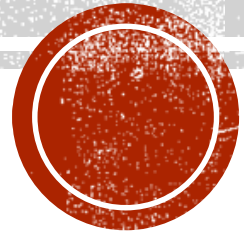
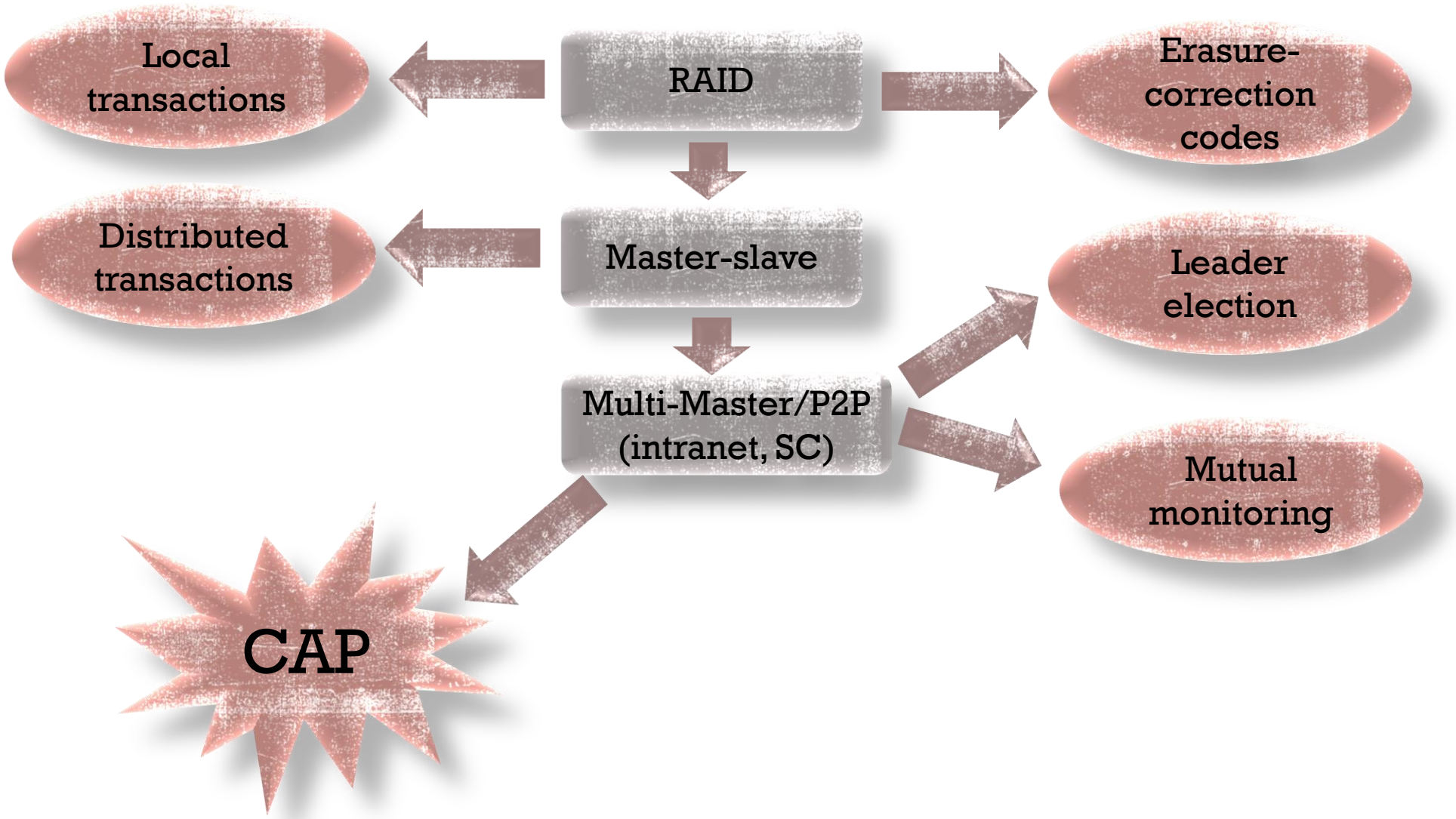
Одноранговые системы хранения (peer-to-peer)



