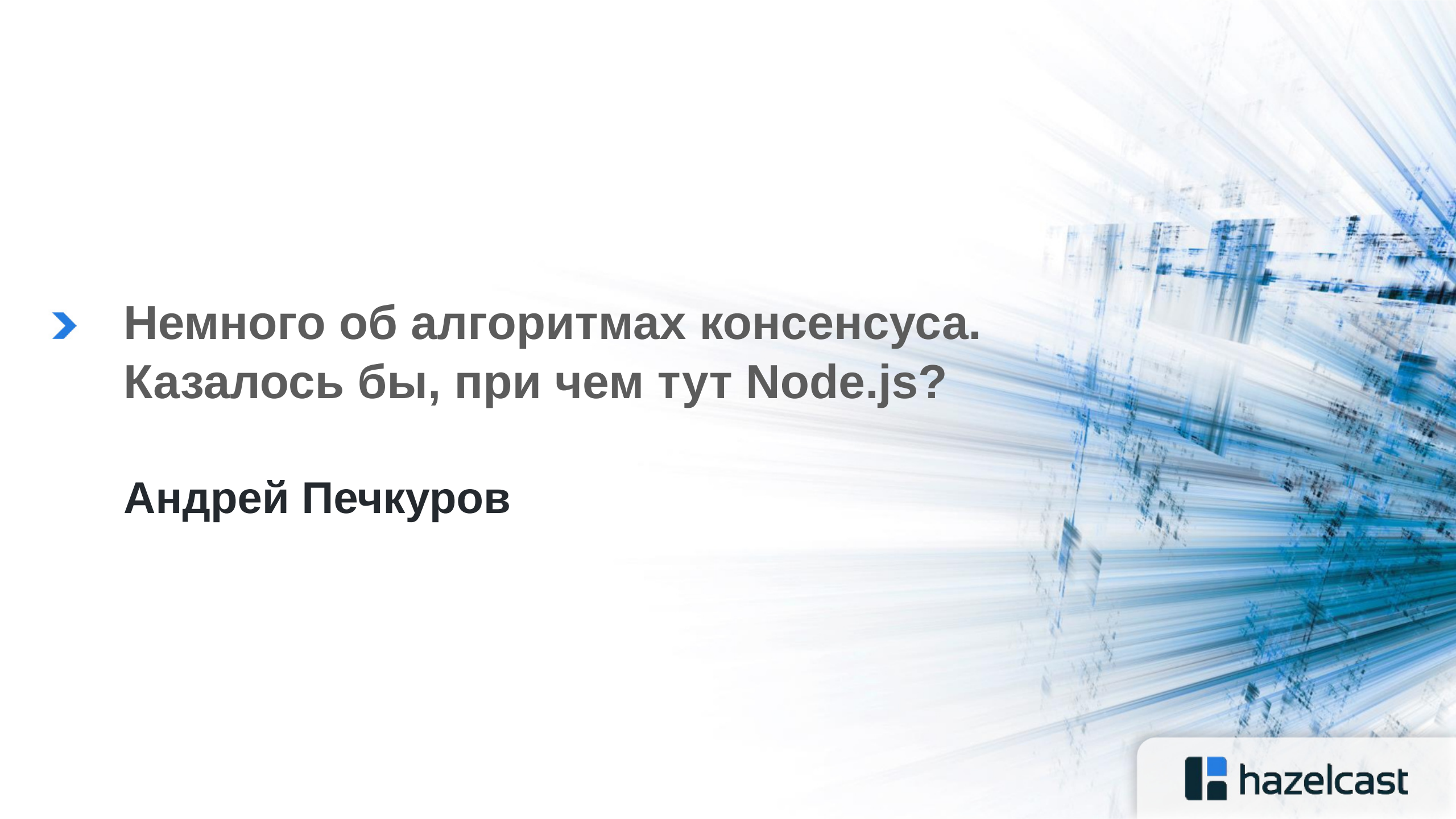
Немного об алгоритмах консенсуса. Казалось бы, при чем тут Node.js?

Андрей Печкуров



О докладчике

- Пишу на Java (очень долго), Node.js (долго)
- Node.js core collaborator
- Интересы: веб, архитектура, распределенные системы, производительность
- Можно найти тут:
  - https://twitter.com/AndreyPechkurov
  - https://github.com/puzpuzpuz
  - https://medium.com/@apechkurov

- Hazelcast In-Memory Data Grid (IMDG)
- Большой набор распределенных структур данных
- Показательный пример - `Map`, который часто используют как кэш
- Написана на Java, умеет embedded и standalone режимы
- Хорошо масштабируется вертикально и горизонтально
- Часто используется в high-load и low-latency приложениях
- Области применения: IoT, in-memory stream processing, payment processing, fraud detection и т.д.

