с помощью TRB рассыпает своё предложение
- Каждое предложение принято либо всеми, либо никем
- Множество предложений одинаковое у всех живых процессов



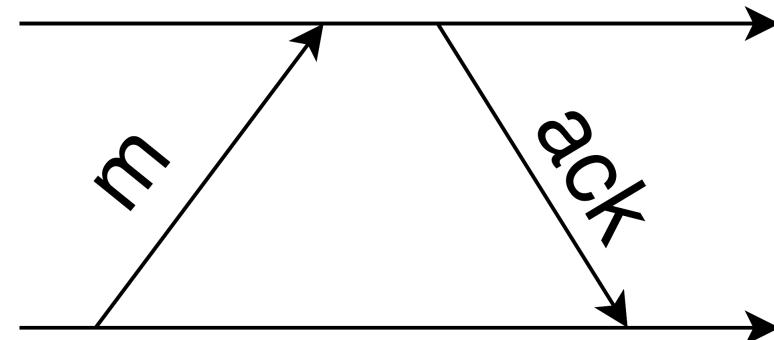
Консенсус => TRB

- Процесс рассыпает сообщение
- Получатели приходят к консенсусу, обрабатывать ли его
- Опоздавшие процессы могут узнать решение позже
- И обработать сообщение, если нужно



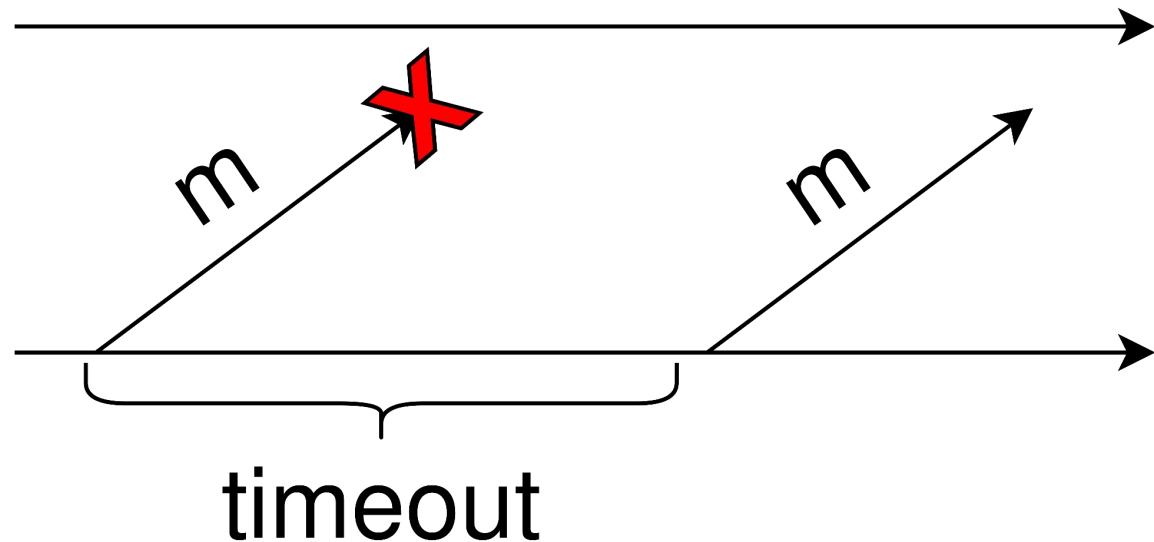
Надёжная доставка

- Имеем канал с ненадёжной доставкой
- Хотим гарантированно доставить сообщение m от процесса P к процессу Q
- Можем это сделать если оба процесса живы
- И $P_{\text{delivery}} > 0$
- Требуем подтверждения получения сообщения
- Подтверждение получено \Rightarrow сообщение точно доставлено