Д.С. Мигинский

ЭВОЛЮЦИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ ХРАНЕНИЯ, СТРОГАЯ СОГЛАСОВАННОСТЬ



RAID

RAID – Redundant Array of Inexpensive Disks

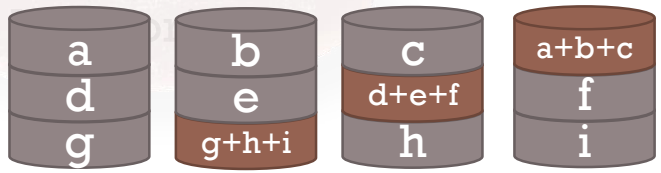
RAID 0 – сквозное пространство из нескольких дисков с чередованием

RAID 1 – “зеркало” из 2-х или более дисков

RAID 4 – RAID 0 + дополнительный диск с четностью

RAID 5 – RAID 4 с “перемешанными” блоками данных и четности

RAID 6 – RAID 5 с еще одной контрольной суммой

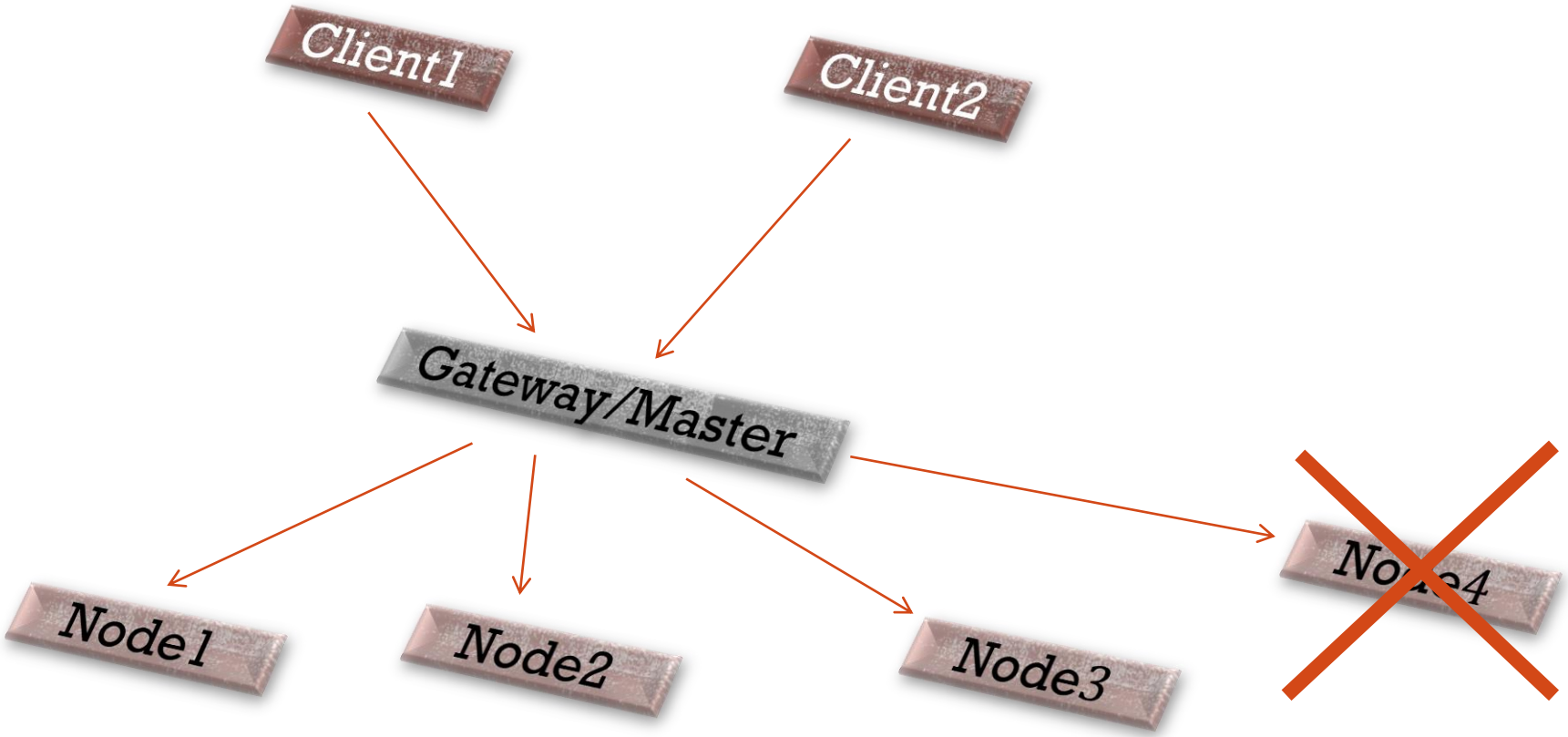


RAID 5

MASTER-SLAVE

Локальная транзакция – один процессор, несколько ресурсов

Распределенная транзакция – несколько процессоров, несколько ресурсов



2-PHASE COMMIT PROTOCOL

Протокол должен учитывать, что в любой момент может произойти сбой (порваться связь и т.д.)

