Программирование распределенных систем

Роман Елизаров, 2021

elizarov@gmail.com

Лекция 4

ИЕРАРХИЯ ОШИБОК / ОТКАЗОВ

Отказы в распределенных системах

- Делают программирование надежных распределенных систем действительно сложным
- **Частичные отказы**
  - Отказ частей системы
  - Остальные части должны продолжать работать
- Работа условии отказов необходимость, а не роскошь
  - Отказ это норма, а не исключение

Иерархия отказов в распределенных системах

- **Иерархия в порядке усложнения**
  - Более сложный отказ может моделировать более простой
  - Умеем работать при сложном ⇒ сможем при простом

Иерархия отказов в распределенных системах

Синхронные и асинхронные системы

- **Синхронные системы**
    - Время передачи сообщений ограничено сверху
    - Можно разбить выполнение алгоритма на фазы
- **Асинхронные системы**
    - Время передачи сообщений не ограничено сверху
    - Но время передачи конечно если нет отказов

Консенсус в распределенной системе

- **Постановка задачи**
  - Каждый процесс имеет свое предложение (**proposal**)
    - Может быть один бит, число, или что-либо другое
  - Все процессы должны прийти к решению (**decision**)
  - Приход к решению это некий распределенный алгоритм
    - Алгоритм **Консенсуса**

Консенсус в распределенной системе

- **Согласие** (agreement)
  - Все (не отказавшие) процессы должны завершиться с одним и тем же решением
- **Нетривиальность** (non-triviality)
  - Должны быть варианты исполнения приводящие к разным решениям
  - Более сильное требование называется **обоснованность** – решение должно быть предложением одного из процессов
- **Завершение** (termination)
  - Протокол должен завершиться за конечное время

Консенсус без отказов – легко!

- Каждый процесс шлет свое **предложение** всем остальным
- Дожидается N-1 предложений от других процессов
- Теперь из N предложений (свое и остальных) выбираем **решение**
  - Используя любую детерминированную функцию над множеством предложений, например, минимум из всех предложений
  - Любая нетривиальная детерминированная функция подойдет!

Консенсус без отказов – легко!

- **Алгоритм работает и в асинхронной системе**
  - Не важно как долго идут сообщения
  - Главное чтобы не было отказов

Невозможность консенсуса в асинхронной системе с отказом узла (FLP)

- Результат Фишера-Линча-Патерсона (FLP), 1985 год
- **Важные предпосылки**
  - Система асинхронна (нет предела времени доставки сообщения)
  - (Один) узел может отказать
  - Консенсус надо достичь за конечное время
- **ТЕОРЕМА:** Невозможно достичь консенсуса N процессам
  - даже на множестве значений из двух элементов 0 и 1
- Доказательство от противного
  - Предположим что такой алгоритм существует и проанализируем возможные варианты его исполнения

FLP: Модель системы

- Нужна четко формализованная модель, так как будем доказывать невозможность.

FLP: Модель системы: Процесс

- **Процесс** это некий *детерминированный* автомат, который может делать
  - **receive():msg** чтобы ожидать получение сообщения
    - Нет возможности указать «время ожидания» (!)
  - **send(msg)** чтобы отправить сообщение
    - Отосланные сообщение не обязательно сразу обрабатываются
  - **decided(value)** когда принято решение
    - Решение процесс принимает один раз
    - Но процесс продолжает выполняться и может сообщать свое решение другим процессам

FLP: Модель системы: Конфигурация

- **Конфигурация** это
    - состояние всех процессов
    - все сообщения в пути (отправленные и не полученные)

FLP: Модель системы: Шаг

- **Шаг** от одной конфигурации до другой это
  - обработка какого-то сообщения процессом (*событие*)
  - внутренние действия этого процесса и посылка им от нуля до нескольких сообщений до тех пор, пока процесс не перейдет к ожиданию следующего сообщения.
    - **Детерминировано** (однозначно определяется) событием (!)

FLP: Модель системы: Начало

- **Начальная конфигурация** содержит начальные данные для каждого из процессов
  - Не обязательно один бит, а сколько угодно входных данных
    - Начальных конфигураций много (на каждый вариант входных данных)
  - И вообще каждый процесс может иметь свою программу

FLP: Модель системы: Исполнение

- **Исполнение** это бесконечная цепочка шагов от начального состояния
  - ибо процессы продолжают выполняться и после принятия решения

FLP: Модель системы: Отказ

- **Отказавший** процесс делает только конечное число шагов в процессе исполнения
  - И такой процесс от силы один
  - А каждый из остальных, не отказавших, процессов делает бесконечное число шагов

FLP: Модель системы: Надежная доставка

- Любое сообщение для не отказавшего процесса обрабатывается через конечное число шагов
  - Сообщения не теряются (!)

FLP: Согласие и решение

- Так как есть согласие, то все процессы пришедшие к решению имеют одно и то же решение (0 или 1)
  - Из-за возможности отказа одного процесса, даже если один процесс не делает шагов, то все остальные должны прийти к решению за конечное число шагов

FLP: Валентность

- Конфигурация называется
  - i-**валентной**, если все цепочки шагов из неё приводят к решению i (0-валентные и 1-валентные конфигурации)
  - **бивалентной**, если есть так цепочки шагов приводящие к решению 0, так и цепочки шагов приводящие к решению 1

FLP: Коммутирующие события

- Цепочки шагов с событиями на разных процессах коммутируют и приводят к одной и той же конфигурации если поменять их порядок выполнения

