

Срезы, снимки состояния, предикаты

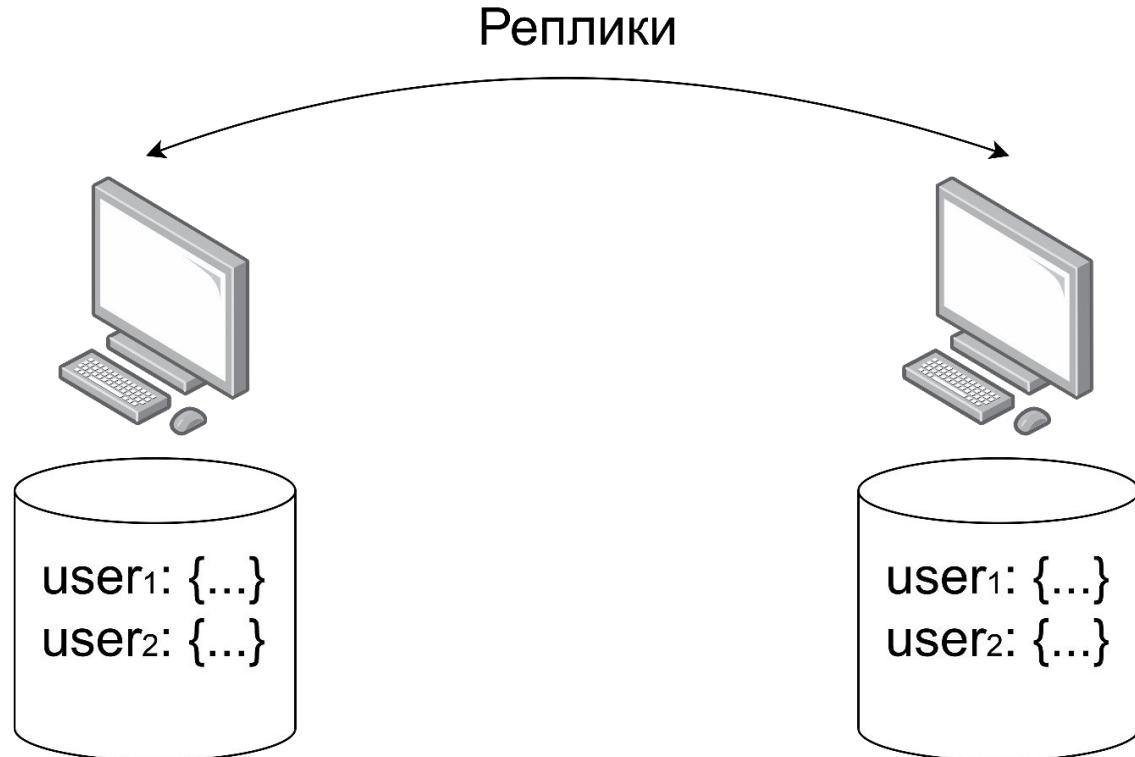


Илья Кокорин

kokorin.ilya.1998@gmail.com

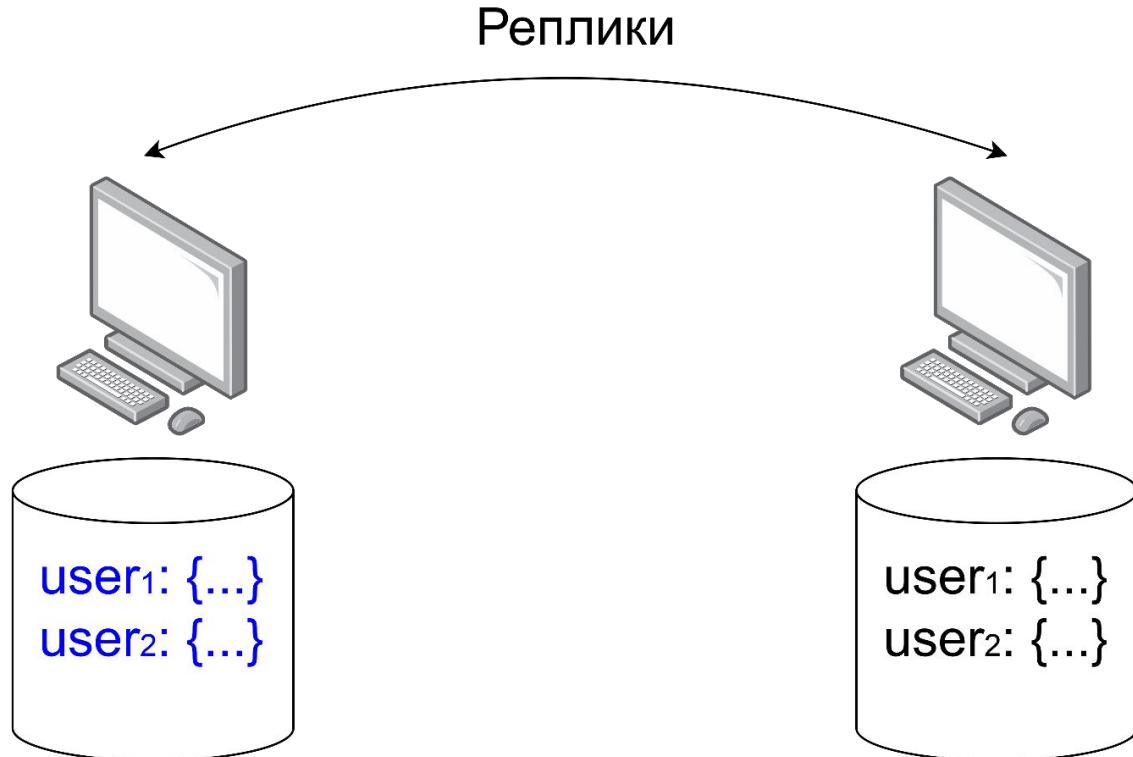
Неопределённость состояния

- Хотим остановить систему и посмотреть на её состояние



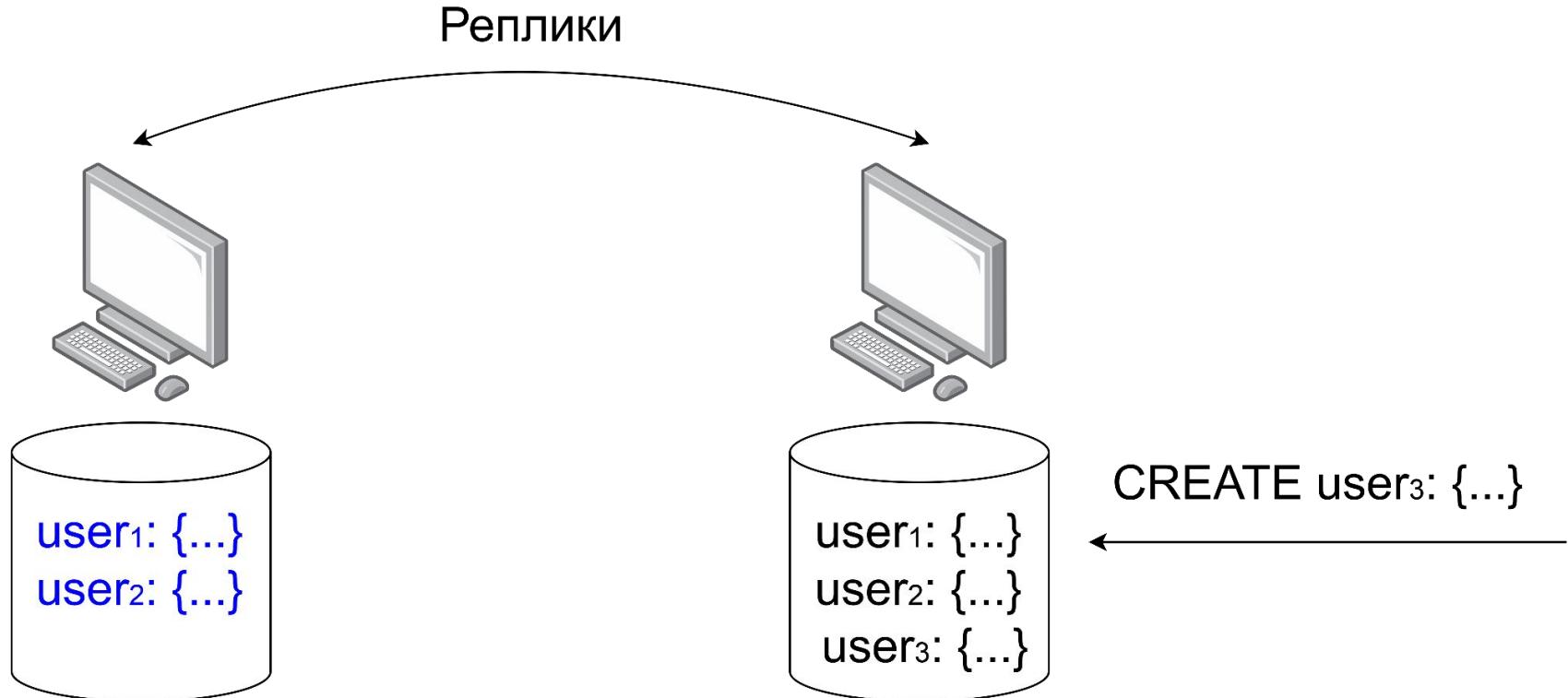
Неопределённость состояния

- Оба состояния одновременно сохранить не можем



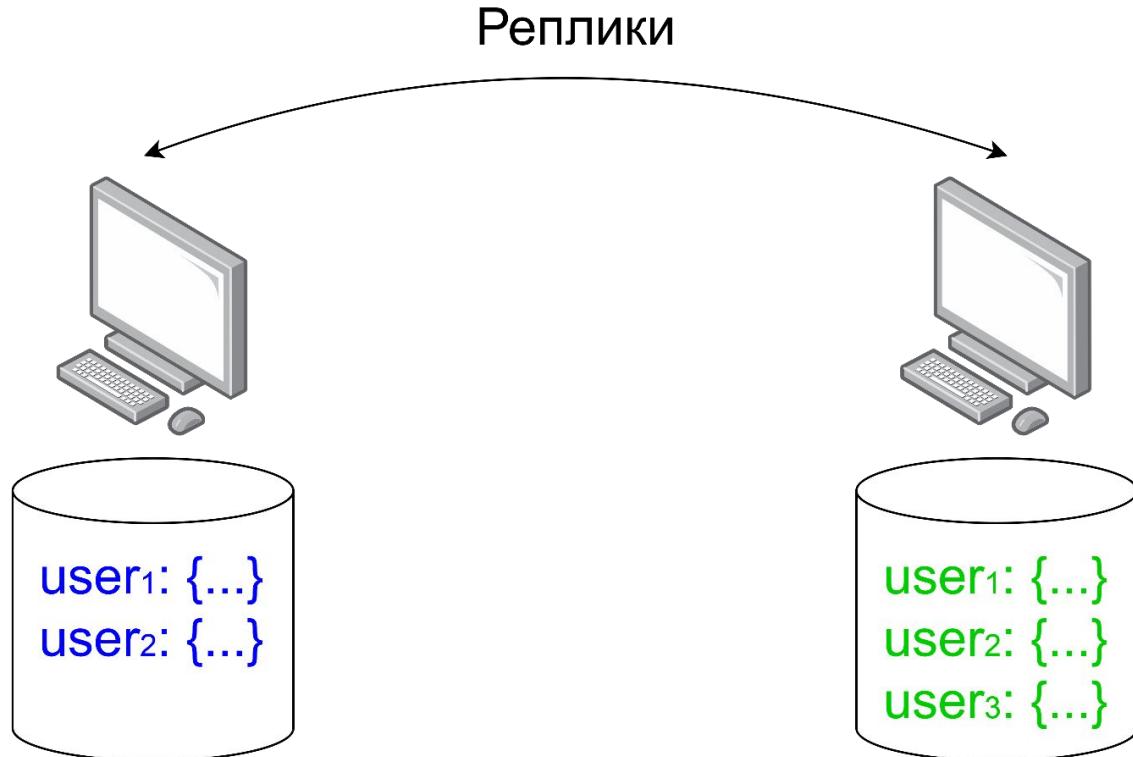
Неопределённость состояния

- Состояние одного из узлов обновляется



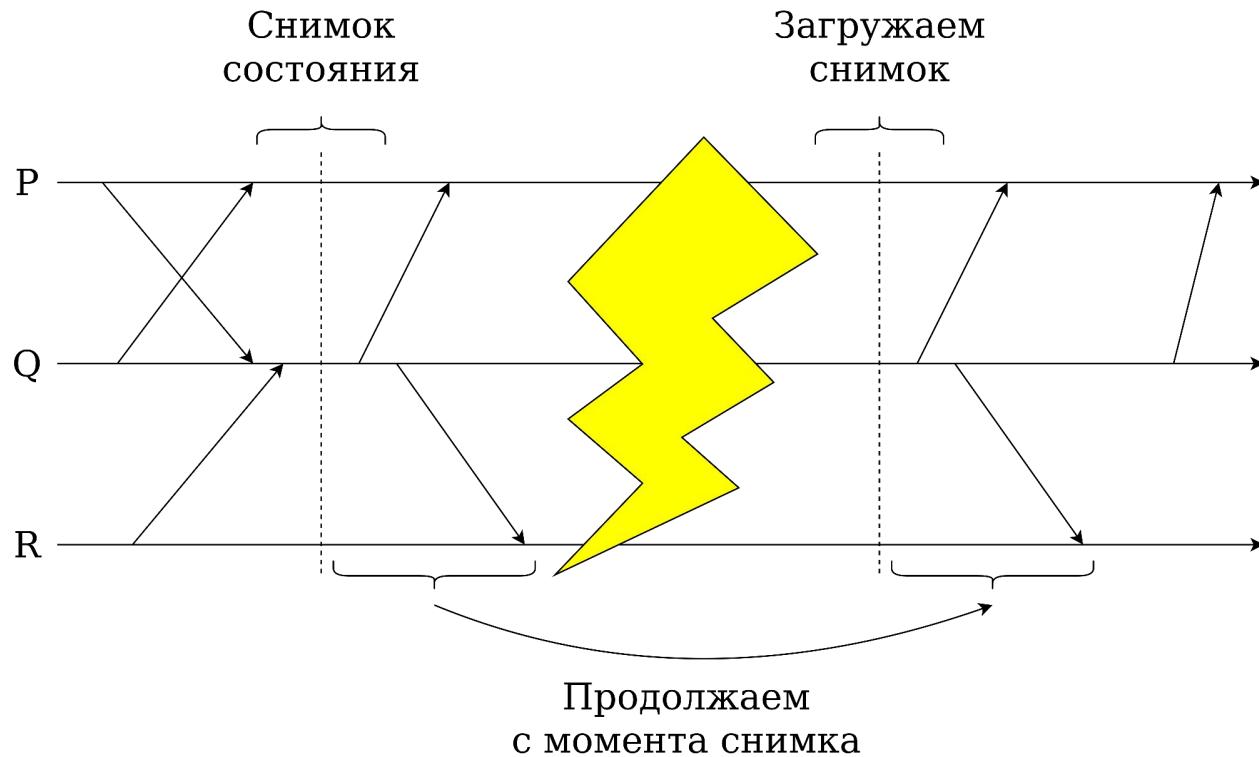
Неопределённость состояния

- На втором узле состояние успело измениться



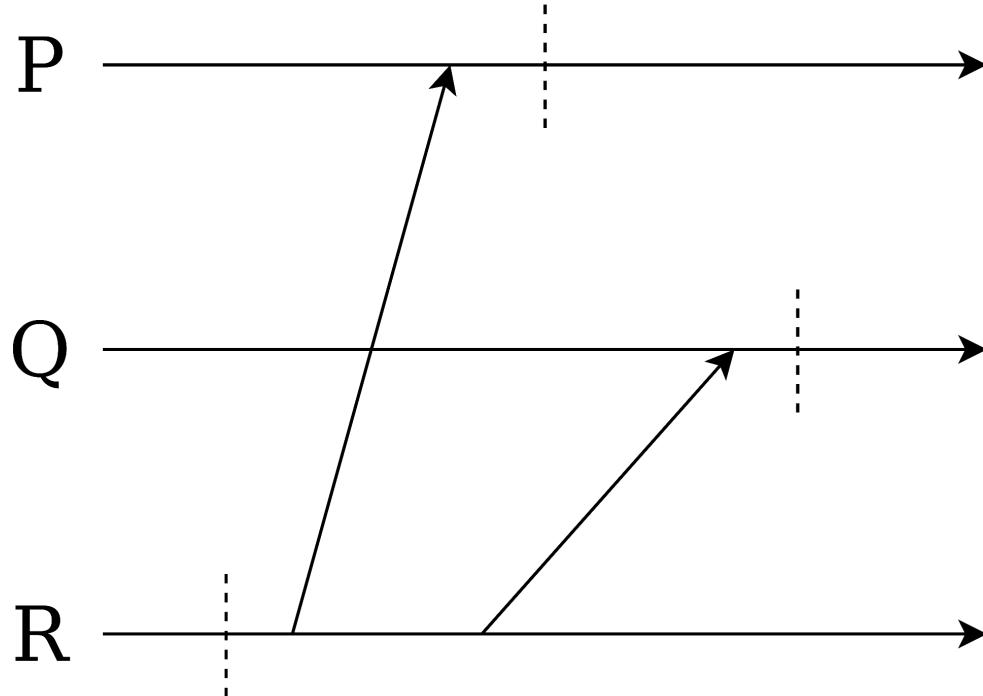
Мотивирующий пример

- Сохраняем снимок состояния на диск
- При сбое повторяем вычисление не с самого начала
- А с момента снимка



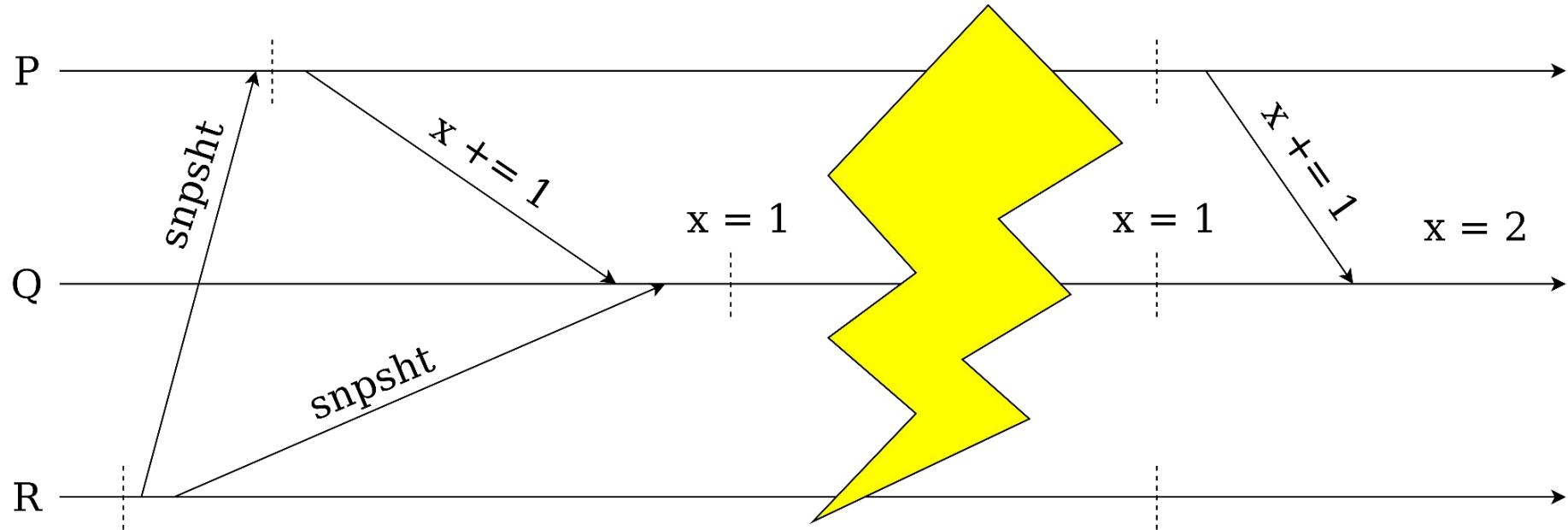
Наивный алгоритм

- Инициатор рассыпает всем процессам команду “Сделать снимок”