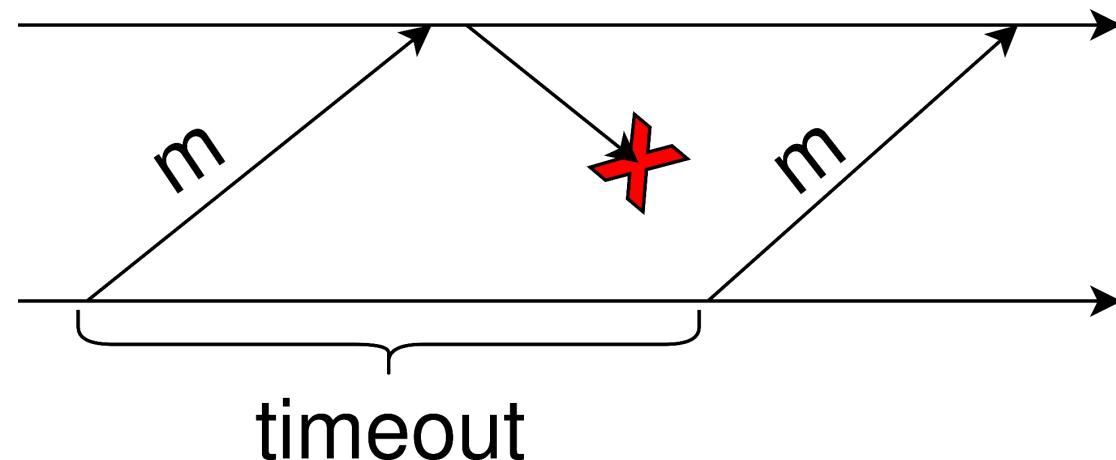
Надёжная доставка

- Если не получили подтверждения за определённое время — посылаем сообщение ещё раз
- Посылаем пока не получим подтверждение



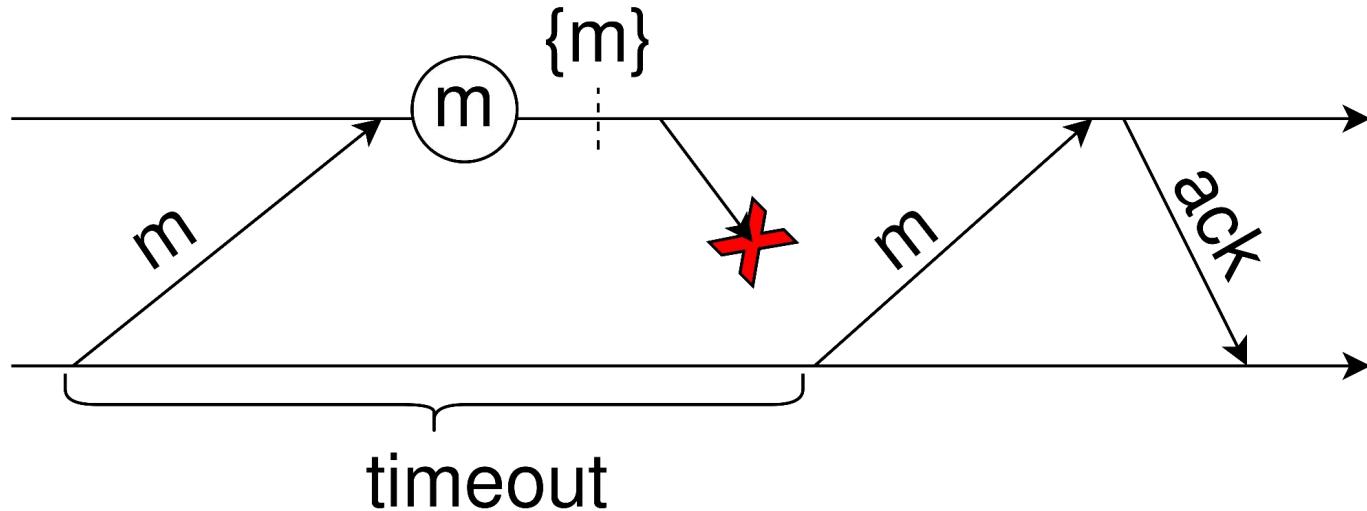
Надёжная доставка: At Least Once Delivery