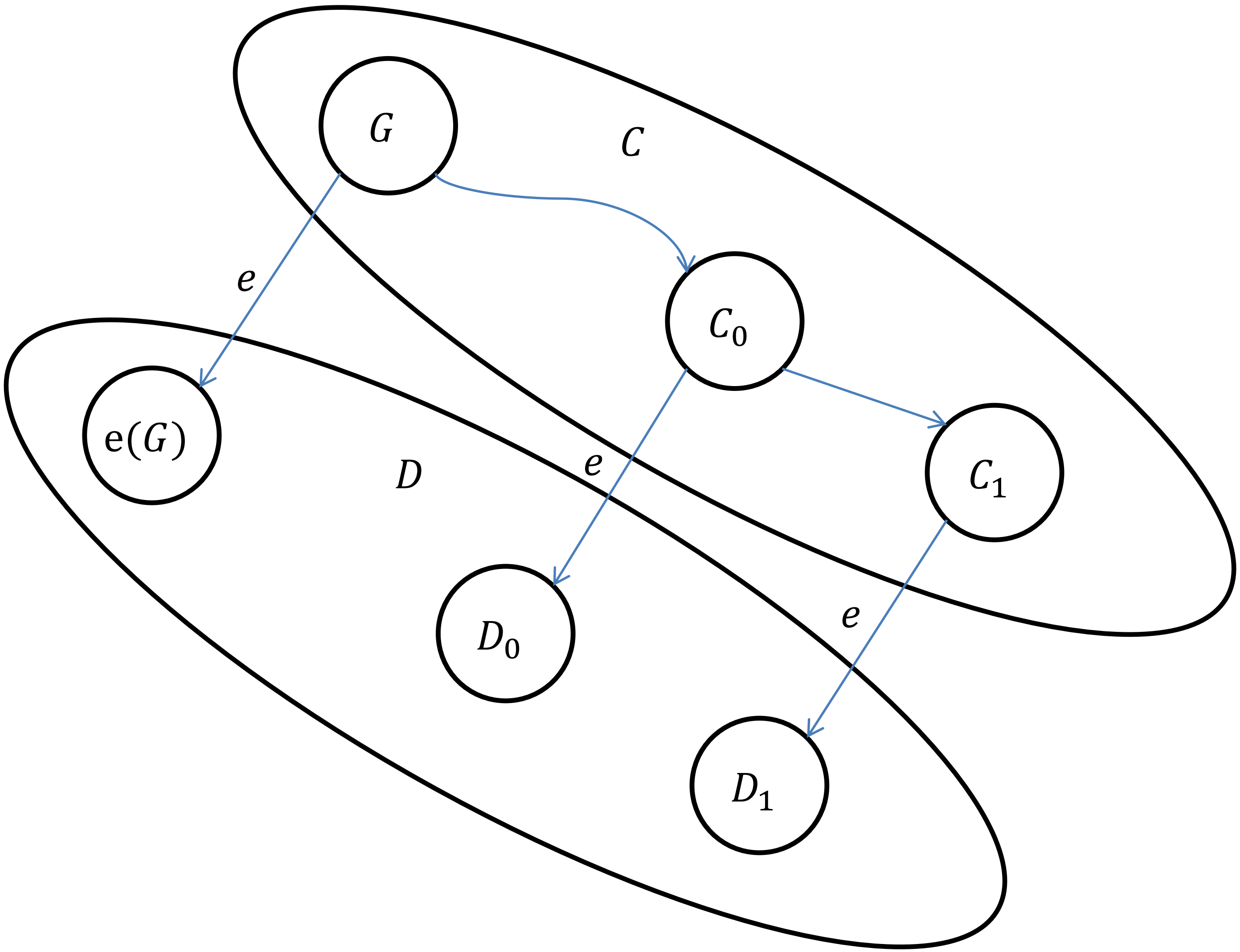
FLP Лемма 1: Существует (начальная) бивалентная конфигурация

- От противного. Если не существует (начальной) бивалентной конфигурации, значит все конфигурации одновалентны
  - Из нетривиальности есть как 0- так и 1- валентные
  - Значит найдем пару начальных конфигураций разной валентности, отличающихся начальным состоянием только одного процесса
  - Но этот процесс может отказать (не исполнятся) с самого начала, и тогда одна и та же цепочка шагов (других процессов) приводящая к решению возможна как в одной (0-валентной) конфигурации, так и в другой (1-валентной) конфигурации. Противоречие.

FLP Лемма 2: Для бивалентной конфигурации можно найти следующую за ней бивалентную

- Если G бивалентная конфигурация, и e это какое-то событие (процесс p и сообщение m) в этой конфигурации, то возьмем
  - C – множество конфигураций достижимых из G без e
  - D – множество конфигураций $D = e(C)$, то есть конфигураций где e это последнее событие