- Получив команду, процесс сохраняет своё состояние
- Работает корректно?



Наивный алгоритм

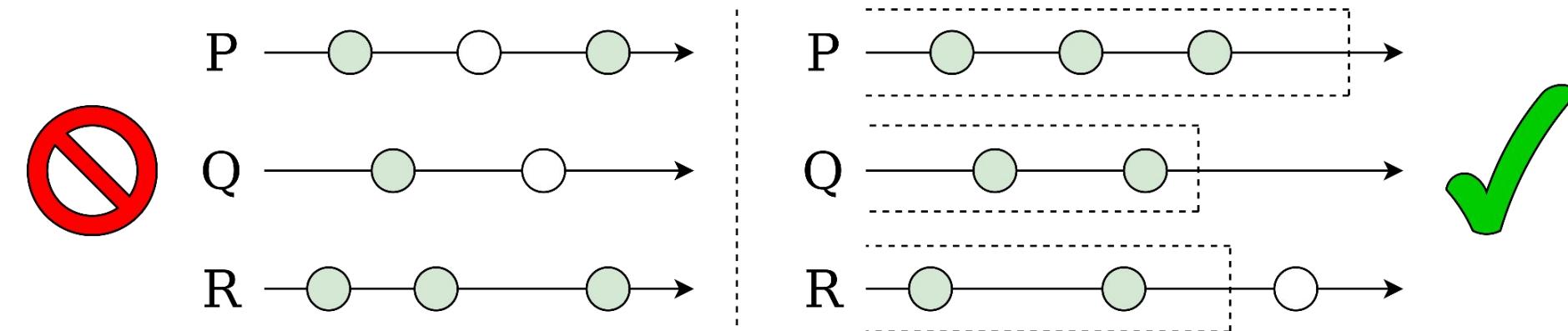
- Состояние меняется дважды
- Отправка сообщения $x += 1$ не попала в снимок
 - Но попало получение



Срез: определение

- $S \subset \mathbb{E}$ называется срезом, если

$$\forall x, y \in \mathbb{E} : proc(x) = proc(y), x < y, y \in S \Rightarrow x \in S$$

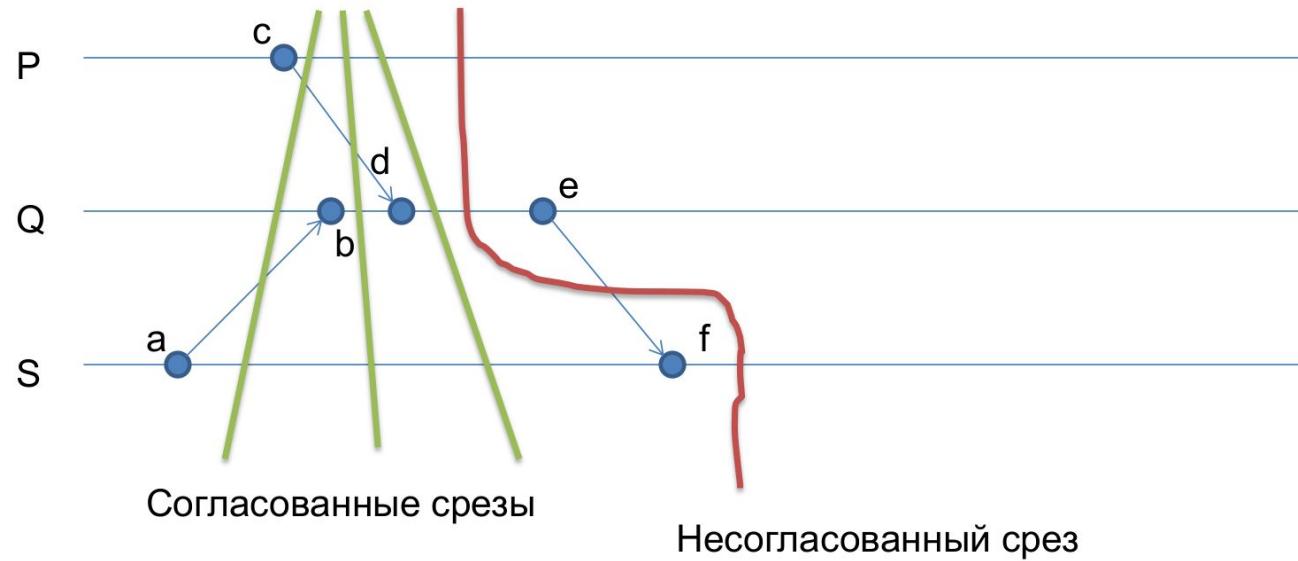


- На каждом процессе выбираем точку
- Все события левее этой точки входят в срез

Согласованный срез: определение

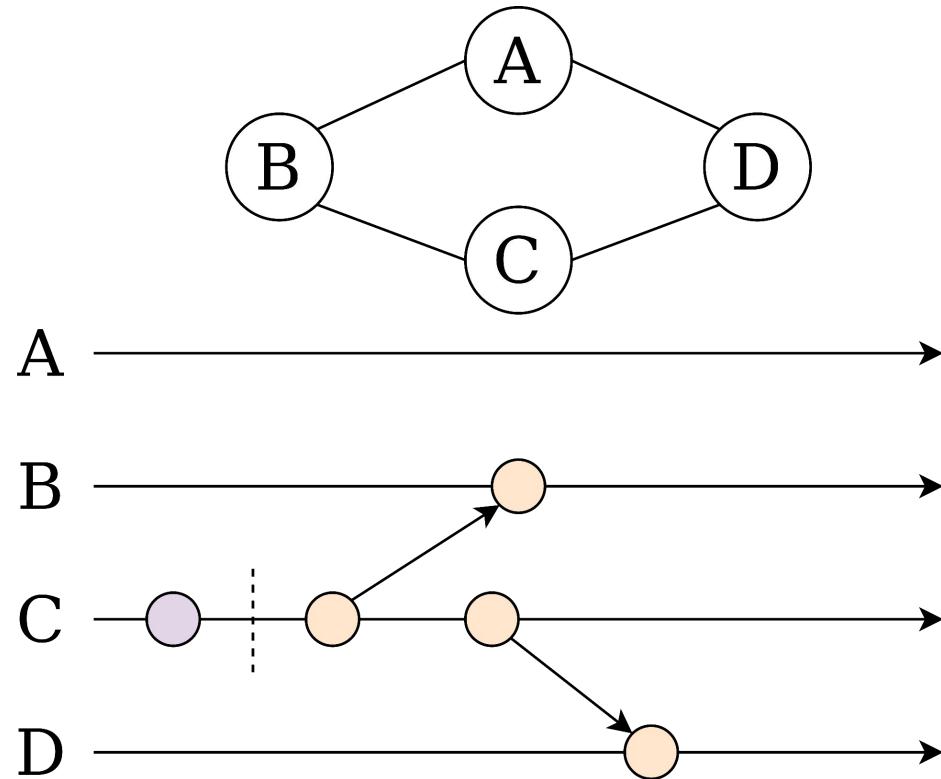
- Срез S называется согласованным срезом, если

$$\forall x, y \in \mathbb{E} : x \rightarrow y, y \in S \Rightarrow x \in S$$



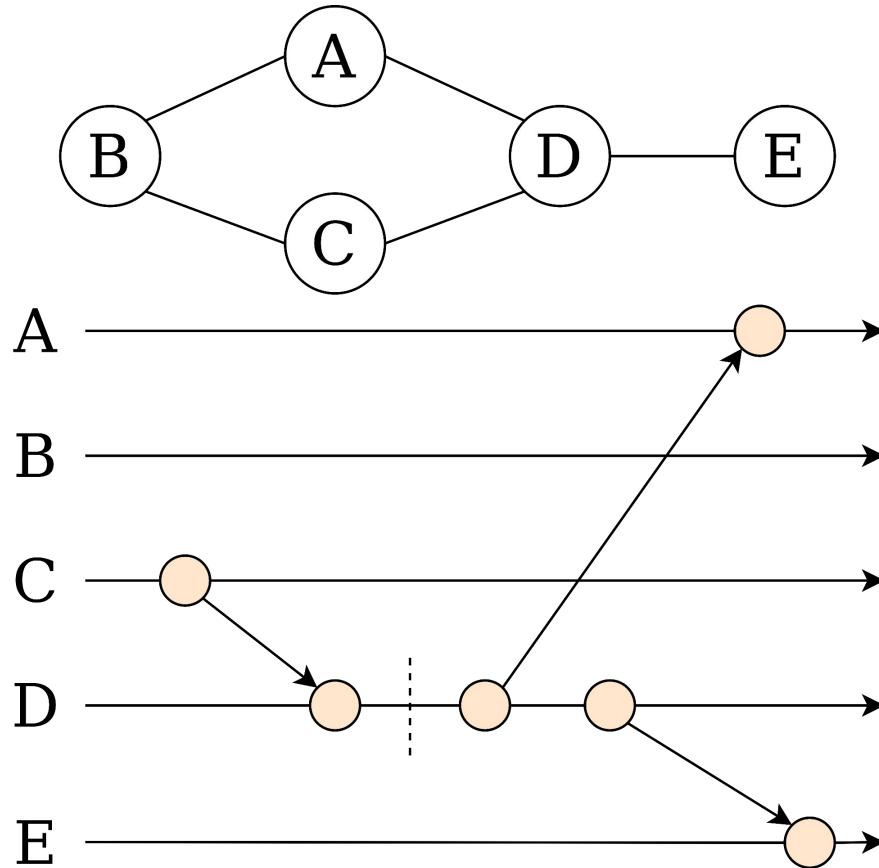
Алгоритм Чанди и Лампорта

- Процесс-инициатор решает, что хочет сделать снимок состояния
- Делает снимок
- Рассыпает маркер своим соседям