- Hazelcast In-Memory Data Grid (IMDG)
- Хотите production-ready Raft? У нас есть CP Subsystem (с Jepsen тестами и локами 🙂)
- https://docs.hazelcast.org/docs/4.0.1/manual/html-single/index.html#cp-subsystem

Hazelcast IMDG Node.js client

- https://github.com/hazelcast/hazelcast-nodejs-client
- Доклад про историю оптимизаций
  - Видео: https://youtu.be/CSnmpbZsVD4
  - Слайды: https://github.com/puzpuzpuz/talks/tree/master/2019-ru-nodejs-library-optimization
- P.S. Поддержки CP Subsystem в этом клиенте пока нет, но она скоро будет

План на сегодня

- Начинаем пугаться распределенных систем
- Знакомимся с видами согласованности (consistency)
- CAP теорема и прочие классификации
- Что за зверь - алгоритм консенсуса?
- История: Paxos и его подвиды, Raft
- CASPaxos, как один из недавних Paxos-образных
- Pet project: CASPaxos на Node.js

- **Начинаем пугаться распределенных систем**

Распределенная система

- Назовем распределенной систему, хранящую состояние (общее) на нескольких машинах, соединенных сетью
- Для определенности будем подразумевать хранилище пар ключ-значение

Упрощенная до ужаса история

- Традиционно были РСУБД на бооольших, дорогих железках
- Однако, в 80-90х уже были академический интерес к распределенным системам
- В начале 2000х некоторые компании (намек на Google) сделали ставку на доступное железо и распределенные системы
- Основной бум пришелся на 2010е годы

Два мира

Централизованная система	**Распределенная система**
Вертикальное масштабирование	Горизонтальное масштабирование
Локальные вызовы	Сетевые вызовы
< p отказа машины	> p отказа машины
> критичность отказа	< критичность отказа

Fallacies of distributed computing

- Инженеры из Sun (R.I.P.) сформулировали список заблуждений (1994):
  - The network is reliable
  - Latency is zero
  - Bandwidth is infinite
  - The network is secure
  - Topology doesn't change
  - There is one administrator
  - Transport cost is zero
  - The network is homogeneous
- P.S. Добавим сюда "Clocks are in sync"

Сеть

- Мы работаем с асинхронными сетями
- Отправленный запрос может:
  - Быть потерян при отправке туда/обратно
  - Находиться в очереди ожидания отправки (если сеть под нагрузкой)
  - Быть получен, но машина-адресат дала сбой до или во время обработки
- Единственный способ подтверждение - получить ответ

Часы

- Глобальные (синхронизированные) часы невозможны без специального железа (например, GPS, но и тут есть нюансы)
- Монотонные часы, впрочем, доступны, но не помогут, например, при разрешении коллизий
- Байка: в Google Spanner часы в ЦОД синхронизированы в пределах 7 мс

Чего мы ждем от распределенной системы?

- Масштабируемость
- Отказоустойчивость
- Удобство поддержки (мониторинг, администрирование)