- Потеряться могло не сообщение, а подтверждение
- Послали сообщение ещё раз
- Получатель обработал сообщение повторно
- Получатель обрабатывает сообщение хотя бы однажды
- Так как мы
прекращаем
посылать только
получив
подтверждение



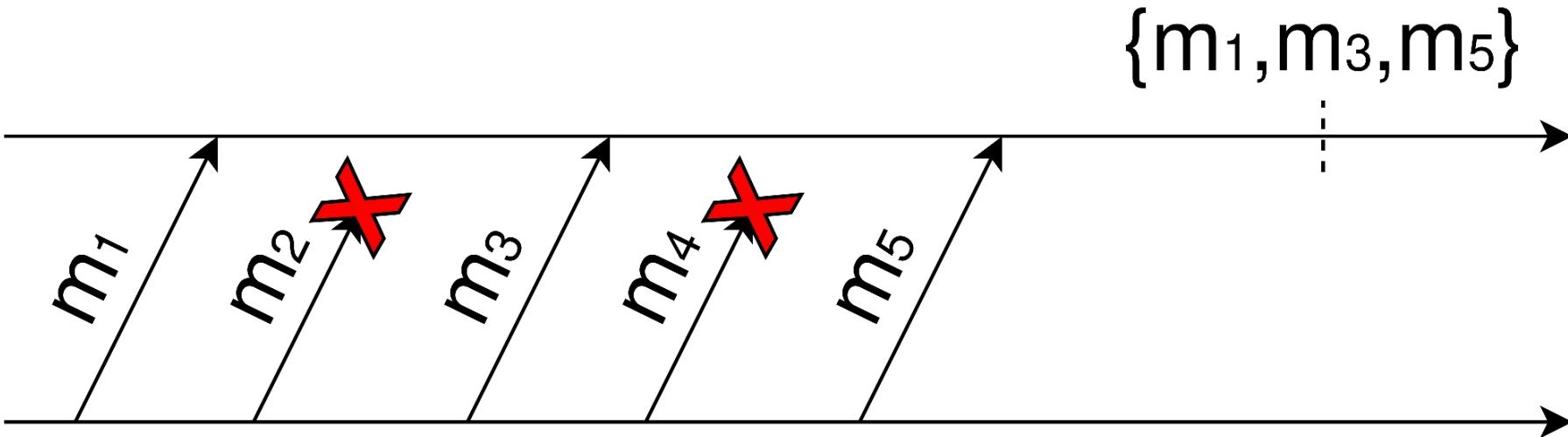
Надёжная доставка: Дедупликация

- Получатель запоминает сообщение при обработке
- Получив его второй раз, не обрабатывает
 - Просто отвечает подтверждением
 - Понимает, что предыдущее подтверждение не дошло