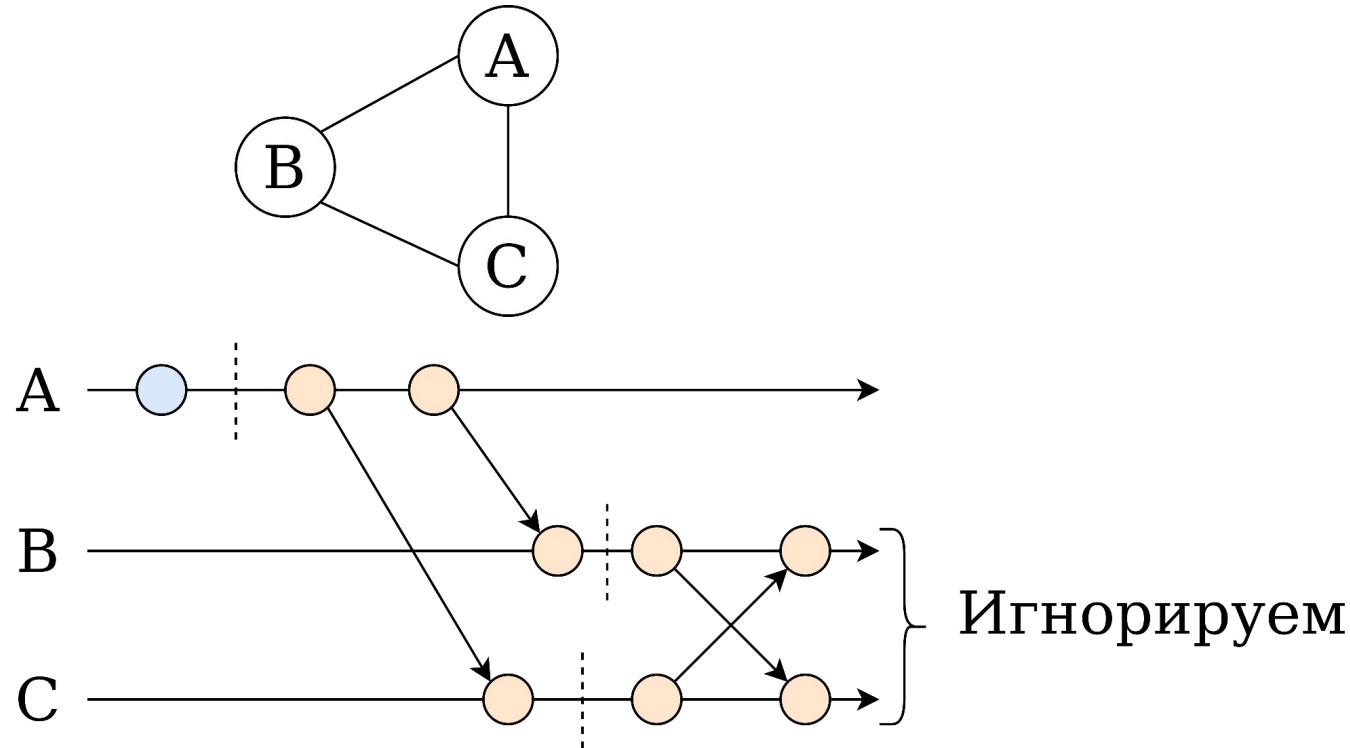
Алгоритм Чанди и Лампорта

- Получив маркер, процесс сохраняет состояние
- И рассыпает маркер соседям
 - Можно не посыпать тому, от кого пришёл запрос
- Если ещё не сохранял состояние до этого



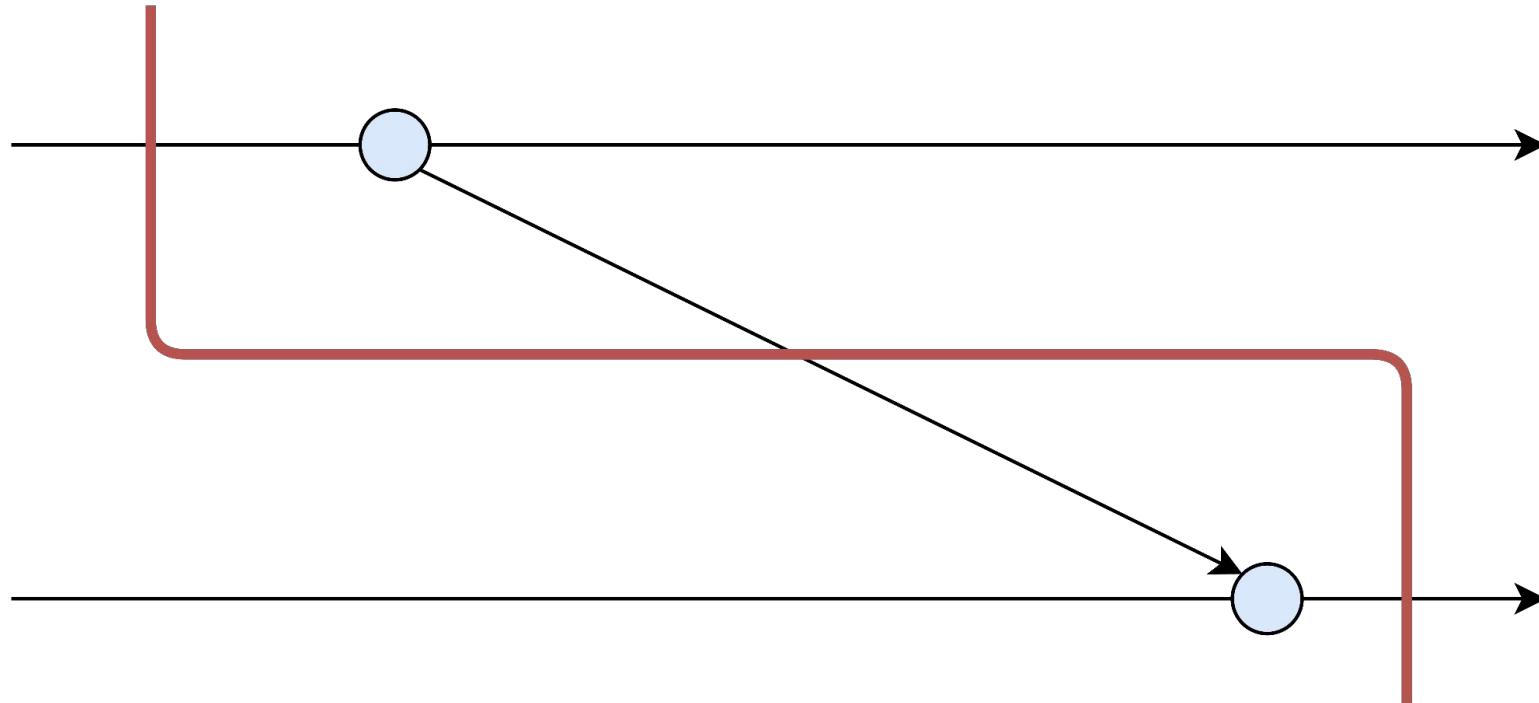
Алгоритм Чанди и Лампорта

- Если процесс уже сохранял состояние, он игнорирует дальнейшие маркеры



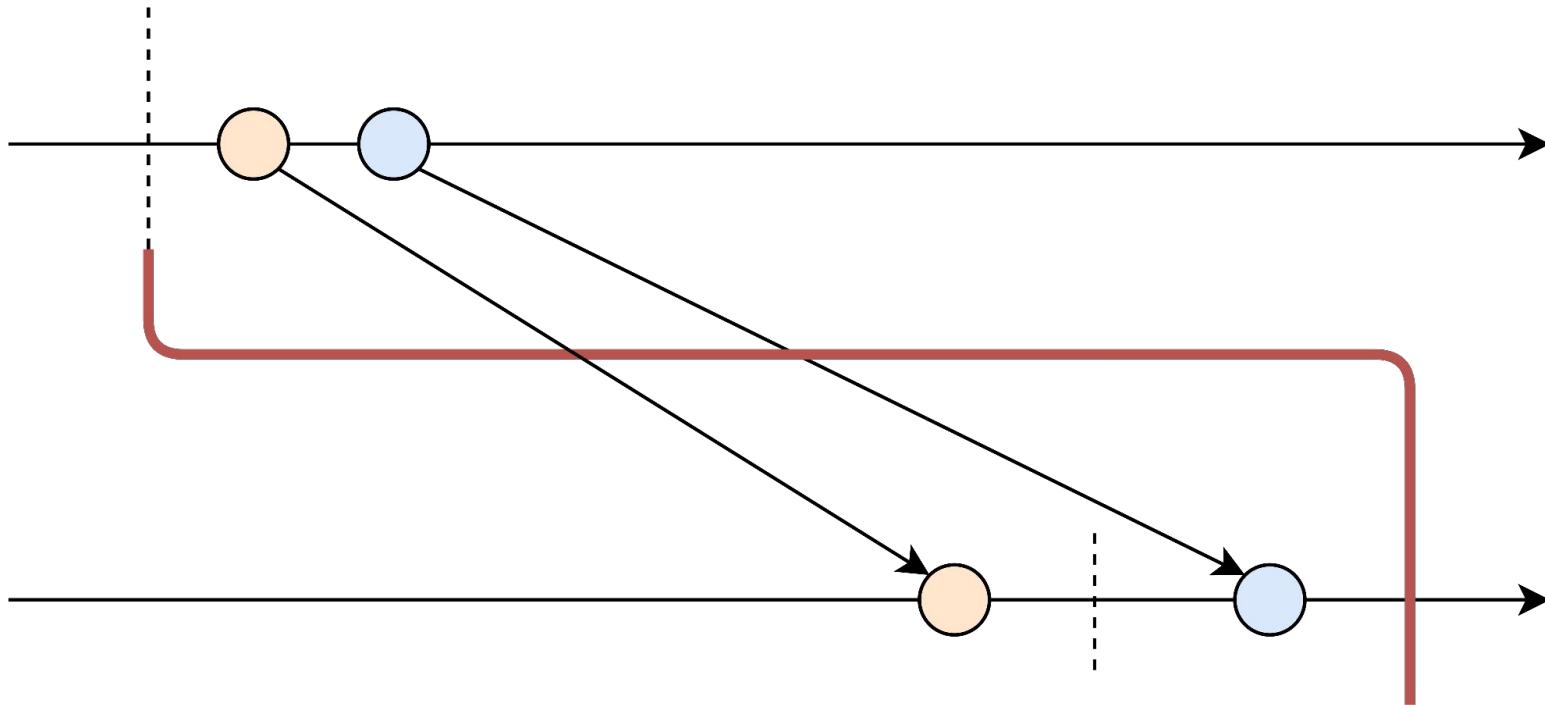
Алгоритм Чанди и Лампорта

- Может ли получиться несогласованный срез?



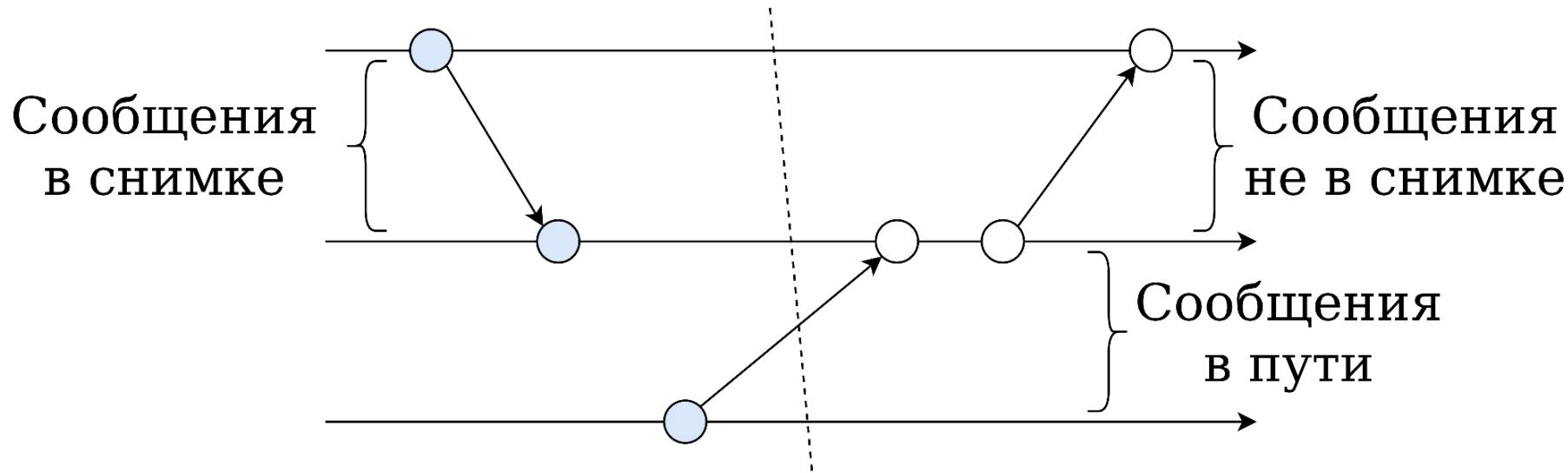
Алгоритм Чанди и Лампорта

- Нет, если сообщения идут FIFO



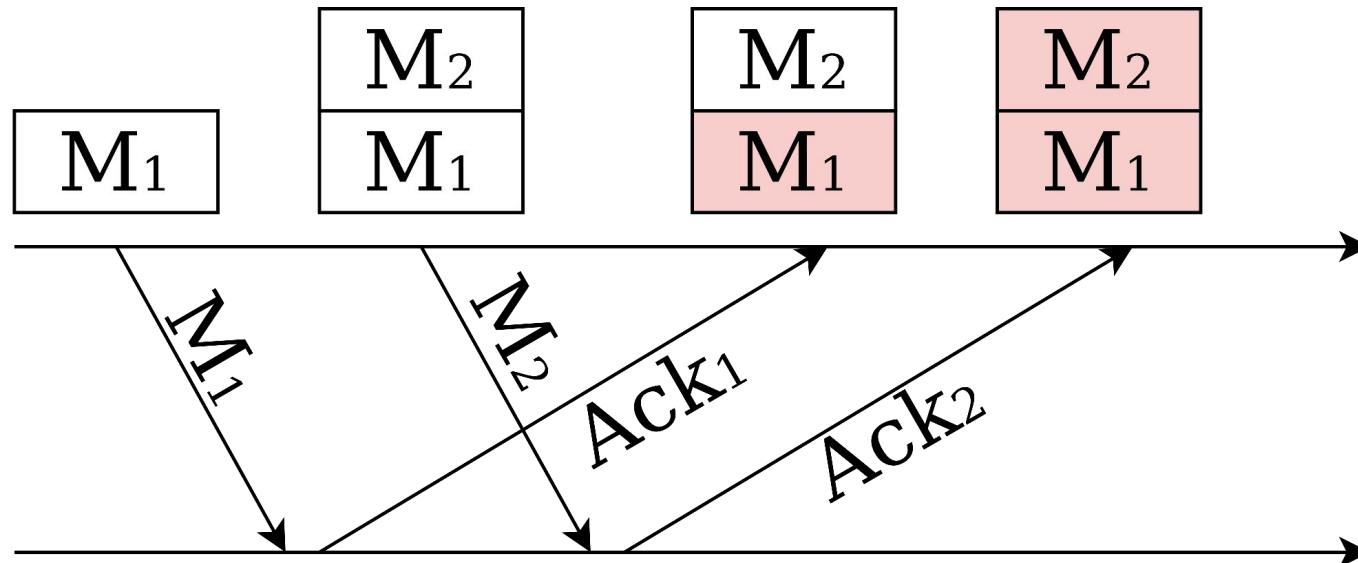
Сообщения в пути

- Сообщения в пути нужно сохранять
- Чтобы при восстановлении из снимка доставить их заново