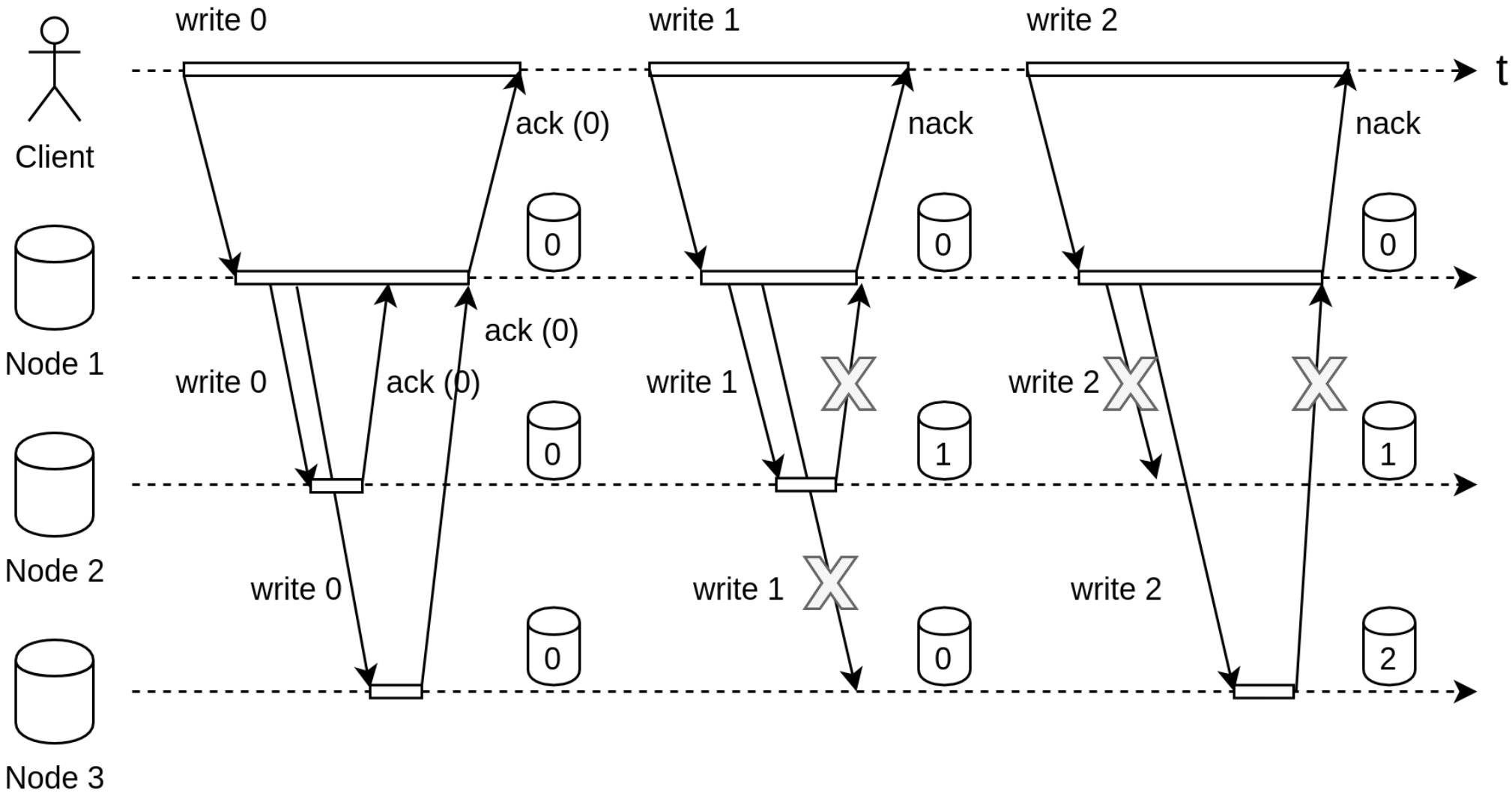
Чего мы еще ждем?

- Допустим, что от распределенного хранилища данных мы ждем того же поведения, что и от централизованного
- А именно - с клиентской стороны поведение должно быть, как если бы это была централизованная система (пока остановимся на этой формулировке)