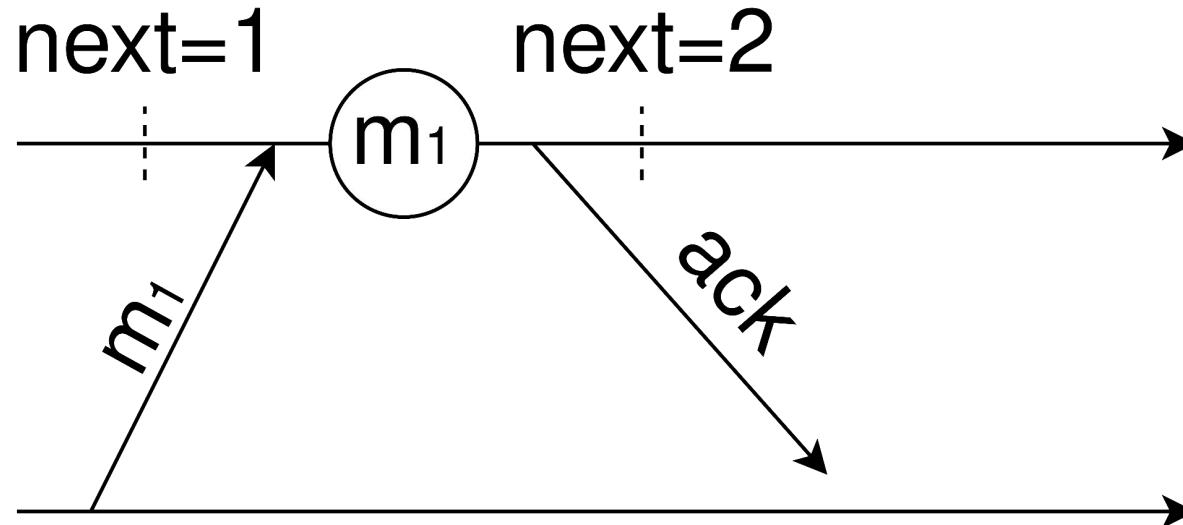
Надёжная доставка: Дедупликация

- Почти всегда отправитель хочет доставить не одно сообщение, а несколько
- Нужно хранить полученные сообщения в множестве
- Размер множества всё время увеличивается



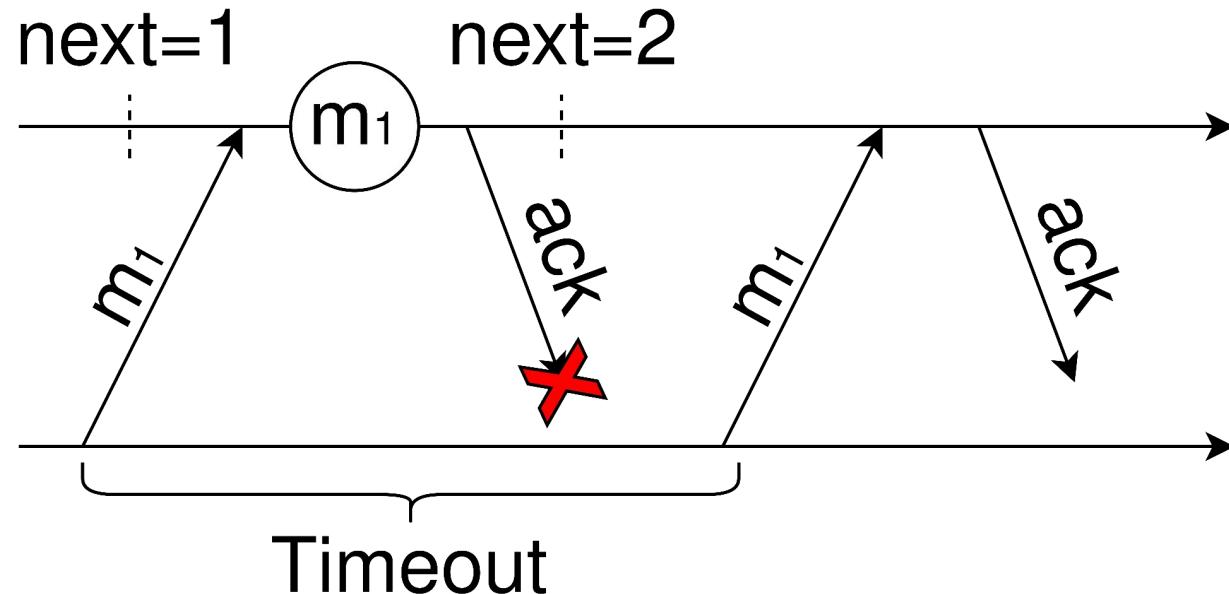
Надёжная доставка: FIFO

- Получатель поддерживает одно число: номер ожидаемого сообщения
- Получив ожидаемое сообщение, обрабатывает его, посыпает подтверждение и увеличивает номер