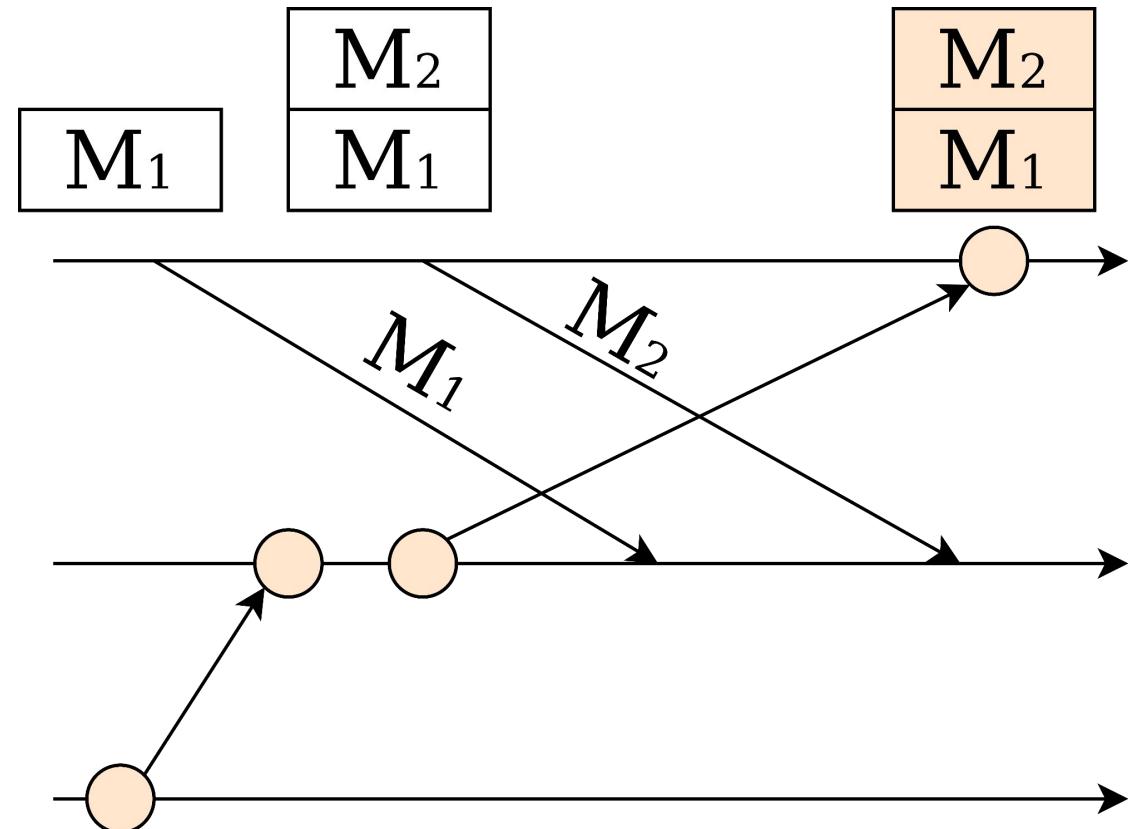
Запоминание на стороне отправителя

- Каждое отправленное сообщение запоминаем локально
 - Если мы пока не делали снимок
- Удаляем, получив подтверждение
- Получатель подтверждает получение каждого сообщения



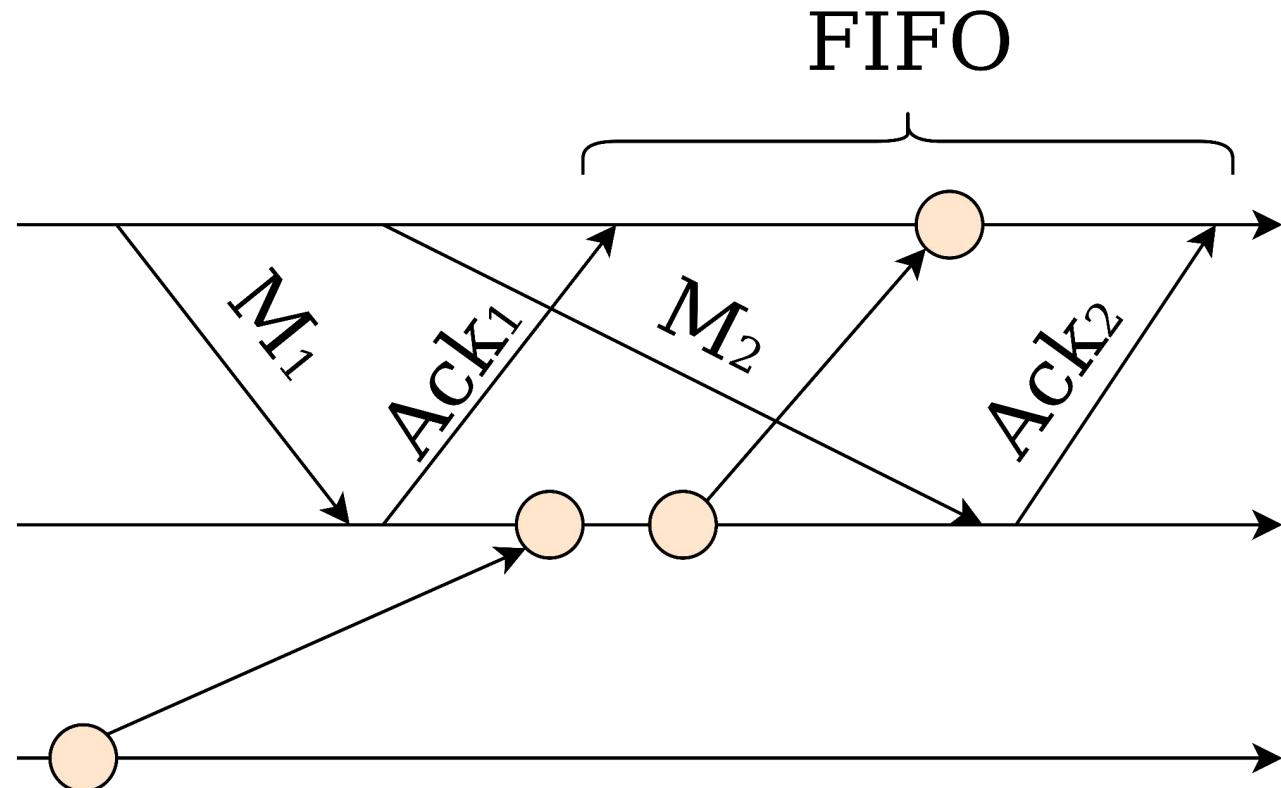
Запоминание на стороне отправителя

- Получив токен от процесса P , запоминаем все неподтверждённые сообщения в пути до P



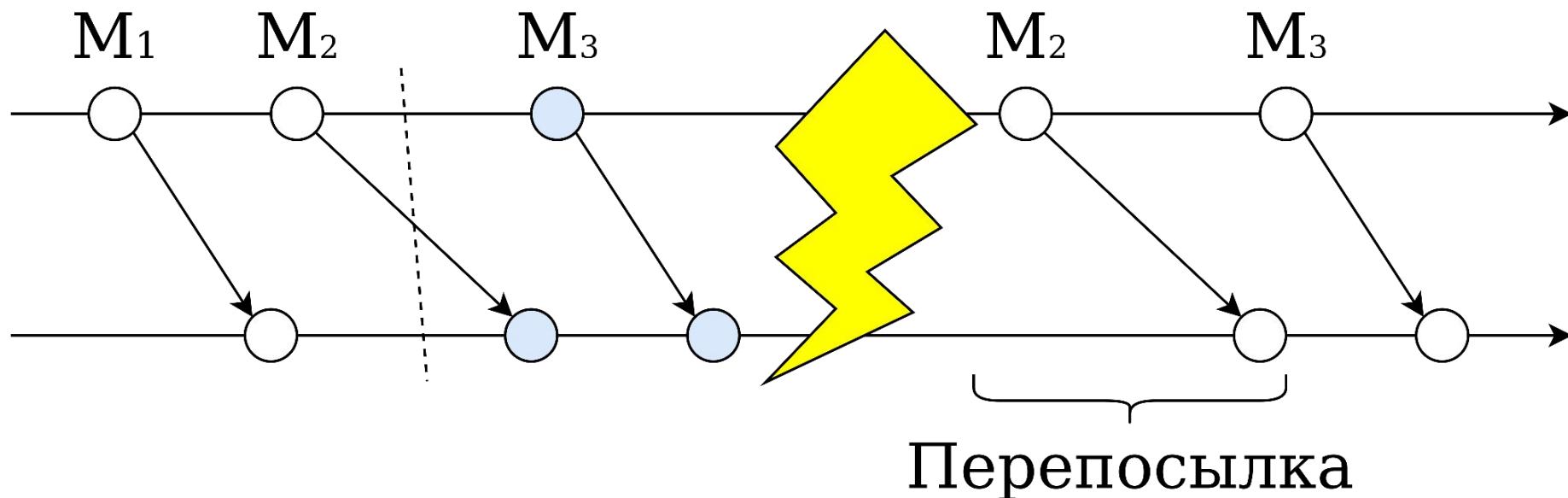
Запоминание на стороне отправителя

- Если сообщение было обработано до обработки токена - Ack придёт после токена
- Иначе - придёт после



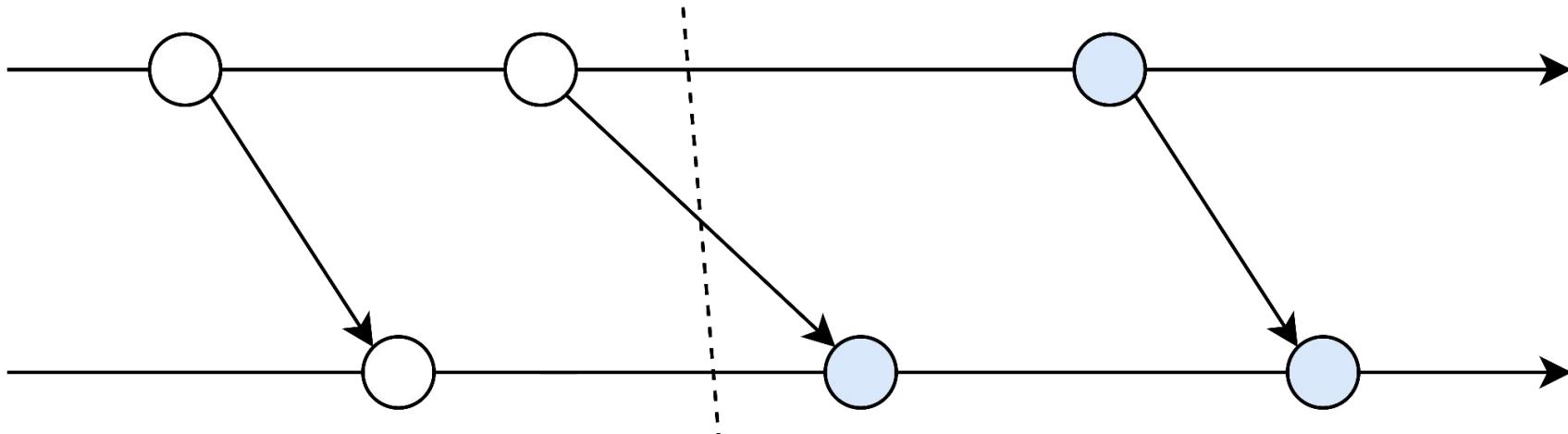
Запоминание на стороне отправителя

- После восстановления отправители перепсылают запомненные сообщения



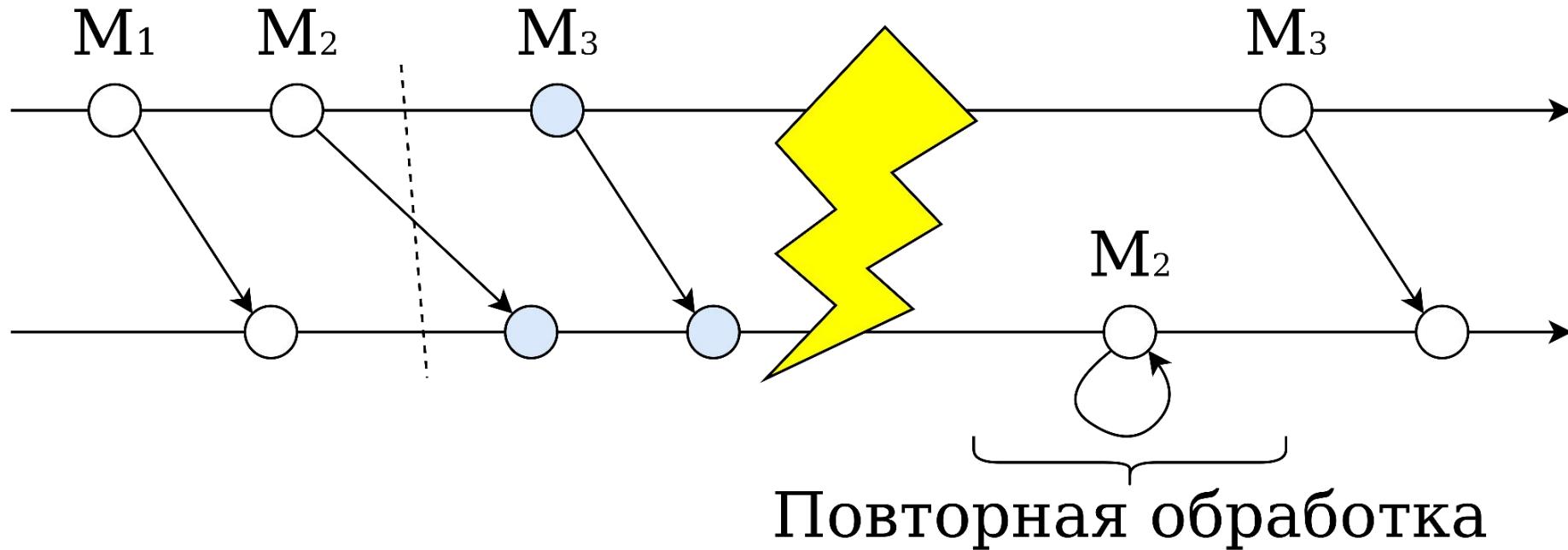
Запоминание на стороне получателя

- К каждому сообщению прикрепляем состояние отправителя
 - Сделал он уже снимок или ещё нет
- Получатель однозначно определяет, нужно ли сохранять состояние