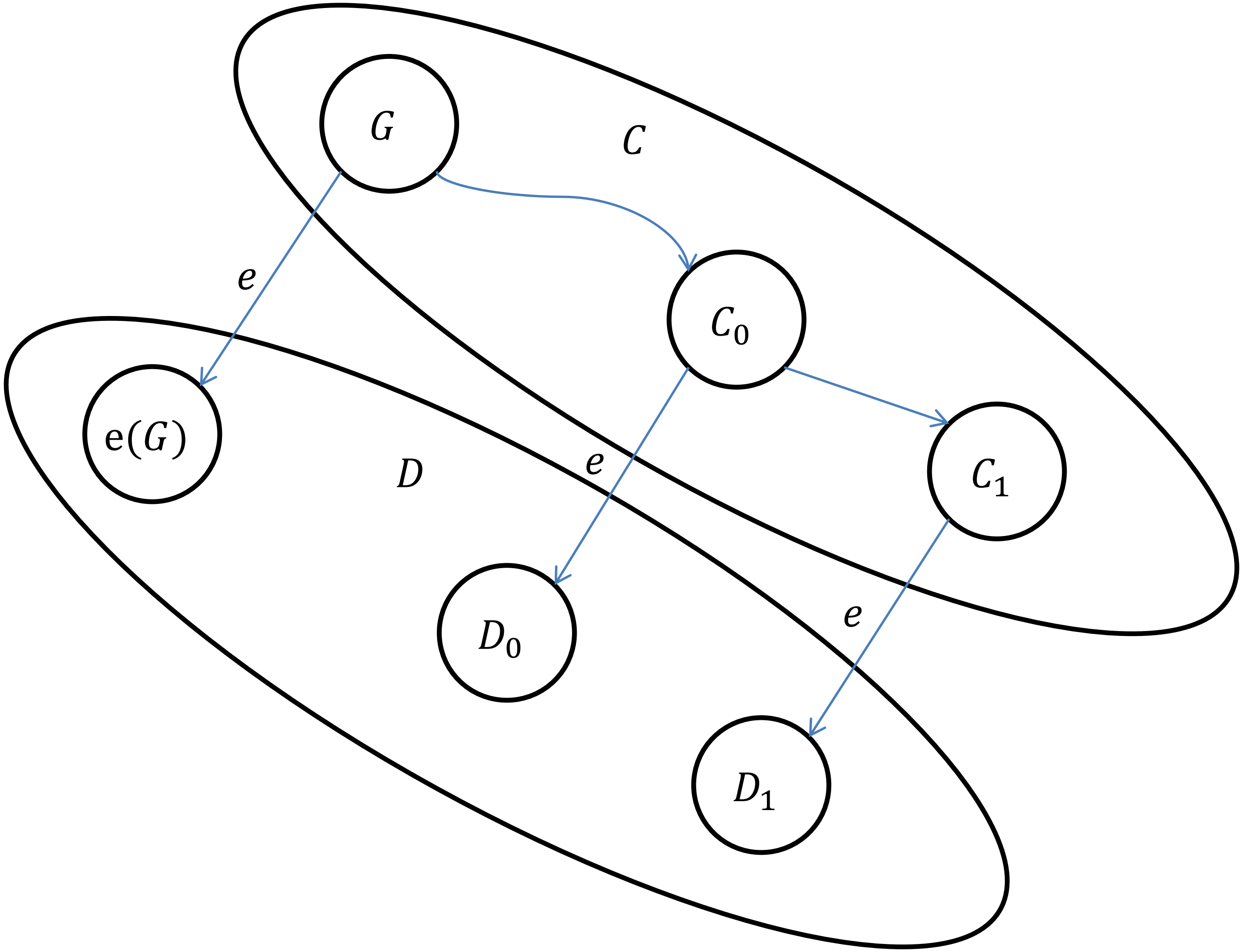
FLP Лемма 2: Иллюстрация



FLP Лемма 2: Для бивалентной конфигурации можно найти следующую за ней бивалентную

- Если G бивалентная конфигурация, и e это какое-то событие (процесс p и сообщение m) в этой конфигурации, то возьмем
  - C – множество конфигураций достижимых из G без e
  - D – множество конфигураций $D = e(C)$, то есть конфигураций где e это последнее событие
- Докажем что D содержит бивалентную конфигурацию
  - **Тем самым придем к противоречию с достижением консенсуса за конечное число шагов и докажем ТЕОРЕМУ**
  - По сути, мы воспользуемся асинхронностью: нет предела на время обработки сообщения, а значит любое сообщение можно отложить на любое конечное время
  - Докажем лемму 2 от противного. Предположим что D не содержит бивалентных конфигураций

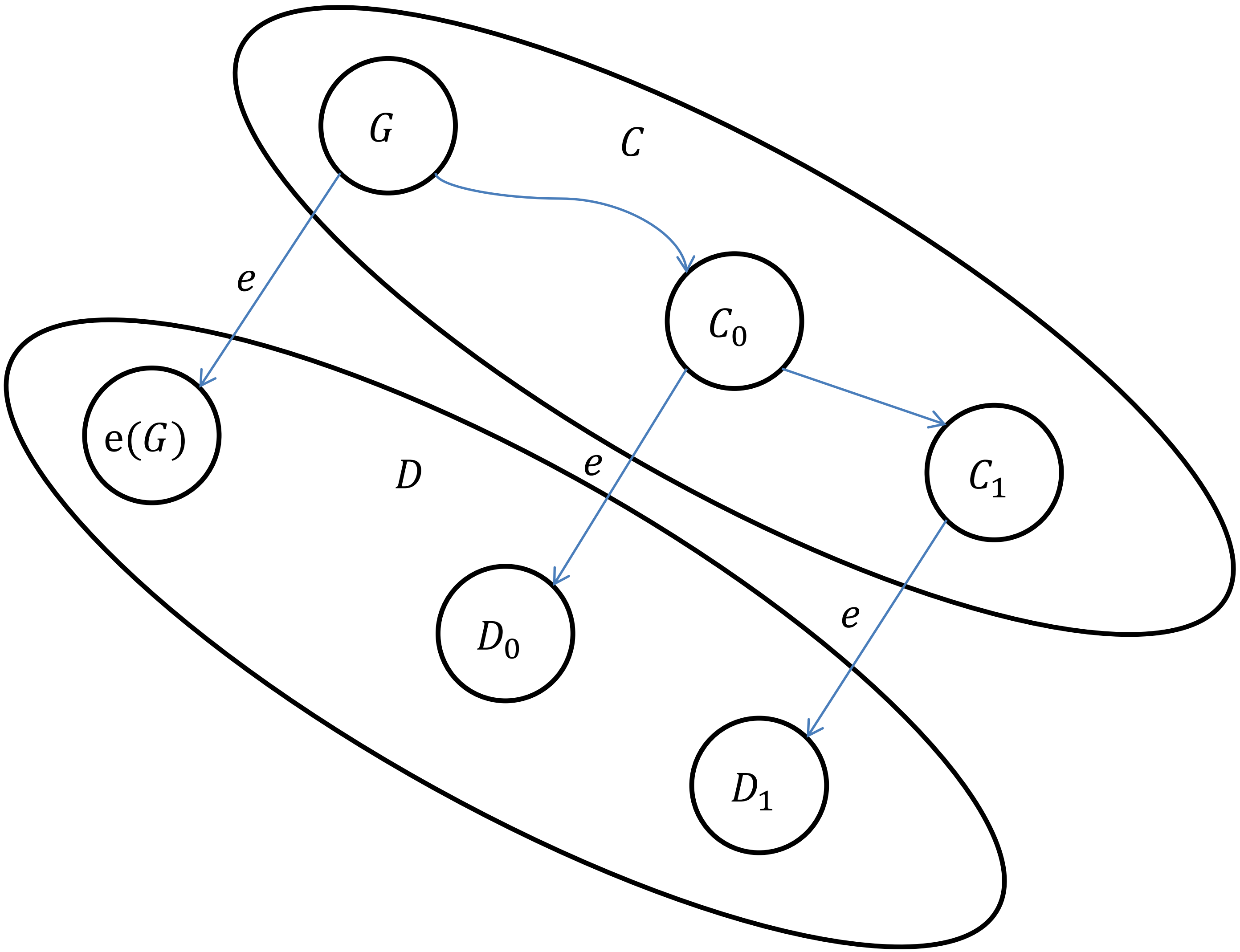
FLP Лемма 2: Иллюстрация



FLP Лемма 2.1: i-валентные конфигурации

- Докажем что есть i-валентная конфигурация в D для любого i (0 или 1)
- Так как G бивалентная конфигурация, то по какой-то цепочке шагов из неё можно дойти до i-валентной E_i.
  - Если $E_i \in D$ то мы нашли искомую конфигурации
  - Если $E_i \in C$ то тогда $e(C) \in D$ искомая конфигурация
  - В противном случае e применялась в цепочке шагов для достижения E_i из G, а значит есть $F_i \in D$ (сразу после применения e) из которой доступна E_i по какой-то цепочке шагов
    - Но так как мы предположили, что в D нет бивалентных конфигураций, то $F_i \in D$ будет i-валентной