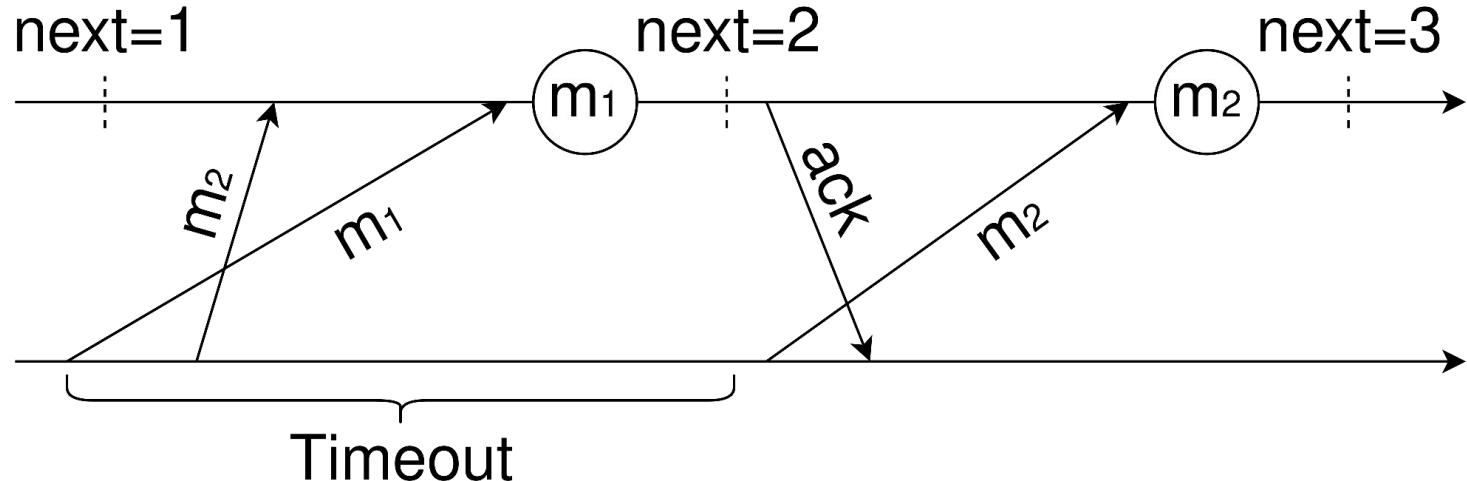
Надёжная доставка: FIFO

- Получив уже обработанное сообщение ещё раз, отправляем подтверждение и ничего не делаем
- Понимаем, что предыдущее подтверждение потерялось