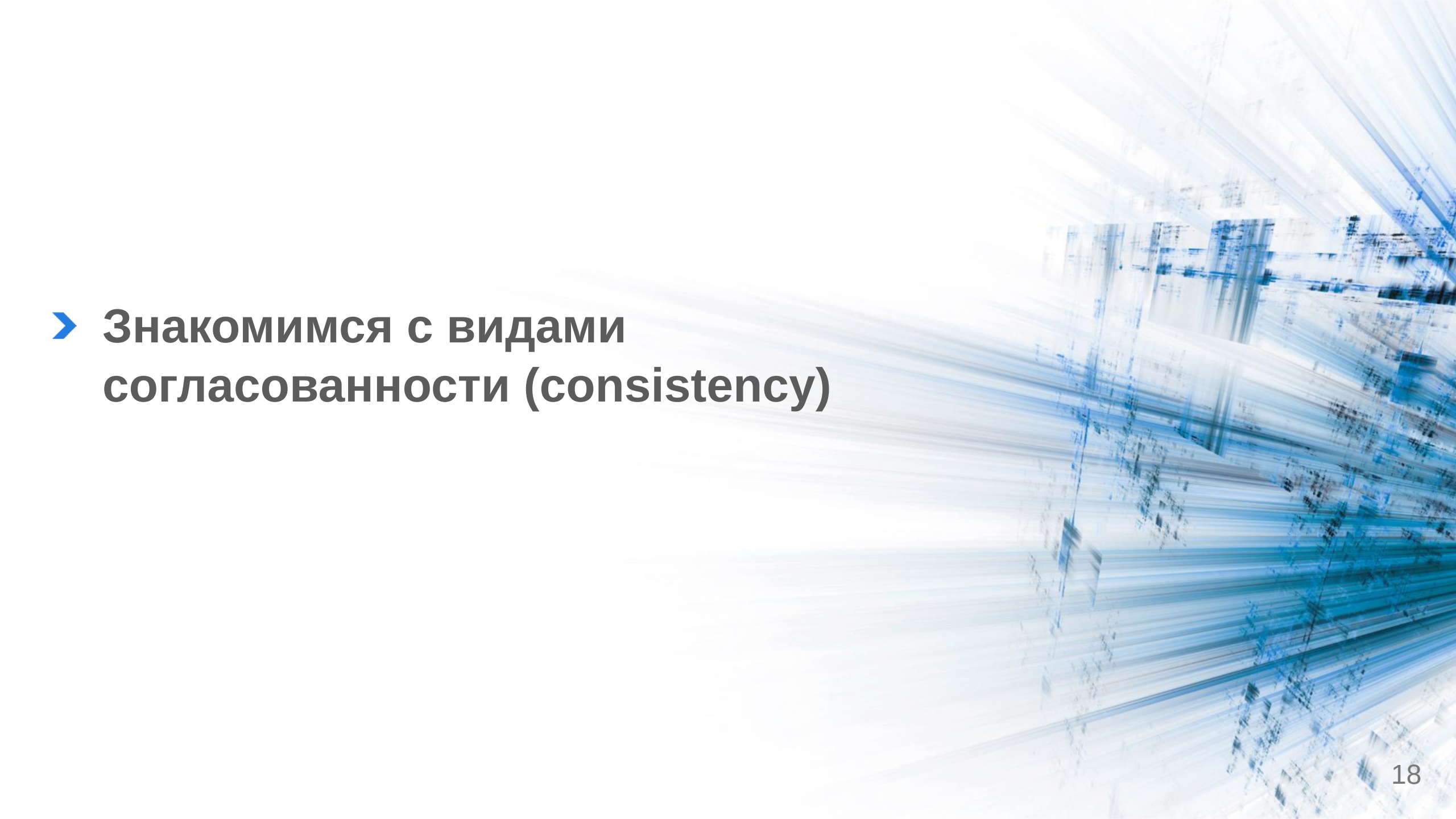
Фигня вопрос - сейчас придумаем алгоритм

1. Любой узел принимает клиентские запросы (прочитать/записать)
2. Затем - отправляет операцию на все остальные узлы
3. Ждет ответов от большинства (консенсус жеж 😄)
4. Дождавшись консенсуса, отправляет клиенту сообщение об успехе

Что не так с нашим изобретением?