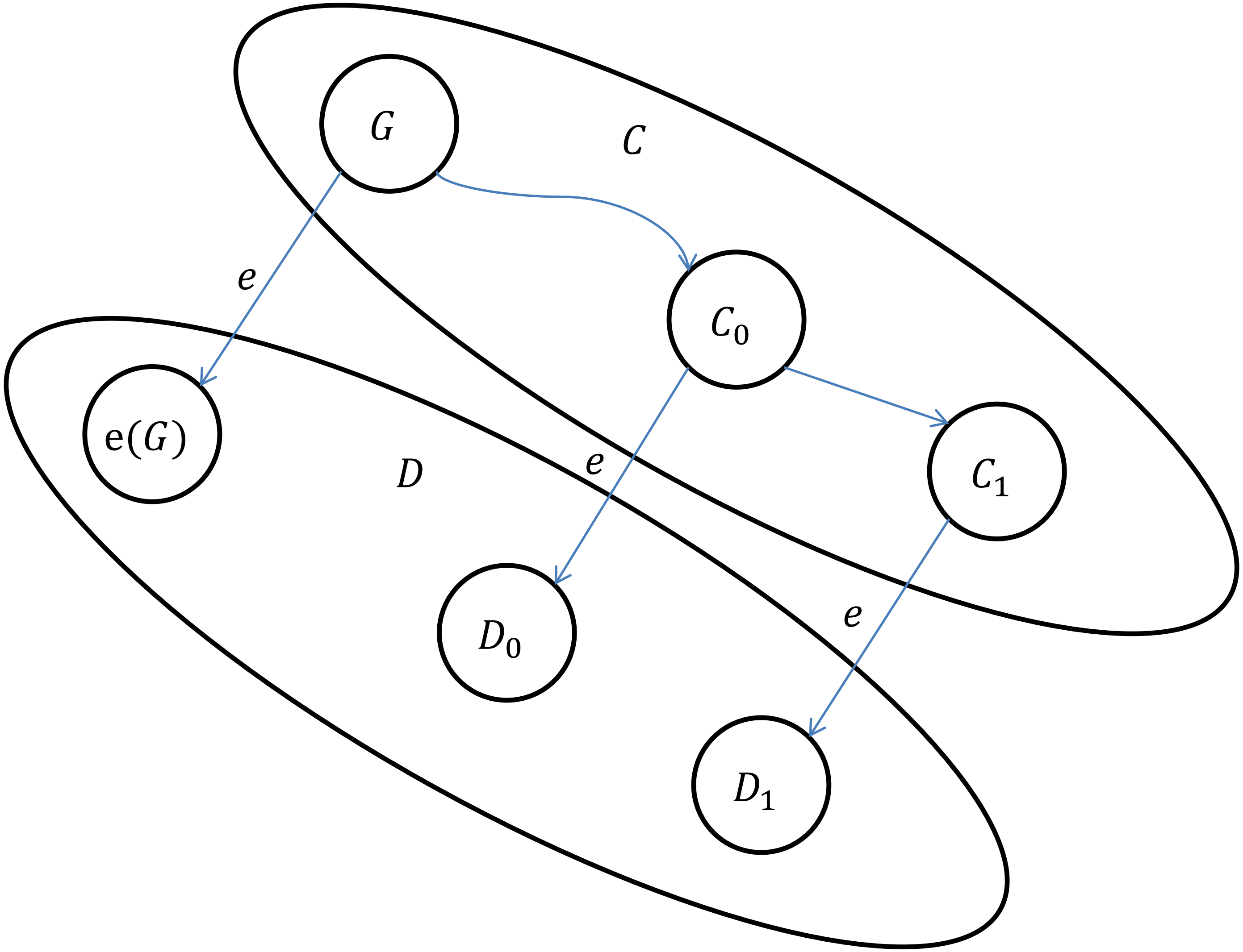
FLP Лемма 2: Иллюстрация



FLP Лемма 2.2: Соседние конфигурации

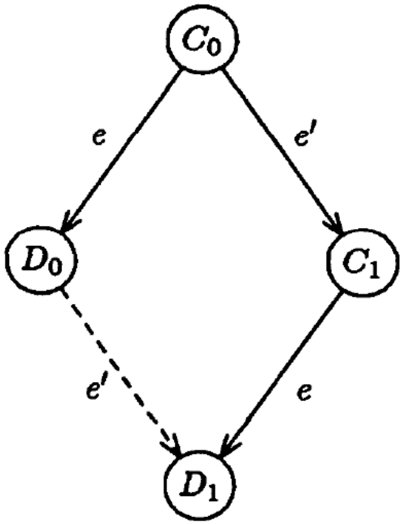
- Найдем такие соседние (отличающиеся одним шагом e') $\mathrm{C}_0 \in \mathrm{C}$ и $\mathrm{C}_1 \in \mathrm{C}$ что $D_0 = e(C_0) \in D$ является 0-валентной, а $D_1 = e(C_1) \in D$ является 1-валентной (например)
  - Пусть, не теряя общности, $C_0 = e'(C_1)$, где событие e' произошло на процессе p'
- Как это сделать:
  - Пусть, не теряя общности, $e(G) \in D$ является 0-валентной (если она 1-валентная, то симметрично)
    - Она соответствует пустой цепочке шагов из G.
  - Тогда по лемме 2.1 в D есть 1-валентная конфигурация $D_1 = e(C_1) \in D$
  - Будем убирать из цепочки шагов ведущей от G к C_1 по одному шагу с конца, пока не найдем искомую пару соседей C_0 и C_1

FLP Лемма 2: Иллюстрация



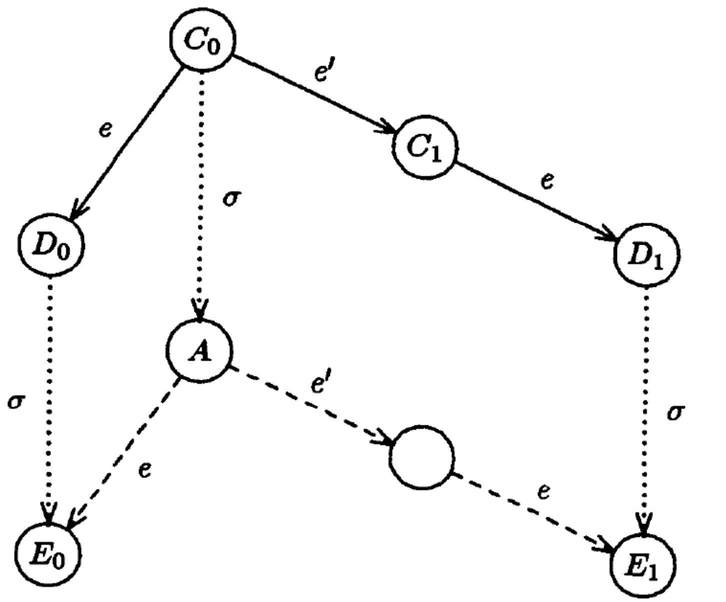
FLP: Разбор случая 1

- **Случай 1**: Если $p \neq p'$ то e и e' коммутируют
  - Получается что D_1 должна быть одновременно 1- и 0- валентной. Противоречие