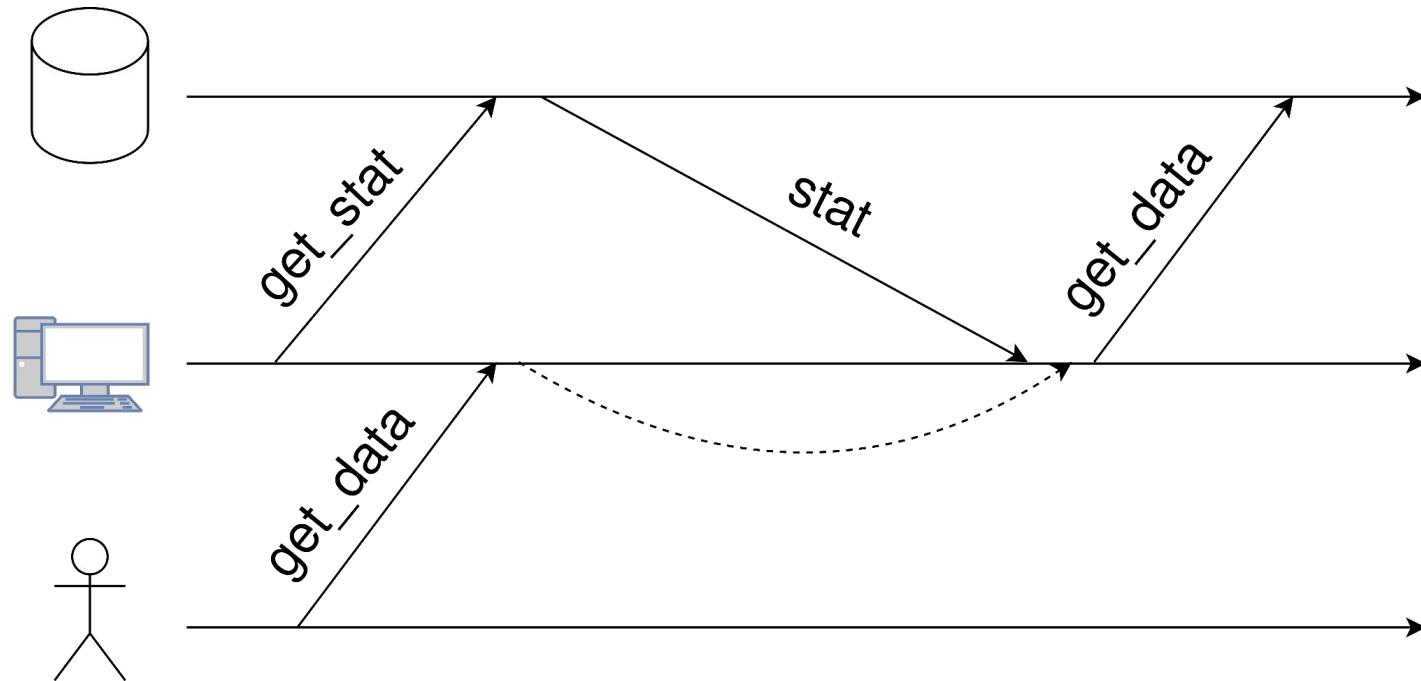
Надёжная доставка: FIFO

- Получив сообщение “из будущего” ничего не делаем
- Отправитель не получит подтверждения и отправит ещё раз
- А мы обработаем в будущем
- Примерно так работает TCP



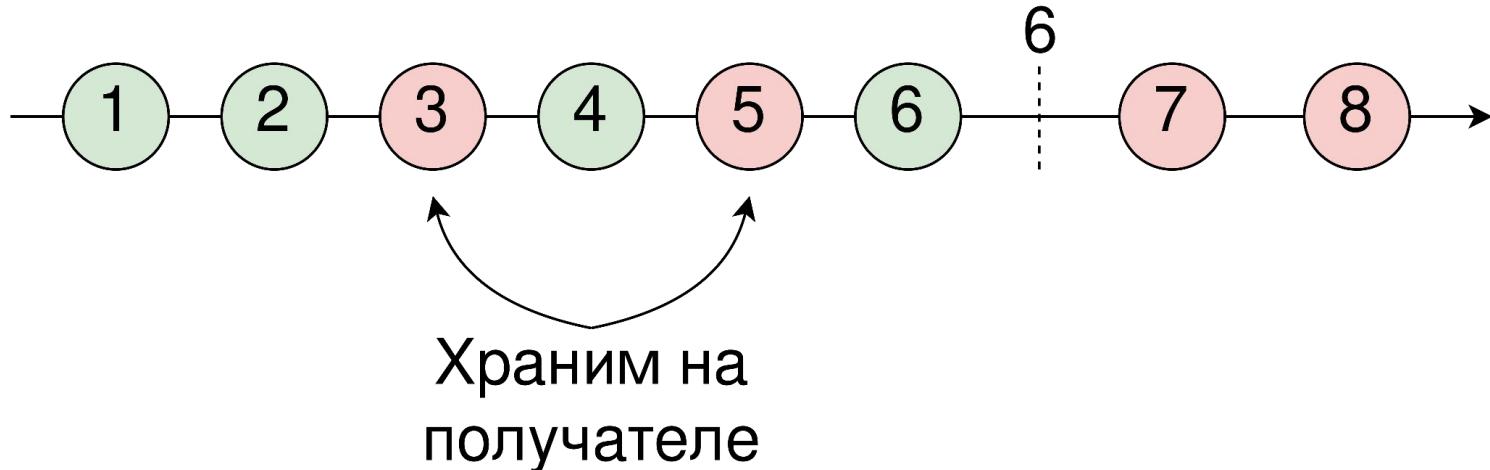
Проблемы FIFO

- Head-Of-Line Blocking
- Не можем отправить высокоприоритетный запрос пока не получен ответ на низкоприоритетный