- **Знакомимся с видами согласованности (consistency)**



Неформальное определение

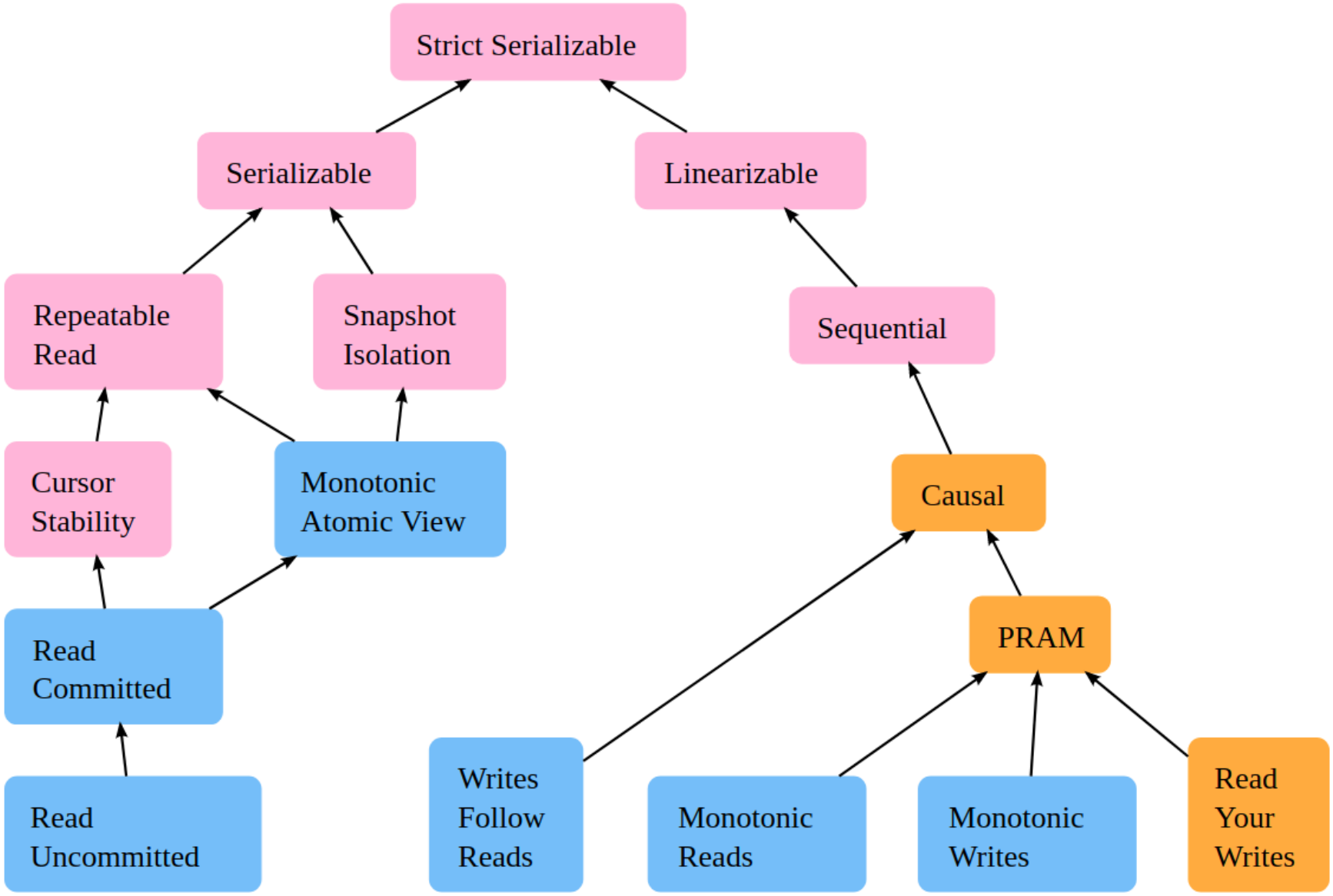
- Модель согласованности (consitency model) - это гарантии, которые система (внезапно, не только распределенная) предоставляет относительно набора поддерживамых операций

Eventual consistency

- Система гарантирует, что данные будут доступны на всех узлах через какое-то (неопределенное) время после завершения операций записи
- Это слабая модель, с точки зрения гарантий

Monotonic reads

- Система гарантирует, что если какой-то (конкретный) клиент читает запись, последовательные чтения той же записи вернут то же самое, или более познее значение