FLP: Разбор случая 2

- **Случай 2:** Если $p = p'$ то рассмотрим цепочку шагов σ от состояния C_0 где процесс p отказал (не выполняется), а остальные пришли к решению
  - Тогда конфигурация $A = \sigma(C_0)$, с решением, должна быть 0- или 1-валентной
  - Но $E_0 = e(A) = \sigma(D_0)$ 0-валентая, а $E_1 = e\big(e'(A)\big) = \sigma(D_1)$ 1-валентная



FLP: Дополнительные наблюдения

- Результат FLP о невозможности консенсуса верен даже если процессу разрешено делать операцию «атомарной передачи» сообщения сразу несколько процессам
  - См. определение шага от одной конфигурации к другой в модели системы теоремы FLP
  - Однако, нет гарантии что все процессы обработают его
    - Один процесс может умереть и не получить

Применения консенсуса

- Terminating Reliable Broadcast
- Выбор лидера

Terminating Reliable Broadcast (TRB)

- Если есть гарантия получения сообщения всеми процессами (или ни одним), то такая операция называется **Terminating Reliable Broadcast** (TRB)
  - Имея TRB можно тривиально на его основе написать алгоритм консенсуса
    - Каждый процесс делает TRB своего предложения
    - Приходим к консенсусу использую детерминированную функцию от полученных предложений

Terminating Reliable Broadcast (TRB)

- **TRB эквивалентен консенсусу**
  - TRB ⇒ Консенсус
  - Консенсус ⇒ TRB
    - Консенсус на одном бите (обрабатываем или не обрабатываем сообщение)

Выбор лидера

- **Задача выбора лидера**
  - Из множества процессов надо выбрать одного (лидера)
  - За конечное время
  - Обычно нужно для координации дальнейшего алгоритма

Выбор лидера

- **Консенсус эквивалентен выбору лидера**
- Консенсус ⇒ Выбор лидера
  - Каждый процесс предлагает себя в качестве лидера
  - Решение алгоритма консенсуса (обоснованное) определяет выбор лидера
- Выбор лидера ⇒ Консенсус
  - Сначала выбираем лидера
  - Лидер шлет всем свое предложение
  - Процессы соглашаются с предложением лидера

Практические выводы

- **Консенсус нужен на практике**

- Но FLP говорит что все четыре свойства не могут быть удовлетворены одновременно:
  1. **Асинхронная система**
  2. Детерминированный алгоритм
  3. Возможность отказа узла
  4. Конечное время достижения консенсуса

- **Значит надо от чего-то отказаться**

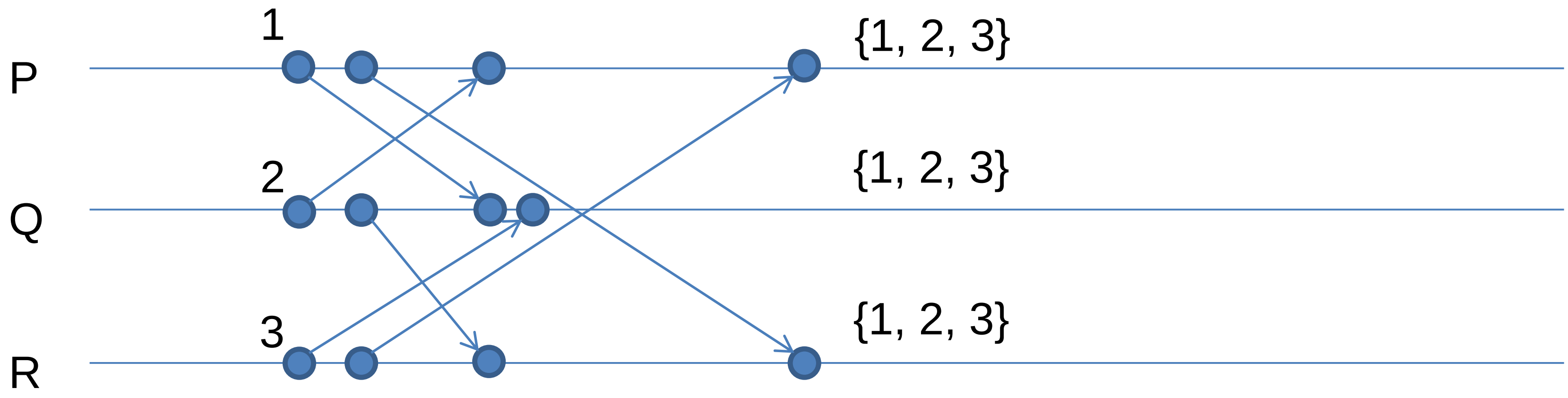
СИНХРОННАЯ СЕТЬ: ОТКАЗ УЗЛА

Синхронная сеть + отказ узла

- Если из N узлов могут оказать f $(0 \leq f < N)$
- Базовый алгоритм – ???