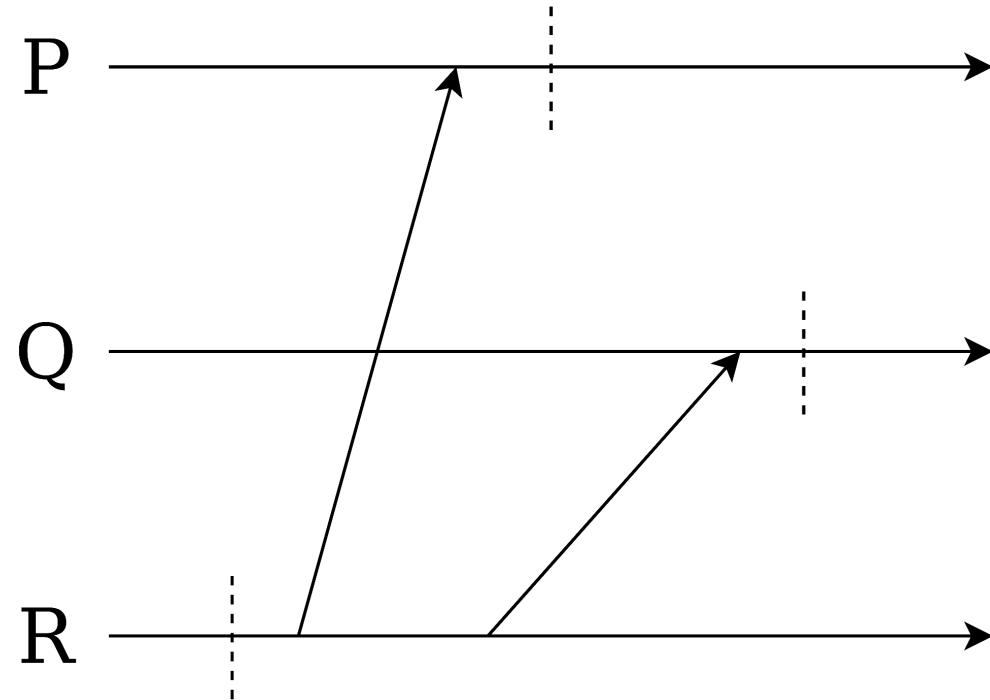
Запоминание на стороне получателя

- После восстановления получатели обрабатывают запомненные сообщения ещё раз



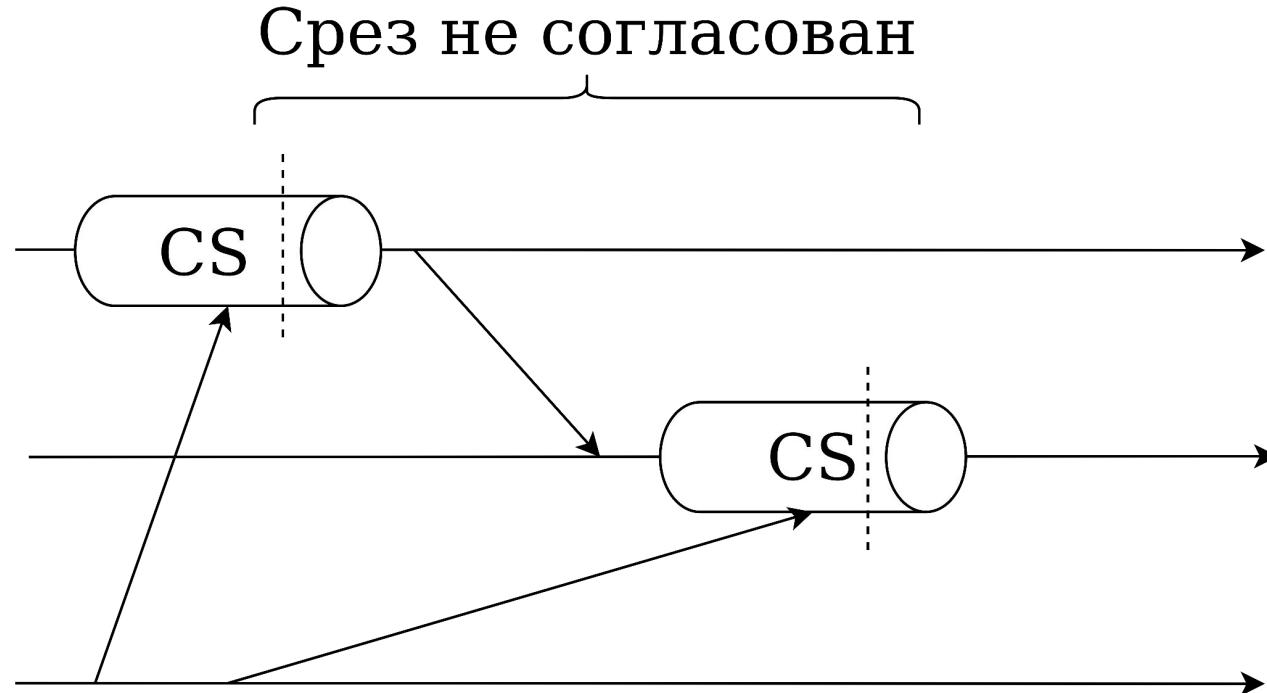
Предикаты

- Хотим уметь проверять что-то про систему
 - Не более одного процесса в КС
 - Токен ровно у одного процесса
 - Нет дедлока
- Можно ли проверять предикат тривиальным алгоритмом?



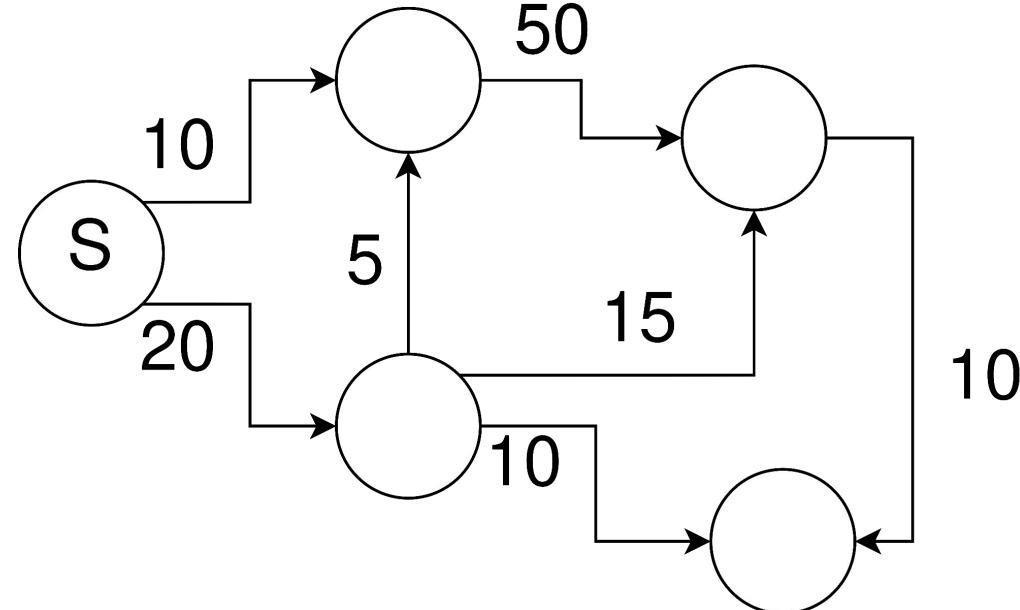
Предикаты

- Только на согласованном срезе
- $O(N^2)$ сообщений. Можно ли дешевле?



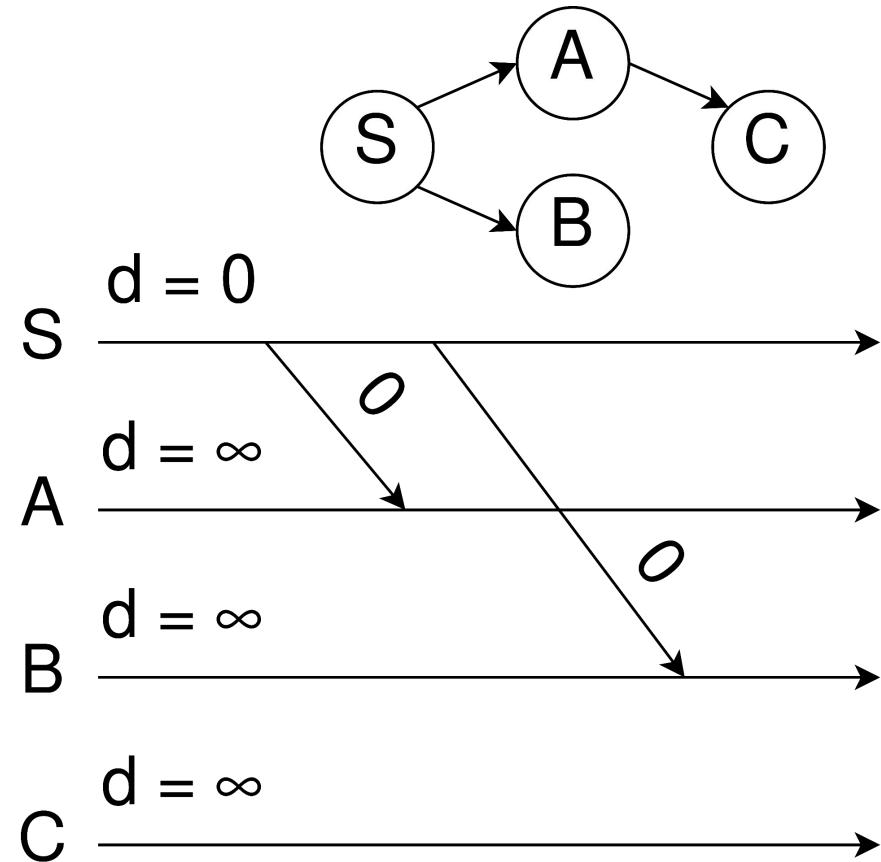
Распределённый алгоритм Дейкстры

- Кратчайший путь в ориентированном взвешенном графе
- Каждому процессу соответствует вершина графа
- От каждого процесса ищем кратчайшее расстояние до процесса-инициатора



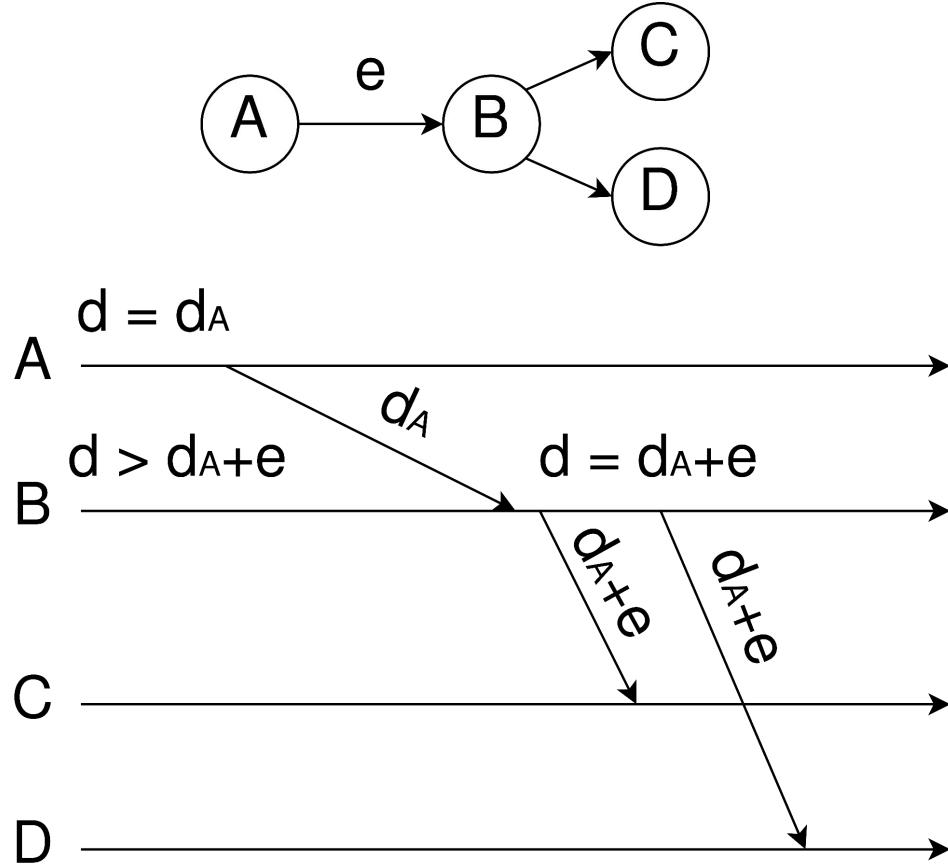
Распределённый алгоритм Дейкстры

- Расстояние от инициатора изначально равно нулю
- От остальных процессов равно бесконечности
- Инициатор рассыпает всем **соседям** сообщение с расстоянием 0
- Похожим образом работает протокол RIP



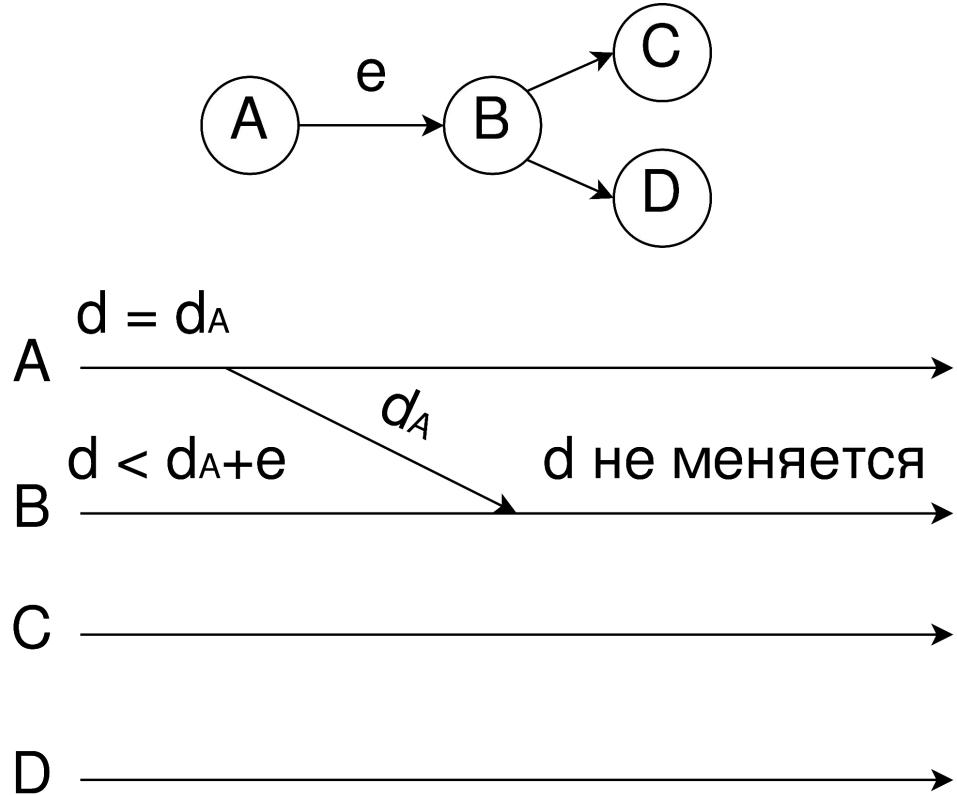
Распределённый алгоритм Дейкстры

- Получив сообщение с расстоянием d_n по ребру весом e , релаксируем
$$d = \min(d, d_n + e)$$
- Если расстояние уменьшилось, рассылаем его всем соседям