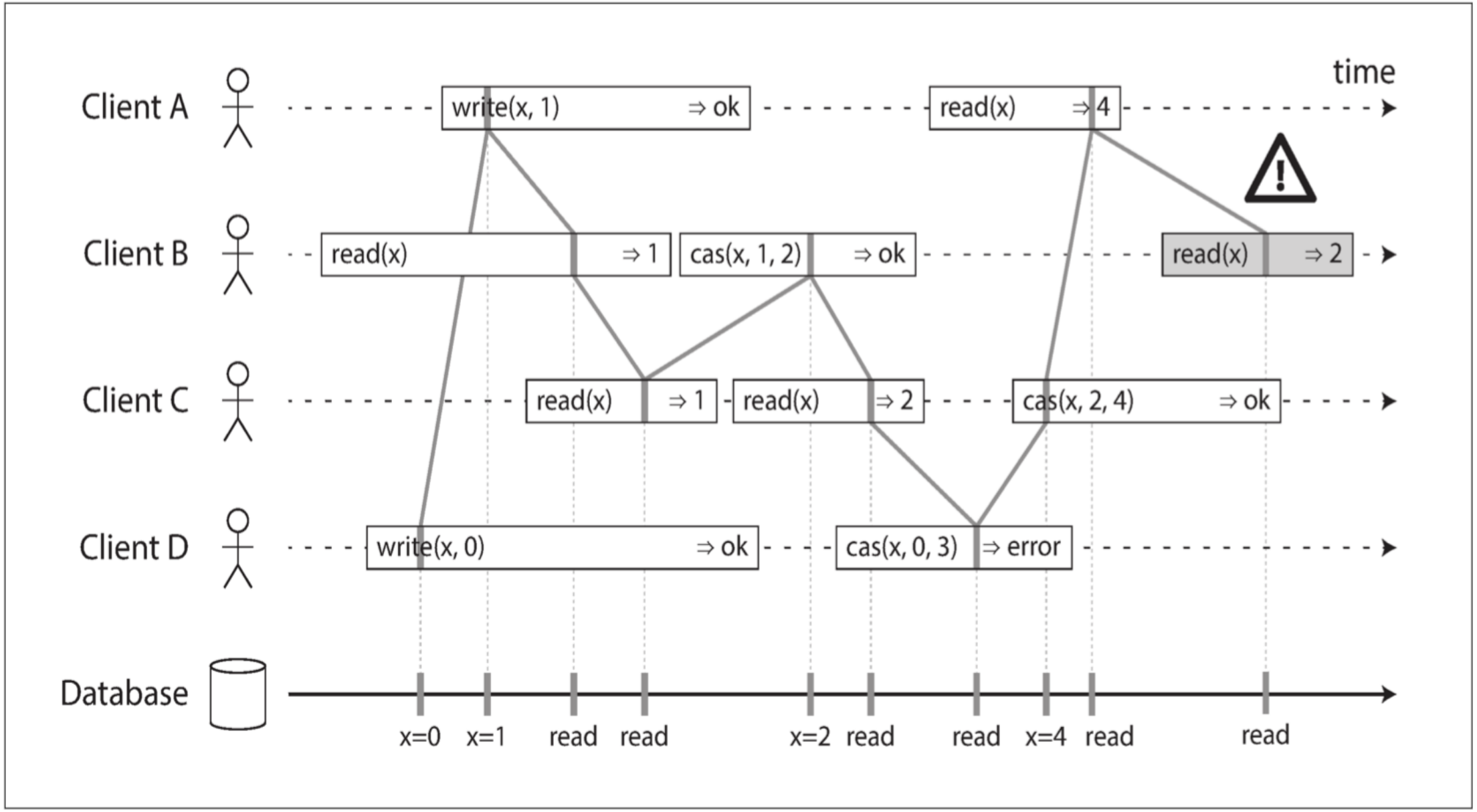
Source: https://jepsen.io/consistency

Linearizability

- Одна из наиболее строгих (сильных) моделей согласованности для **одного** объекта
- Именно ее мы подразумевали (надеюсь ранее):
  "с клиентской стороны поведение должно быть, как если бы это была централизованная система"

Неформальное определедение linearizability

- Система гарантирует, что каждая операция выполняется атомарно, в **некотором** (общем для всех клиентов) порядке, не противоречащим порядку выполнения операций в реальном времени
- Т.е. если операция A завершается до начала операции B, то B должна учитывать результат выполнения операции A



Source: DDIA book

- CAP теорема и прочие классификации

CAP теорема

- Сформулирована Eric Brewer в 1998, как утверждение
- В 2002 появилось формальное доказательство
- Рассматривается система с одним регистром
- CAP:
  - Consistency: здесь подразумевается линеаризуемость
  - Availability: каждый запрос, полученный нормально функционирующим узлом системы, должен приводить к ожидаемому ответу (не к ошибке)
  - Partition tolerance: подразумевает коммуникацию через асинхронную сеть

Partition tolerance

Network partition - сценарий, когда узлы продолжают функционировать, но некоторые из них не могут общаться между собой

Неформальное определение

В условиях network partition рассматриваемая система может быть:

- Доступна (AP)
- Согласована (CP)

P.S. CA опции в CAP теореме нет и в помине

Критика

- Однобокая классификация, которая почему-то прижилась
- Например, РУСБД с одной read-only репликой не является ни CP, ни AP
- Теорема ничего не говорит о latency системы, т.е. AP система может отвечать сколь угодно медленно
- К тому же, network partition - далеко не единственный сценарий отказа

Альтернативы

- PACELC теорема - расширение CAP теоремы (Daniel J. Abadi, 2010)
- PAC = PA | PC (та же CAP теорема)
- ELC = EL | EC:
  - E: else
  - L: latency
  - C: consistency