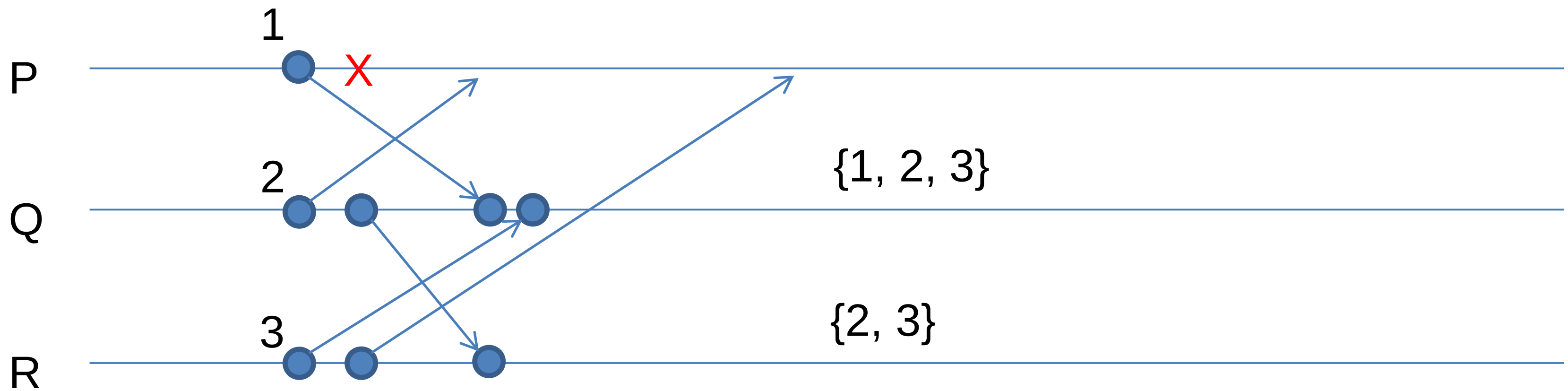
Синхронная сеть + отказ узла

- Если из N узлов могут оказать $f\ (0 \leq f < N)$
- Базовый алгоритм – пересылаем все известные предложения всем другим узлам. Что может пойти не так?



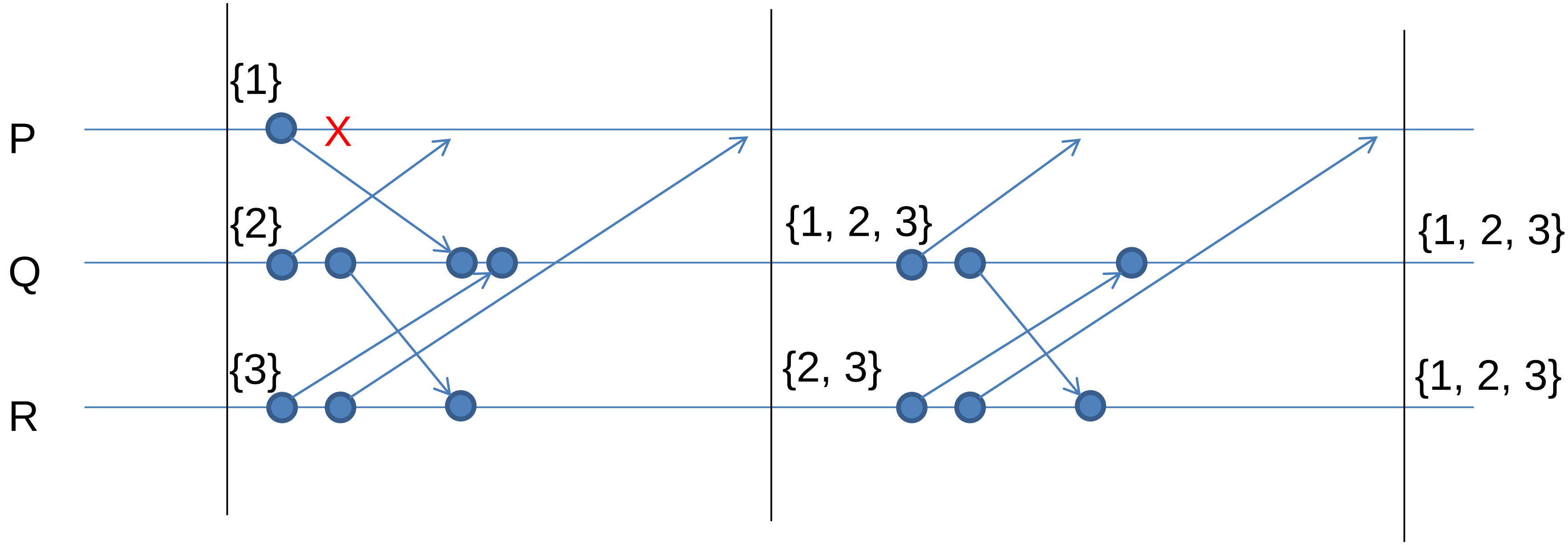
Синхронная сеть + отказ узла

- Если из N узлов могут оказать f $(0 \leq f < N)$
- Базовый алгоритм – пересылаем все известные предложения всем другим узлам. Что может пойти не так?



Синхронная сеть + отказ узла

- Если из N узлов могут оказать f $(0 \leq f < N)$
- Делаем $f + 1$ фазу базового алгоритма
  - Рассылаем известные *множества* предложений
  - Фаза алгоритма = максимальное время доставки сообщения



Синхронная сеть + отказ узла

- Если из N узлов могут оказать f $(0 \leq f < N)$
- Делаем $f + 1$ фазу базового алгоритма
  - Рассылаем известные *множества* предложений
  - Фаза алгоритма = максимальное время доставки сообщения
- Алгоритм корректен по принципу Дирихле
  - Хотя бы в одной фазе не будет отказов

СИНХРОННАЯ СЕТЬ: ПОТЕРЯ СООБЩЕНИЯ

Синхронная сеть на практике

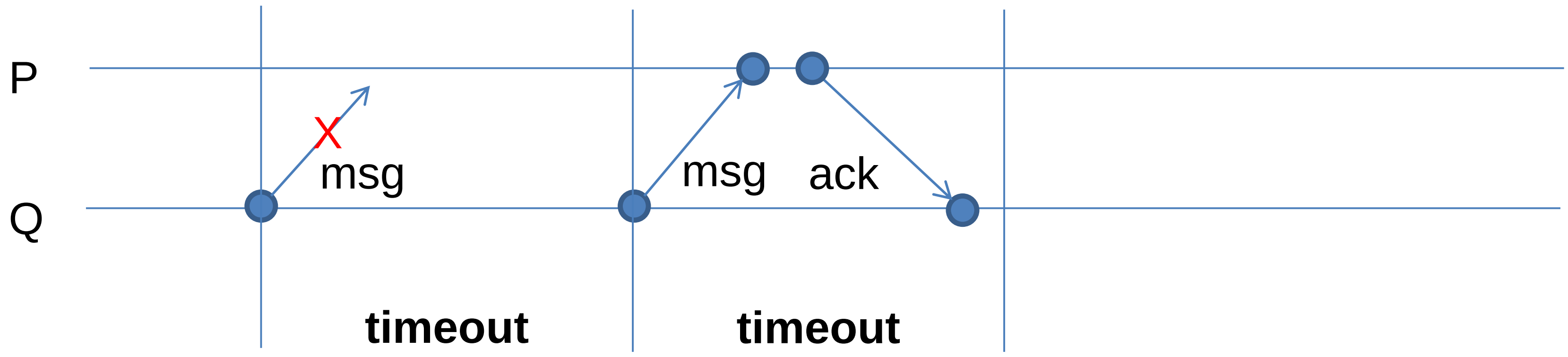
- На практике часто вводят некий **timeout** – максимальное практически возможное время доставки сообщения
    - Сообщение доставлено вовремя – Ok
    - Сообщение не доставлено вовремя – считаем потерянным