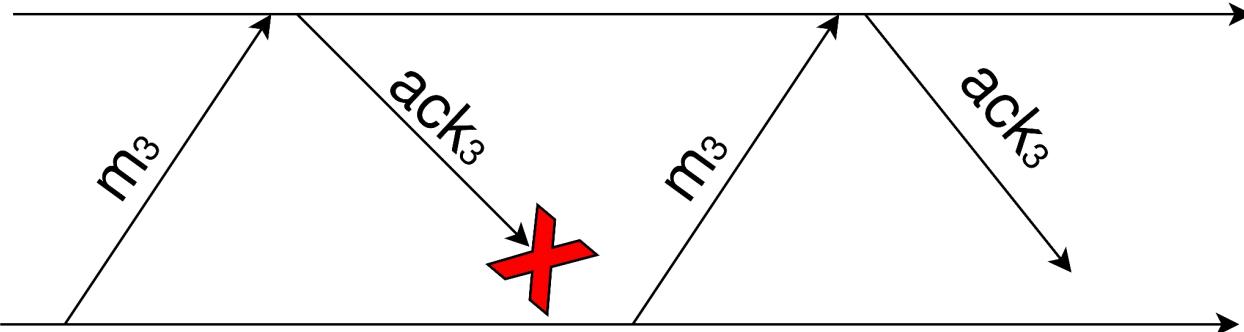
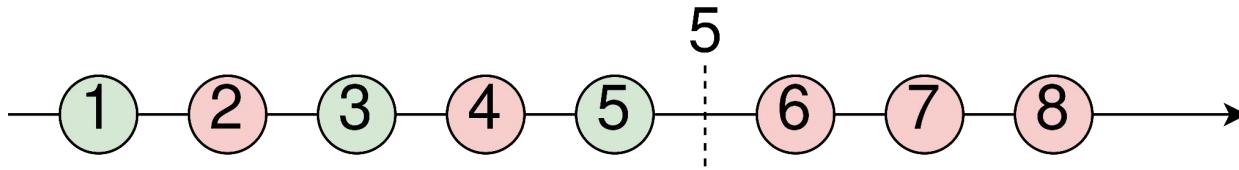
Асинхронная надёжная доставка

- На стороне получателя храним максимальный номер полученного сообщения
- Все сообщения с большим номером точно не получены
- И множество номеров не полученных сообщений
 - Номер которых меньше максимального полученного