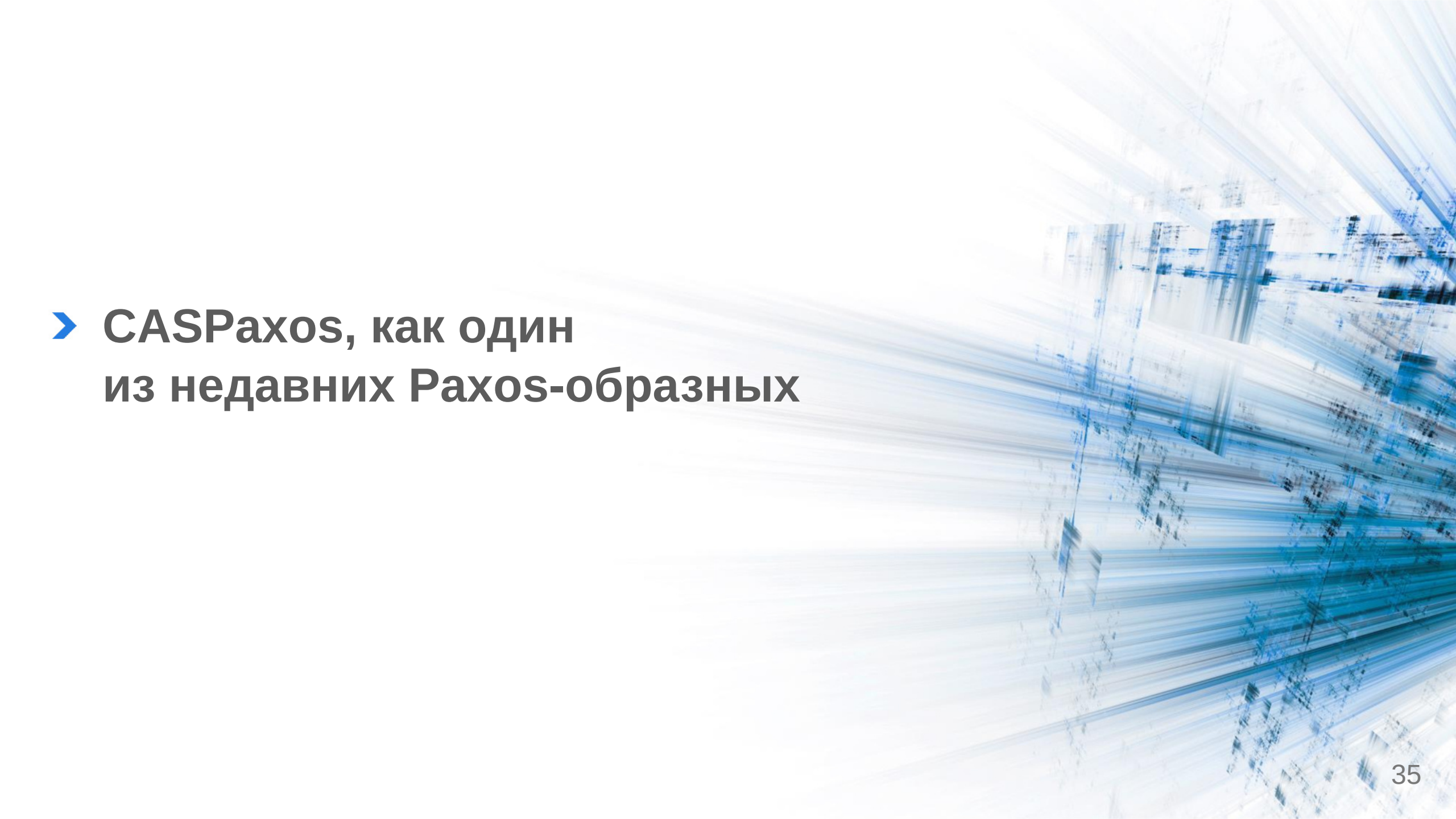
- Что за зверь - алгоритм консенсуса?