- **Такую сеть можно считать синхронной!**

- На практике **timeout** часто задает максимальное время доставки туда и обратно (в две стороны)

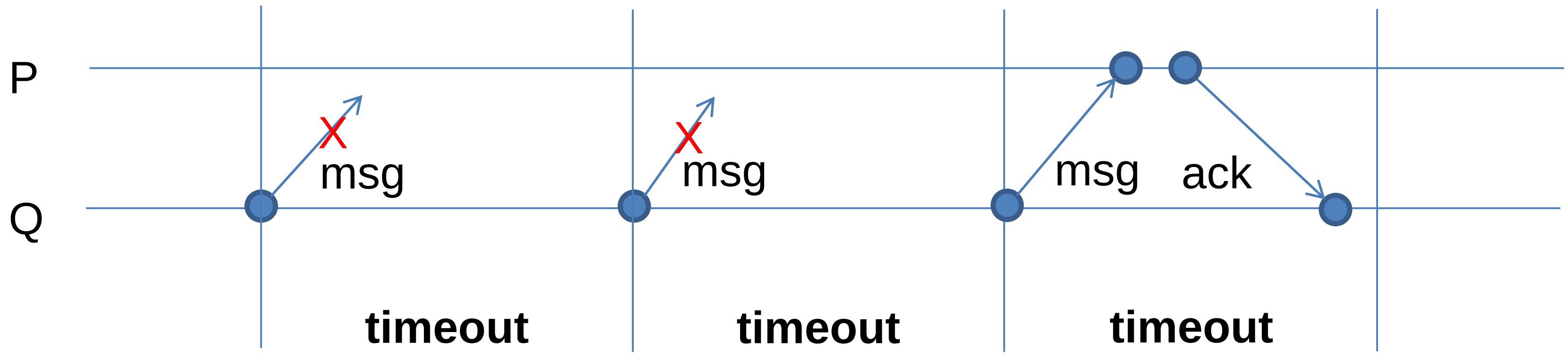
Возможные гарантии доставки

- **At least once**
  - Сообщение должно быть доставлено хотя бы раз
  - Алгоритм:
    - Требуем подтверждения
    - Если подтверждения нет в течение timeout посылаем снова и снова, пока не получим подтверждение



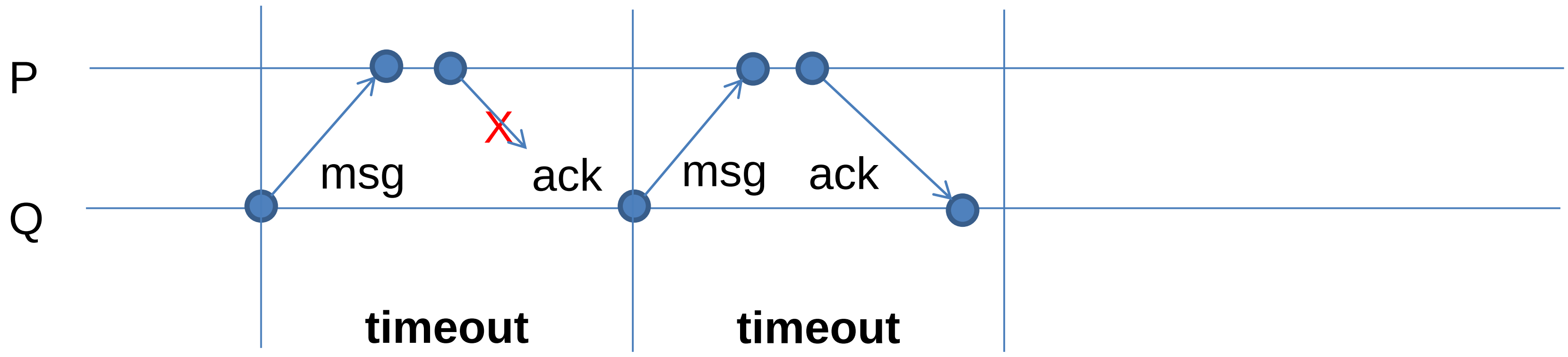
Возможные гарантии доставки

- **At least once**
  - Сообщение должно быть доставлено хотя бы раз
  - Алгоритм:
    - Требуем подтверждения
    - Если подтверждения нет в течение timeout посылаем снова и снова, пока не получим подтверждение



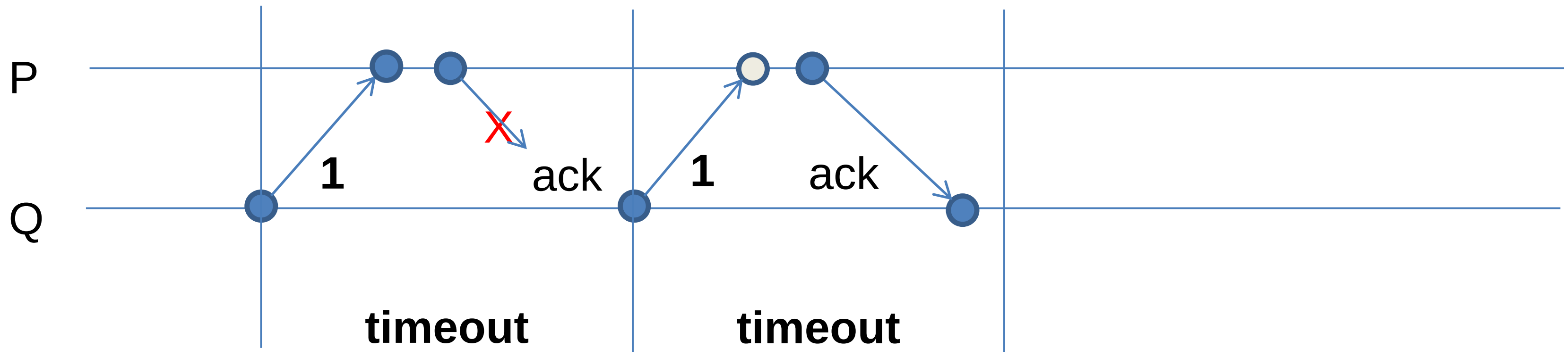
Возможные гарантии доставки

- **Exactly once**
  - Сообщение должно быть **ровно один** раз



Возможные гарантии доставки

- **Exactly once**
  - Сообщение должно быть **ровно один** раз
  - Алгоритм: **at least once** + **at most once**
  - **At most once:**
    - Нумеруем сообщения
    - Сообщение пришло повторно – игнорируем но подтверждаем