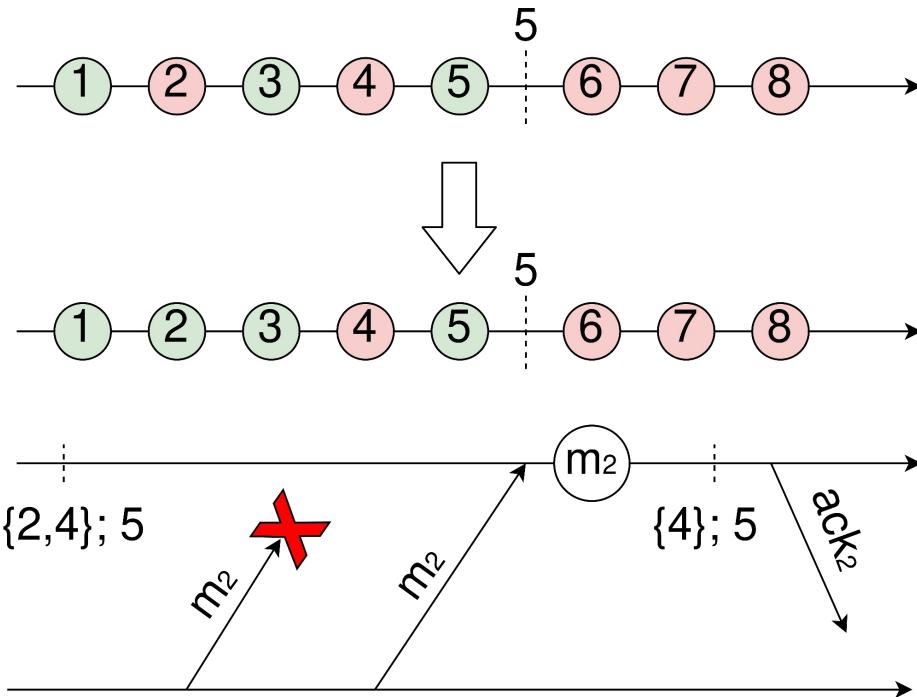
Асинхронная надёжная доставка

- Получили обработанное сообщение
 - $\text{num} \leq \text{max_received_num}$
 - $\text{num} \notin \text{not_received}$
- Посыпаем подтверждение
- Ничего не делаем



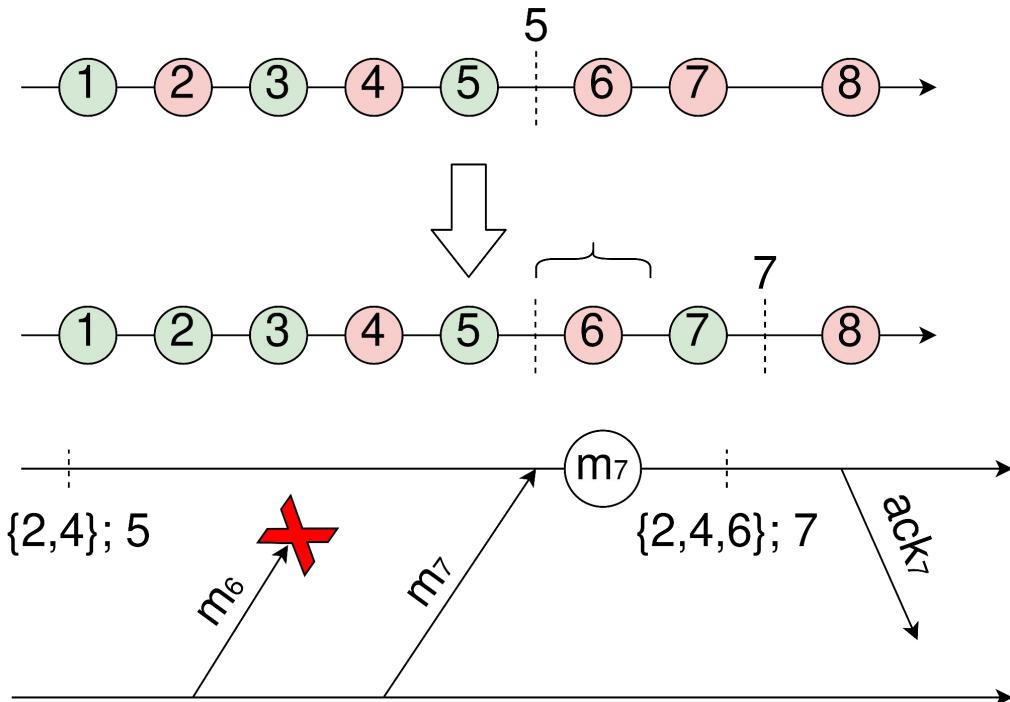
Асинхронная надёжная доставка

- Получили сообщение из множества необработанных
 - $\text{num} \leq \text{max_received_num}$
 - $\text{num} \in \text{not_received}$
- Убираем номер из множества
- Обрабатываем сообщение
- Размер множества не растёт бесконечно



Асинхронная надёжная доставка

- Получили сообщение из множества необработанных
 - $\text{num} > \text{max_received_num}$
- Обрабатываем
- Высылаем подтверждение
- Сдвигаем границу вправо
- Добавляем в множество все номера от max_received_num до $\text{num} - 1$



Что почитать:

- *Fischer M. J., Lynch N. A., Paterson M. S.* Impossibility of distributed consensus with one faulty process
- *Lynch N. A.* Distributed algorithms
- *Fischer M. J.* The consensus problem in unreliable distributed systems (a brief survey)
- *Kevin Sookocheff.* How Does TCP Work
 - sookocheff.com/post/networking/how-does-tcp-work

Thanks for your attention