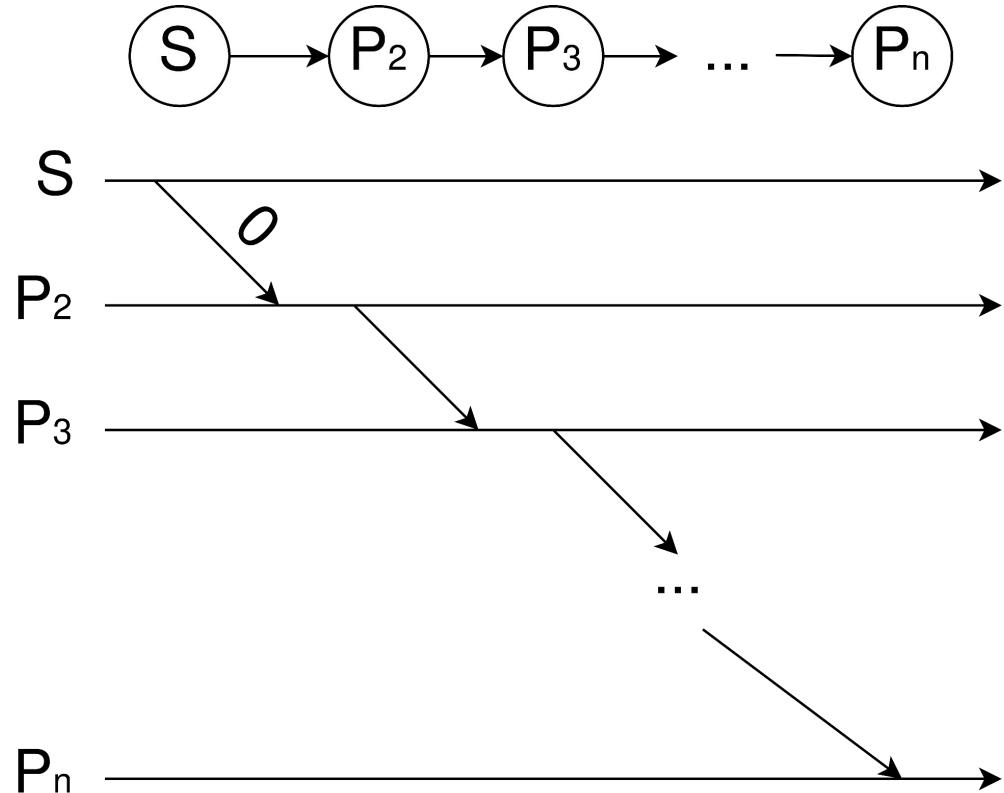
Распределённый алгоритм Дейкстры

- Получив сообщение с расстоянием d_n по ребру весом e , релаксируем
$$d = \min(d, d_n + e)$$
- Если расстояние не уменьшилось, ничего не делаем



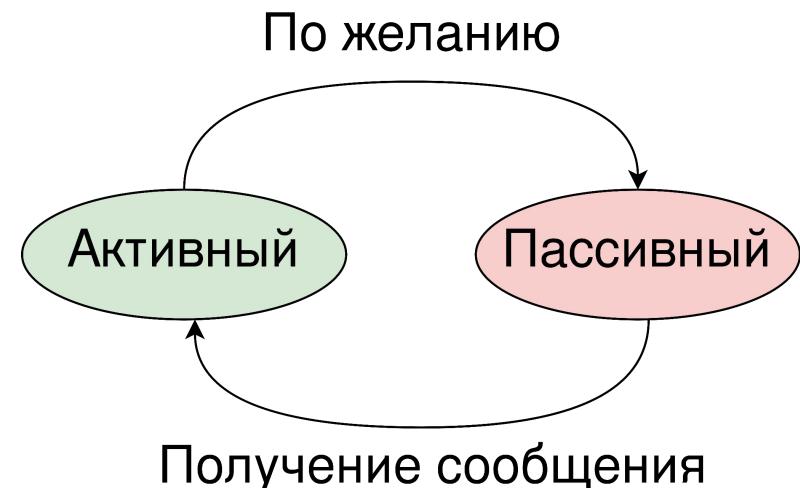
Распределённый алгоритм Дейкстры

- Алгоритм находит все кратчайшие пути
- И завершается
 - Расстояния не релаксируются
 - Значит, никто не рассыпает новое расстояние



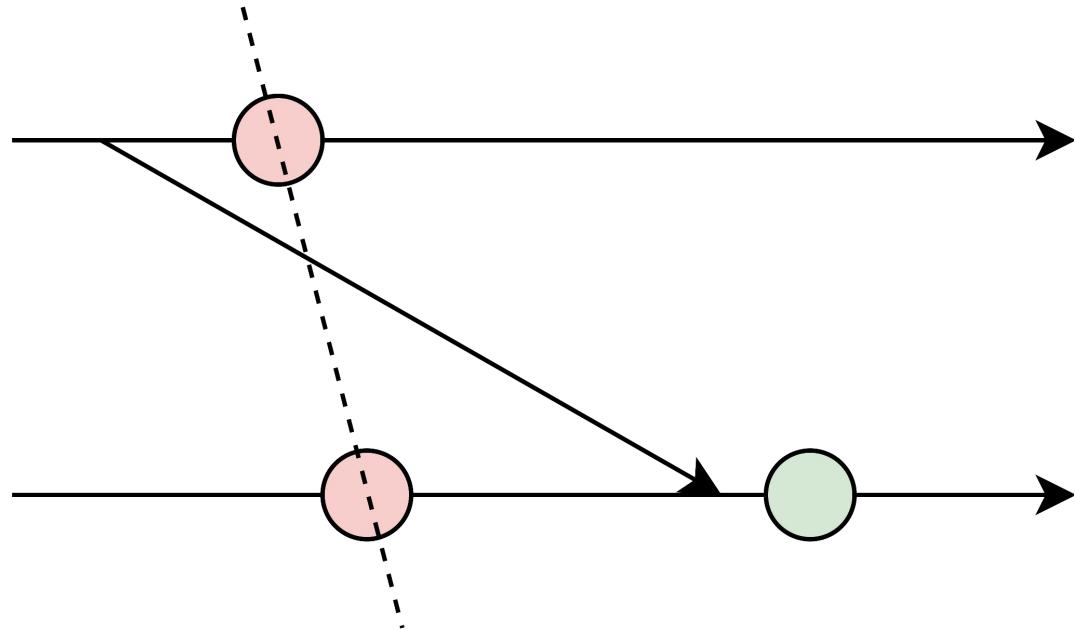
Диффундирующие вычисления

- Каждый процесс может быть либо активен, либо пассивен
- Только активный процесс может посылать сообщения
- Пассивный процесс становится активным, получив сообщение
- Активный процесс может в любой момент стать пассивным
- Изначально есть один активный процесс — инициатор



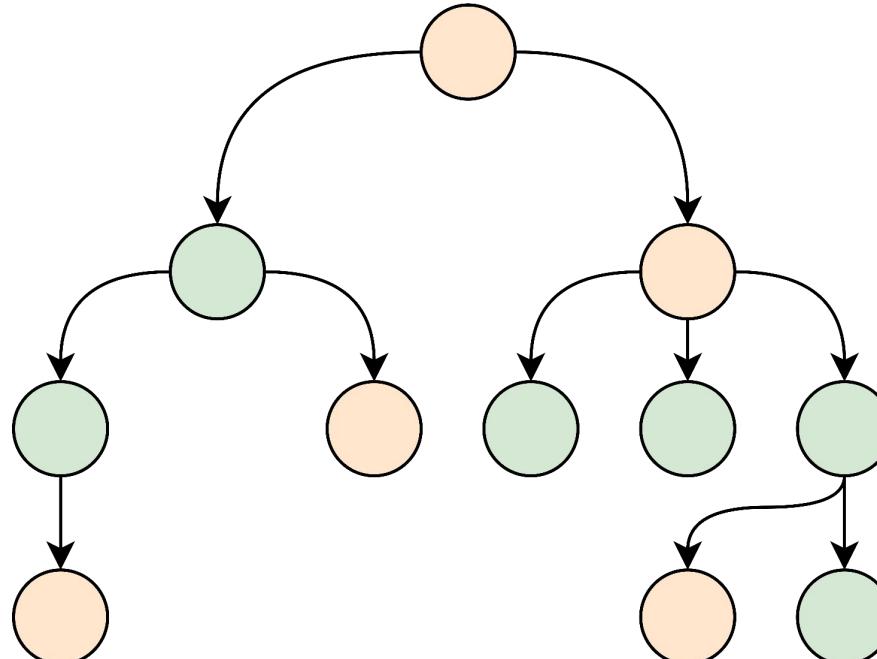
Диффундирующие вычисления

- Вычисление завершилось когда все процессы пассивны
- И нет сообщений в пути
- В противном случае даже согласованность среза не гарантирует нам останов вычисления



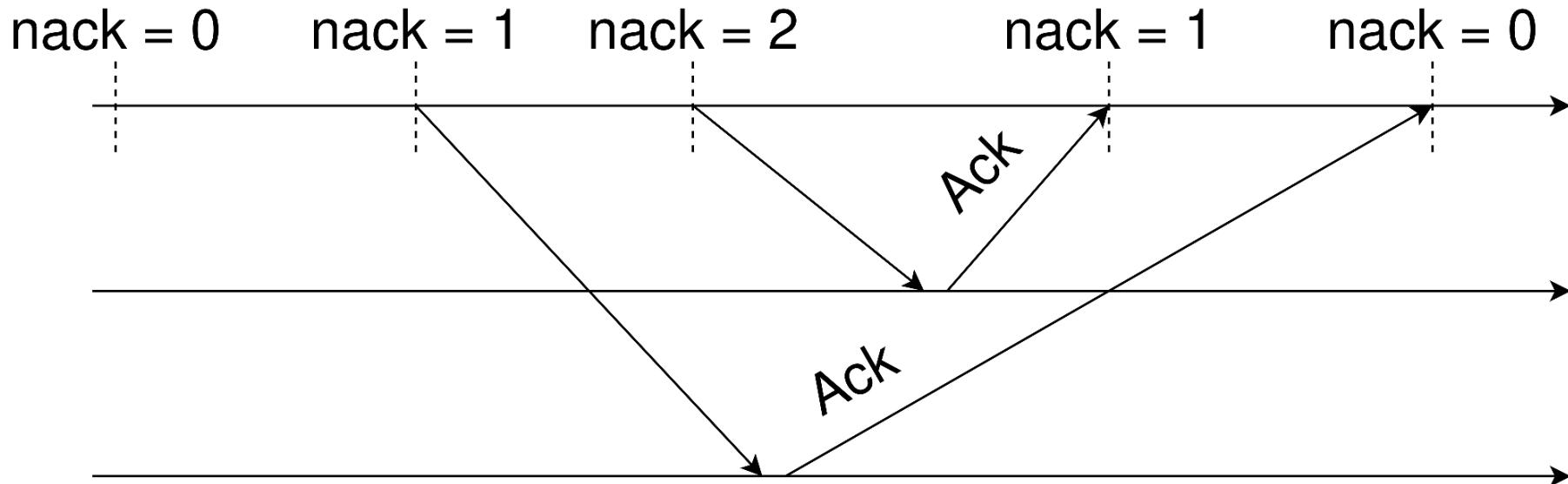
Алгоритм Дейкстры и Шолтена

- Выстраиваем некоторые процессы в дерево
- В дереве могут быть активные и пассивные процессы
- Остальные процессы свалены в кучу
- Только пассивные



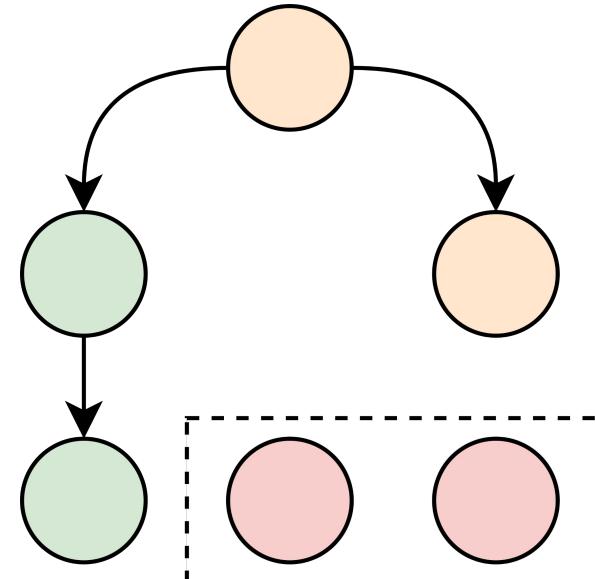
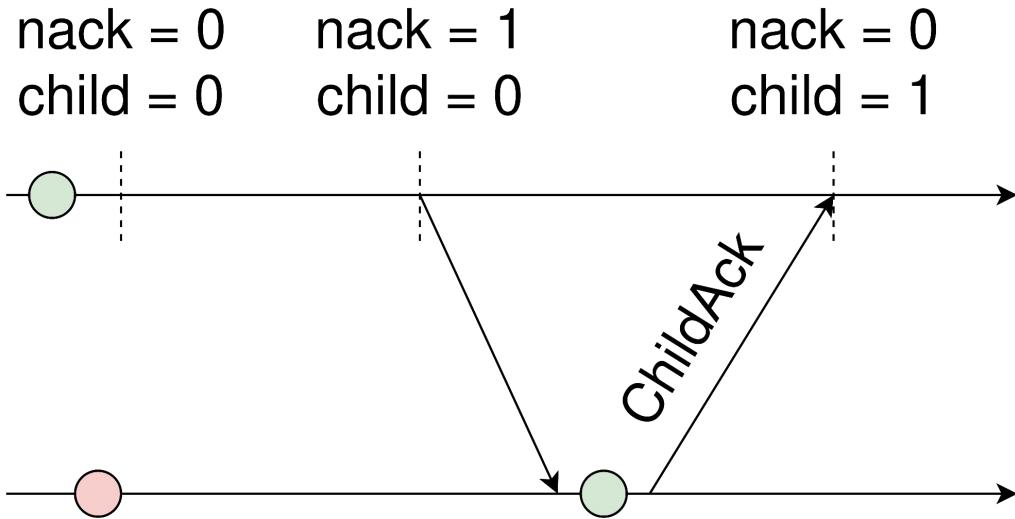
Алгоритм Дейкстры и Шолтена