Умеем надежно доставлять сообщения несмотря на потери

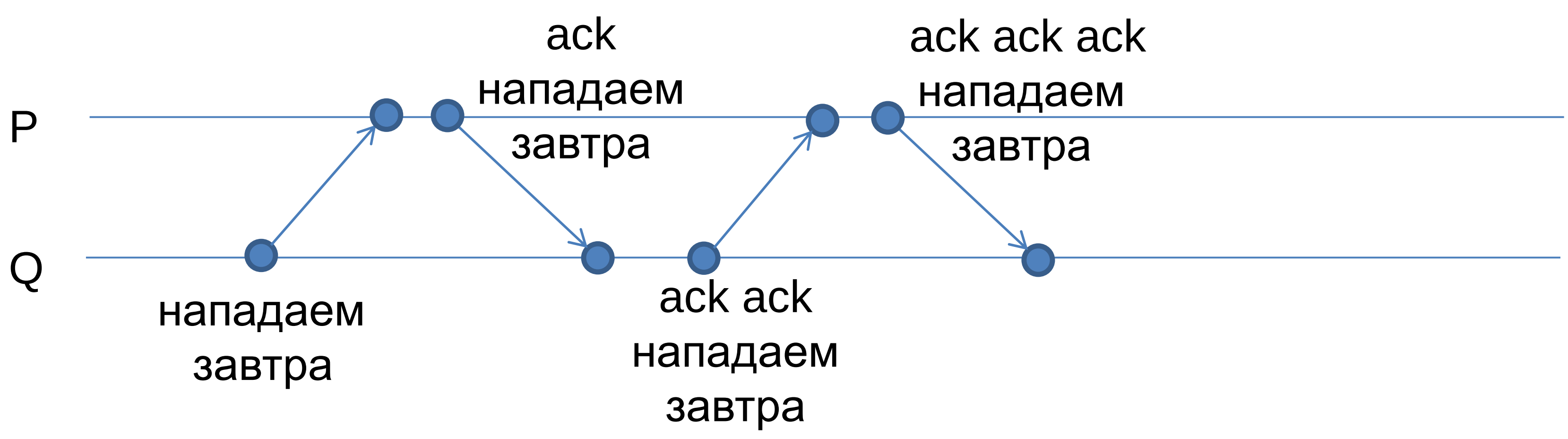
МОЖНО ЛИ ТАК ПРИЙТИ К КОНСЕНСУСУ?

Синхронная сеть + ненадежная доставка

- Проблема **двух генералов**
  - Два генерала осаживают вражескую крепость
  - Им нужно или напасть утром на крепость вместе или не нападать вовсе (иначе разобьют поодиночке)
  - **Двум генералам необходимо прийти к консенсусу, но враг может перехватить гонцов.**

Синхронная сеть + ненадежная доставка

- Проблема **двух генералов**
  - Двум генералам необходимо прийти к консенсусу, но враг может перехватить гонцов.



Синхронная сеть + ненадежная доставка

- Проблема **двух генералов**
  - Двум генералам необходимо прийти к консенсусу, но враг может перехватить гонцов.

- **Из-за ненадежности доставки невозможно прийти к консенсусу**
  - Доказательство анализом последнего сообщения

- То же и для большего числа процессов если *каждый* канал ненадежен
  - Чтобы рабочие процессы могли прийти к консенсусу **нужны какие-то *надежные* каналы передачи данных**
  - Ненадежный канал к процессу ~= сбойный процесс

СИНХРОННАЯ СЕТЬ: ВИЗАНТИЙСКАЯ ОШИБКА

Синхронная сеть + Византийская ошибка процесса

- Проблема **Византийских генералов**
  - N генералов хотят прийти к консенсусу, но среди них f предателей

Синхронная сеть + Византийская ошибка процесса

- Проблема **Византийских генералов**
  - N генералов хотят прийти к консенсусу, но среди них f предателей
- При Византийской ошибке
  - Решение возможно в синхронной системе только если $N > 3f$

Синхронная сеть + Византийская ошибка процесса

- Проблема **Византийских генералов**
  - N генералов хотят прийти к консенсусу, но среди них f предателей
- При Византийской ошибке
  - Решение возможно в синхронной системе только если $N > 3f$
  - 2-х фазный алгоритм решения при $N = 4$, $f = 1$
    - В первой фазе все процессы шлют предложение всем
    - Во второй фазе все пересылают всю полученную в первой фазе информацию (вектор) всем другим процессам
    - После 2-ой фазы у каждого процесса есть матрица информации

Анализ алгоритма для N=4, f=1

- У каждого не сбойного процесса есть матрица полученных данных из 2-й фазы (диагональ не учитываем)

Кто получил?

D - Византийский

От кого получил?

	A	B	C	D	большинство
A	x	x	x	t_4	x
B	y	y	y	t_5	y
C	z	z	z	t_6	z
D	t_1	t_2	t_3	t_7	t

Одно у всех!

Синхронная сеть + Византийская ошибка процесса

- Проблема **Византийских генералов**
  - N генералов хотят прийти к консенсусу, но среди них f предателей
- При Византийской ошибке
  - Решение возможно в синхронной системе только если $N > 3f$
  - 2-х фазный алгоритм решения при $N = 4$, $f = 1$
    - В первой фазе все процессы шлют предложение всем
    - Во второй фазе все пересылают всю полученную в первой фазе информацию (вектор) всем другим процессам
    - После 2-ой фазы у каждого процесса есть матрица информации
    - **Обобщается на общий алгоритм с $f + 1$ фазами**

Невозможность Византийского консенсуса

- Византийский консенсус невозможен при $N \leq 3f$
- Простое доказательство при $N = 3, f = 1$
  - От противного. Запустим алгоритм в 4-х копиях