Неформальное определение

- Алгоритм консенсуса - алгоритм, позволяющий узлам системы достигнуть консенсус, т.е. принять совместное решение **о том или ином** действии
- Можно показать, что linearizability и алгоритмы консенсуса можно свести друг к другу
- А значит, мы говорим о CP системах, с точки зрения пресловутой CAP теоремы

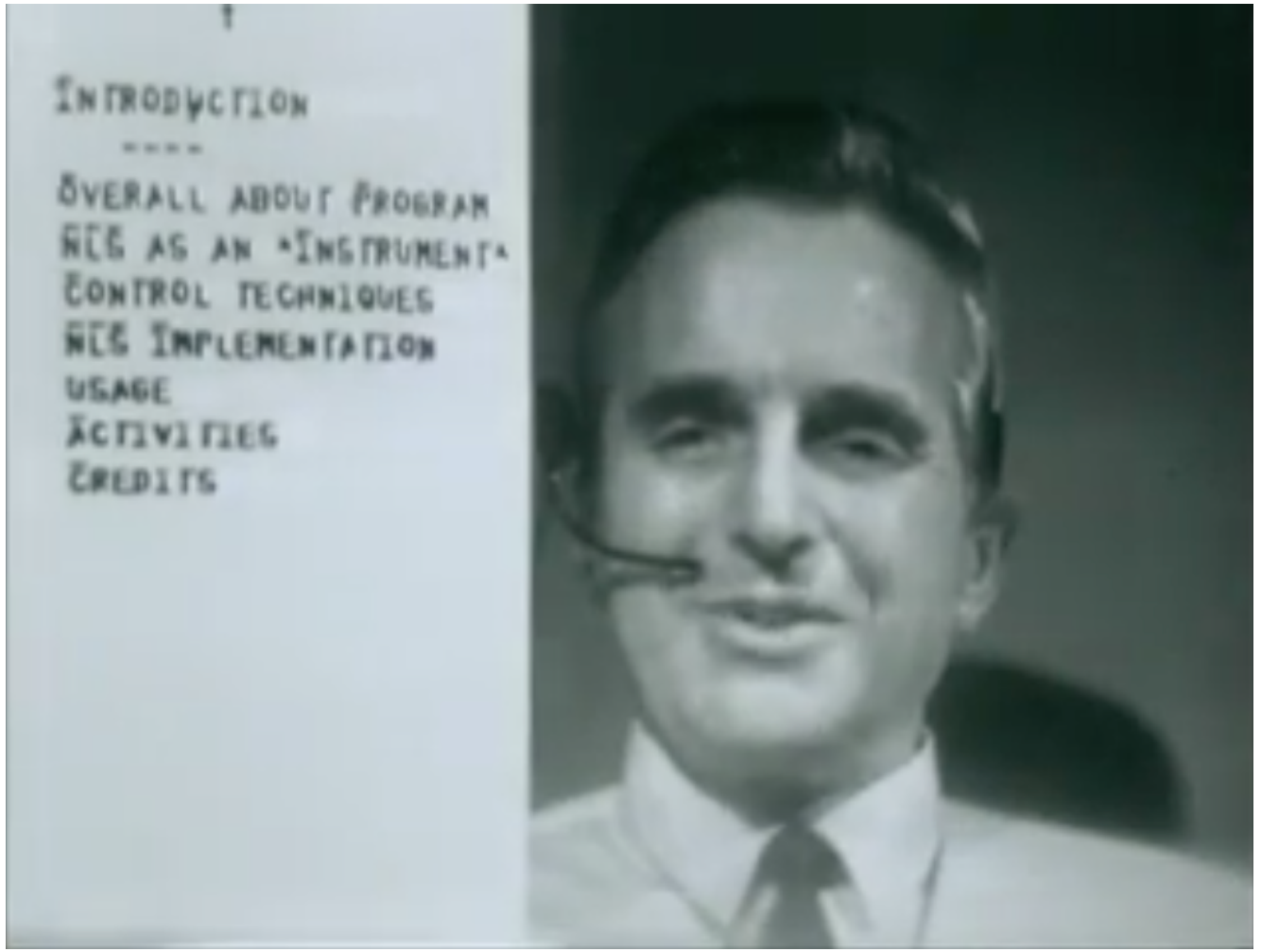
- История: Paxos и его подвиды, Raft

- **CASPaxos, как один из недавних Paxos-образных**



- **Pet project: CASPaxos на Node.js**

Демо (если это можно так назвать)



Call to Action

- Распределенных систем бояться - на server-side не ходить
- Все, кому интересны высокопроизводительные библиотеки (и распределенные системы) - welcome
- https://github.com/hazelcast/hazelcast-nodejs-client
- P.S. Contributions are welcome as well

Спасибо за внимание!

Полезные книги и ссылки

- Designing Data-Intensive Applications, Martin Kleppmann, 2017
- CASPaxos: Replicated State Machines without logs, Denis Rystsov, 2018 - https://arxiv.org/abs/1802.07000
- https://raft.github.io/
- https://jepsen.io
- https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html