- На каждое сообщение высылается подтверждение
- Каждый активный процесс считает количество неподтверждённых сообщений



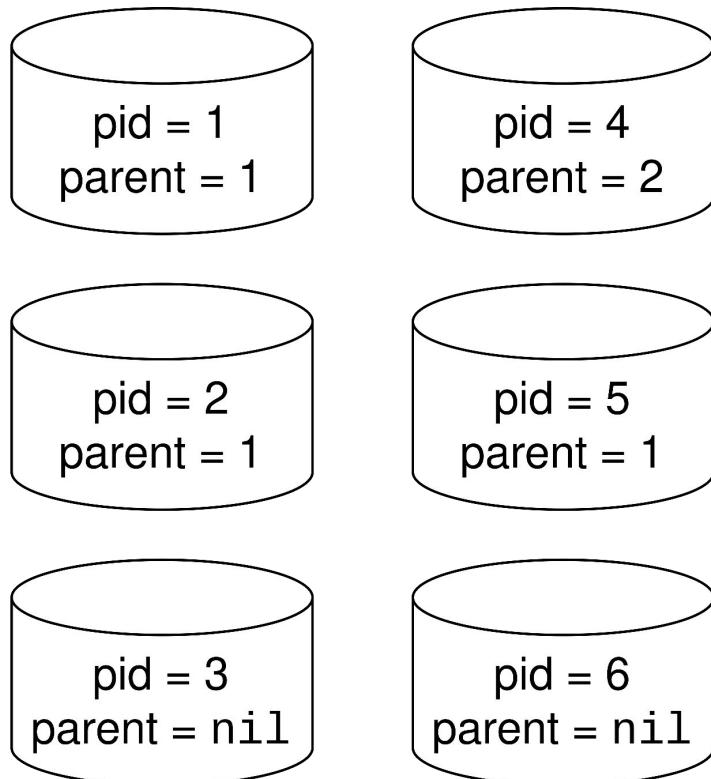
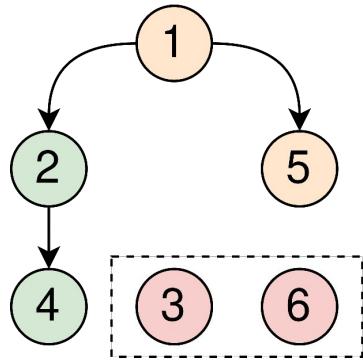
Алгоритм Дейкстры и Шолтена

- Если процесс из кучи получает сообщение, он становится ребёнком отправителя в дереве
- Вместо Ack посылает сообщение ChildAck



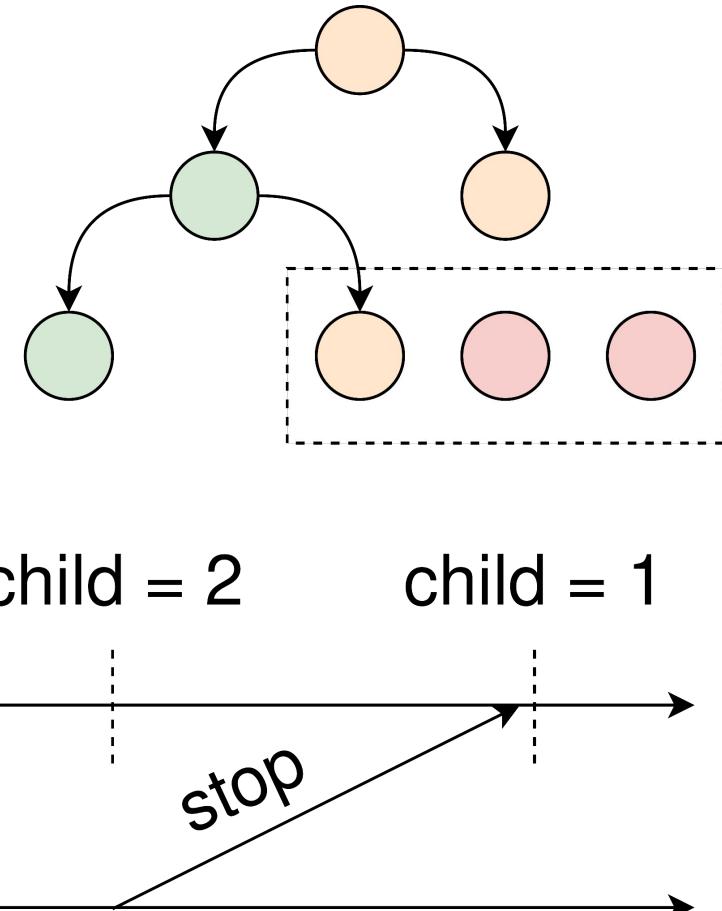
Алгоритм Дейкстры и Шолтена

- Дерево нигде не хранится централизованно
- Просто каждый процесс помнит своего родителя



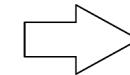
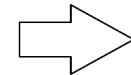
Алгоритм Дейкстры и Шолтена

- Процесс может выйти из дерева если он:
 - Пассивен
 - Не имеет детей
 - Не имеет неподтверждённых сообщений
- Посыпает родителю сообщение ChildStop
- Родитель уменьшает число детей

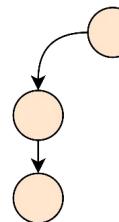
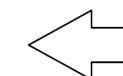
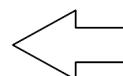
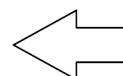
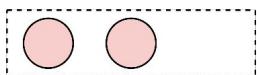
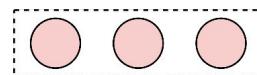
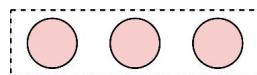


Алгоритм Дейкстры и Шолтена

- Изначально в дереве находится только корень (Процесс-инициатор)

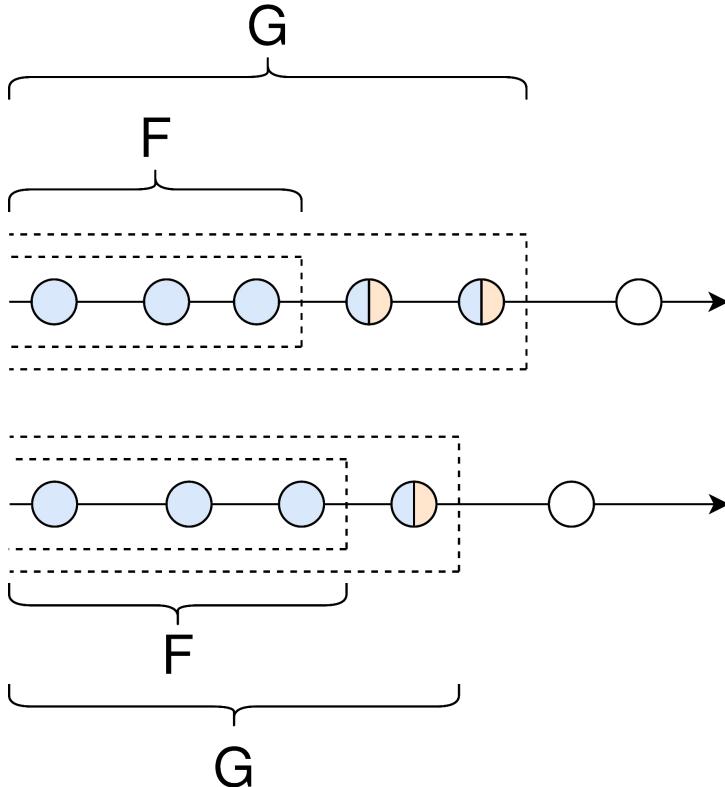


- Алгоритм завершается как только корень выходит из дерева



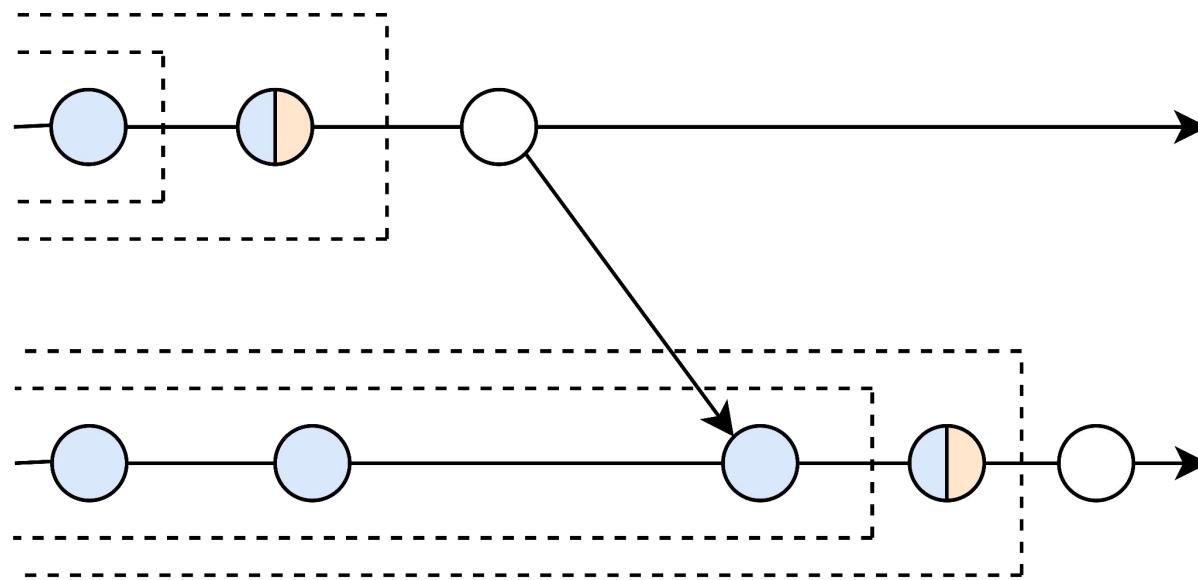
Согласованные интервалы

- Пара срезов $F, G \subset E$ называется интервалом, если $F \subset G$
- Обозначаем $[F, G]$



Согласованные интервалы

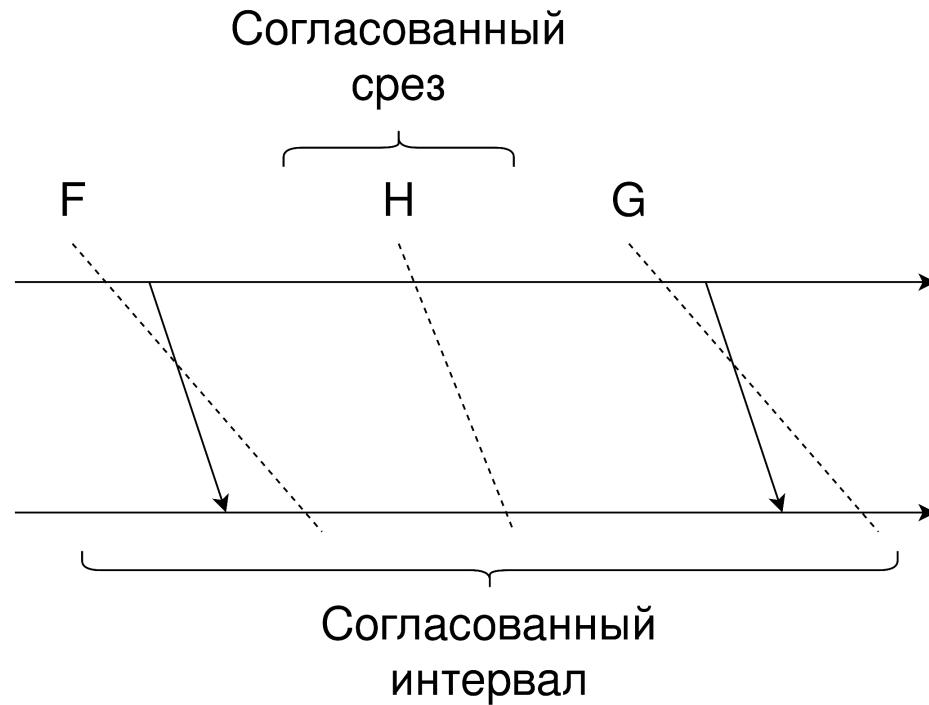
- Интервал $[F, G]$ называется согласованным, если
$$\forall x \in E, y \in F : x \rightarrow y \Rightarrow x \in G$$
- Эквивалентно $\nexists x \in E \setminus G, y \in F : x \rightarrow y$



Согласованные интервалы

- **Теорема:** интервал $[F, G]$ согласован тогда и только тогда, когда внутри него существует согласованный срез H

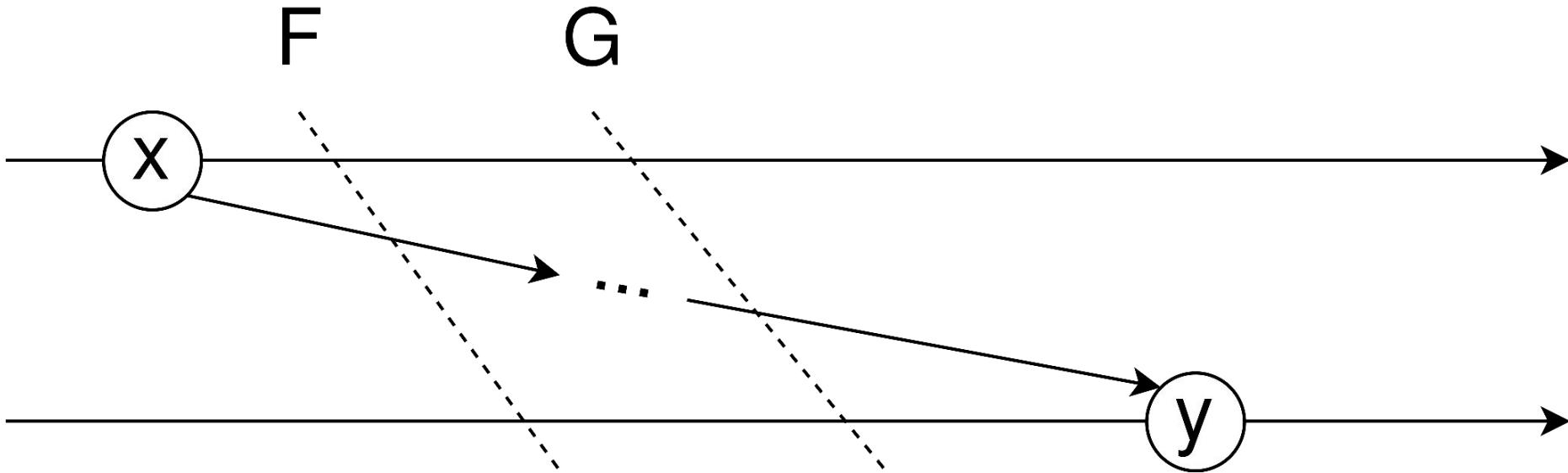
$$F \subset H \subset G$$



Согласованные интервалы

- Интервал $[F, G]$ барьерно синхронизирован, если

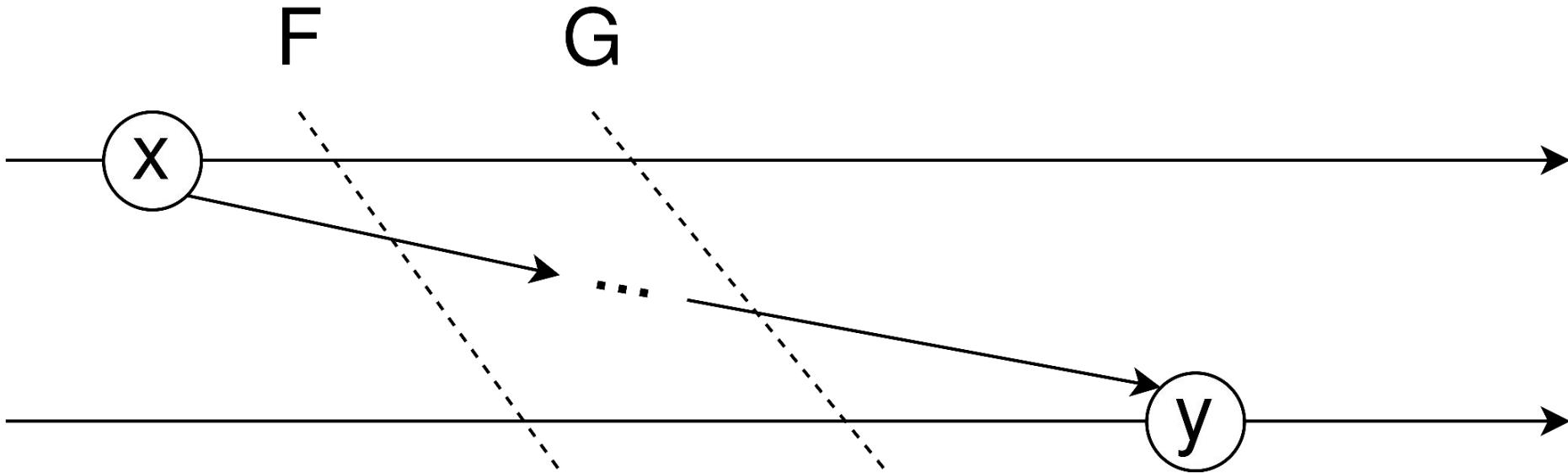
$$\forall x \in F, y \in E \setminus G : x \rightarrow y$$



Согласованные интервалы

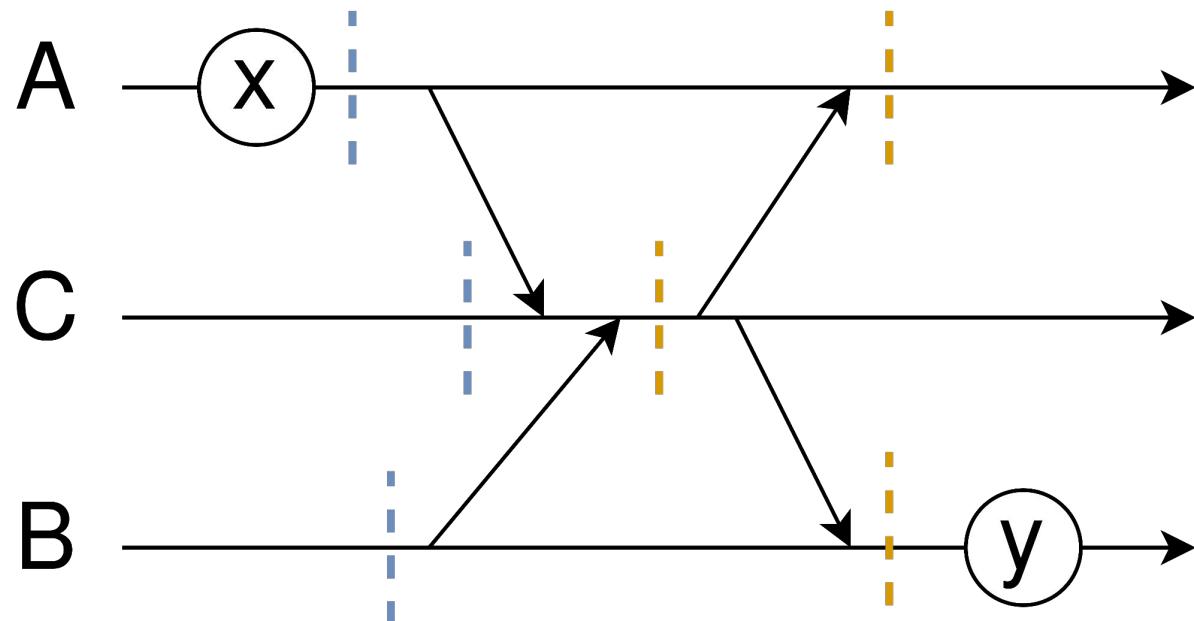
- Любой барьерно синхронизированный интервал согласован

$$(\forall x \in F, y \in E \setminus G : x \rightarrow y) \Rightarrow (\exists y \in E \setminus G, x \in F : y \rightarrow x)$$



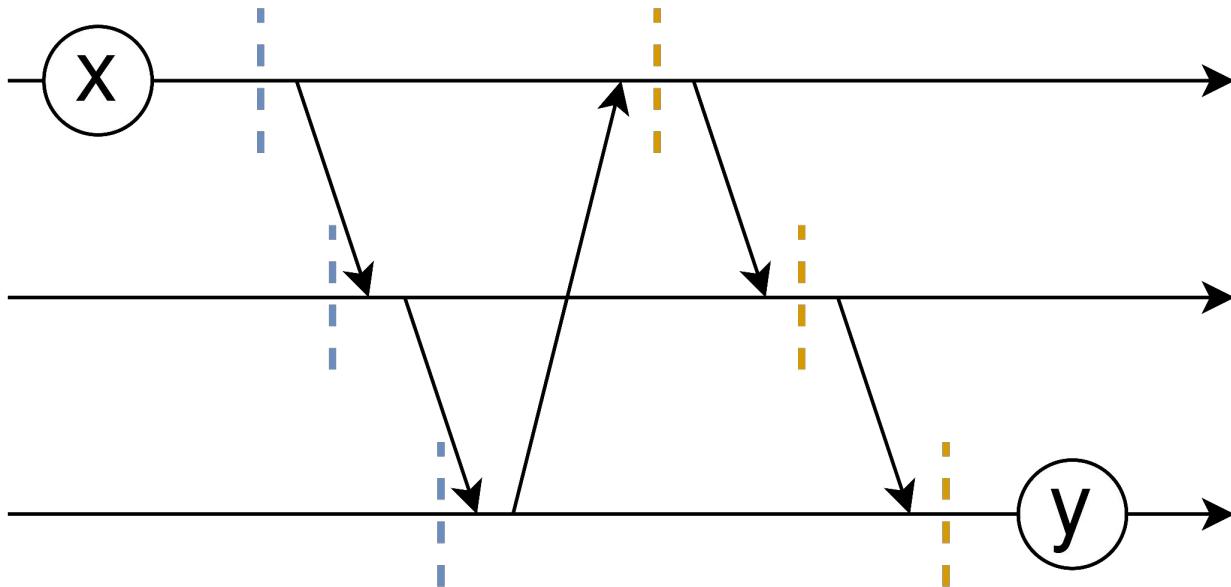
Согласованные интервалы

- Строим барьерно синхронизированный интервал через координатора
- 2 ($N - 1$) сообщение
- Может ли координатор отвечать сразу, не дожидаясь сообщений от всех?



Согласованные интервалы

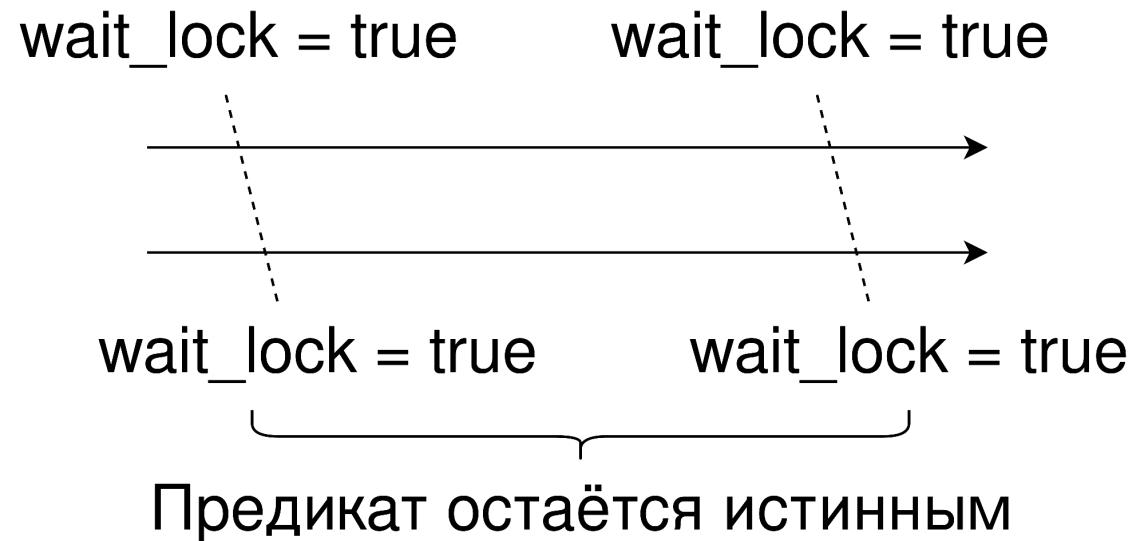
- Посыпаем токен по кругу два раза
- $2(N - 1)$ сообщение



Локально стабильные предикаты

- Став однажды истинным, остаётся истинным всегда
- Определяется группой процессов, состояние которых не меняется
- Дедлок
- Окончание вычисления

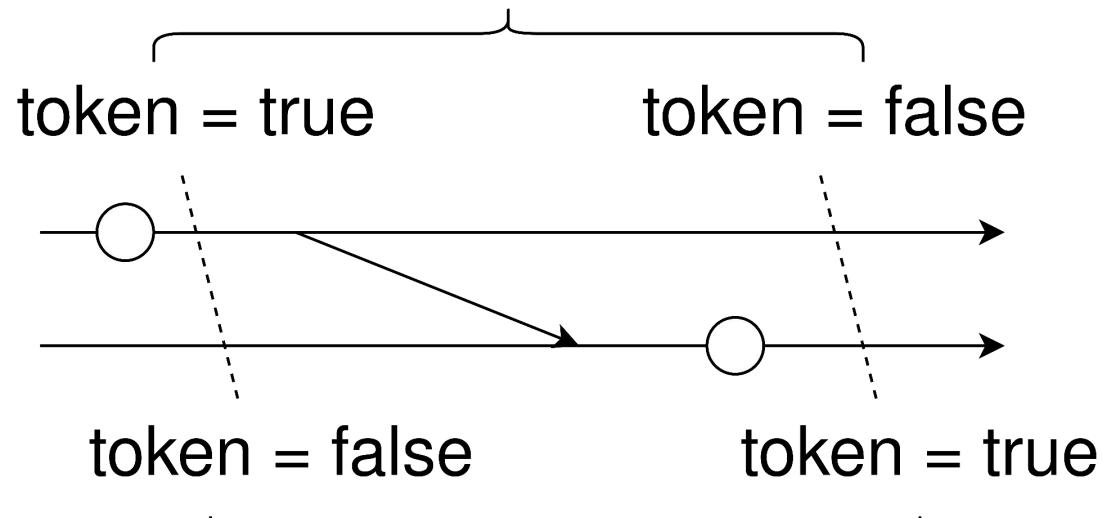
Состояние процессов
не меняется



Локально стабильные предикаты

- Наличие в системе токена это не локально стабильный предикат

Состояние процессов меняется

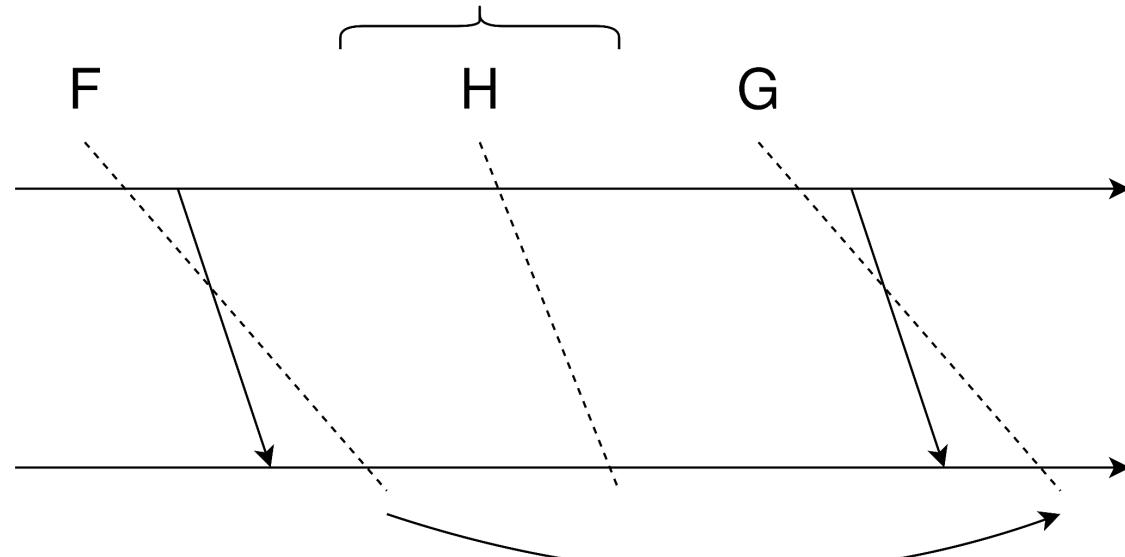


Предикат остаётся истинным

Локально стабильные предикаты

- Строим согласованный интервал за $O(N)$ сообщений
- Если на F предикат выполнен
- И состояние не менялось до G
- То предикат выполнен на H

Согласованный срез



Предикат выполнен,
состояние не менялось

Что почитать:

- *Chandy K. M., Lamport L.* Distributed snapshots: Determining global states of distributed systems
- *Dijkstra E. W., Scholten C. S.* Termination detection for diffusing computations
- *Atreya R. et al.* Efficient detection of a locally stable predicate in a distributed system

Thanks for your attention