РАСПРЕДЕЛЕННЫЕ ТРАНЗАКЦИИ

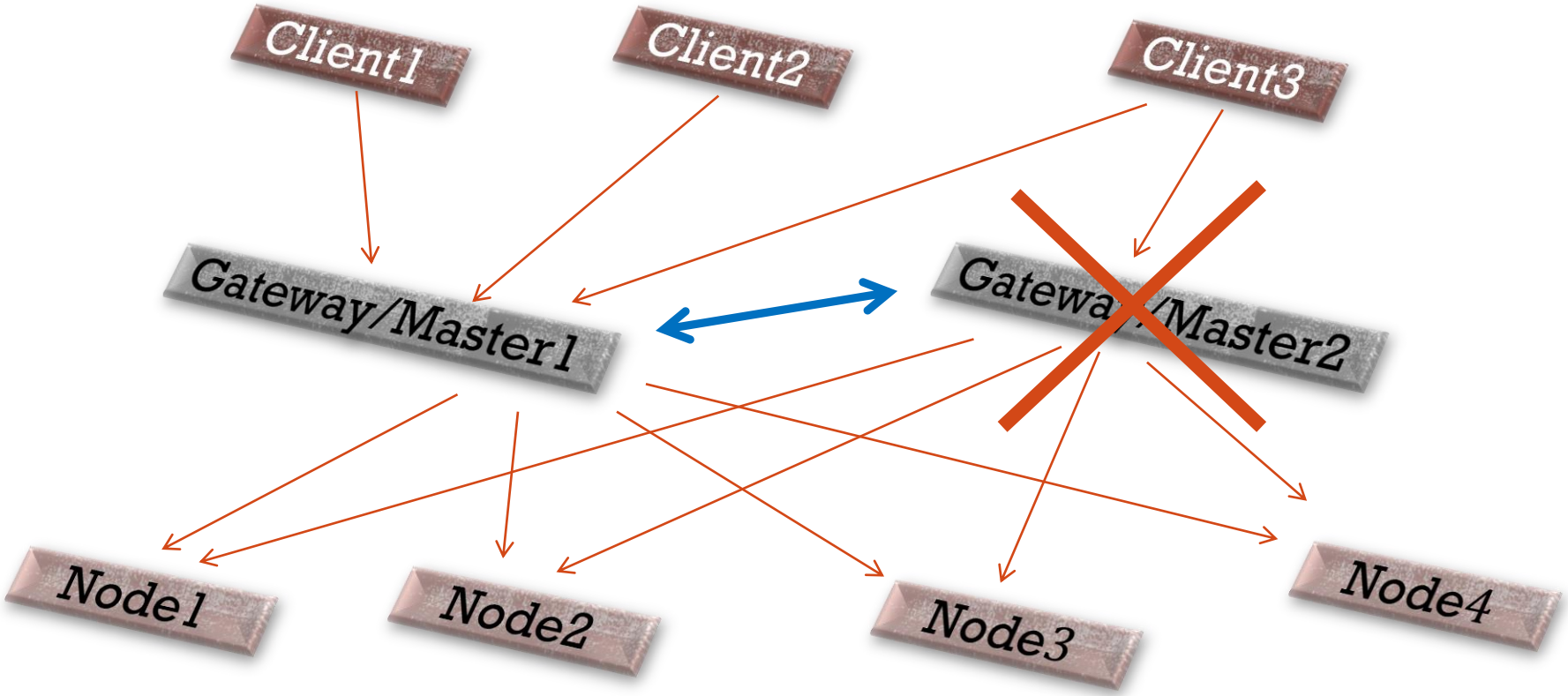
Проблема: Все узлы участвующие в транзакции должны принять <u>согласованное</u> решение о завершении транзакции (фиксация или откат), в т.ч. в условиях сбоев.

2-phase commit protocol – обеспечивает согласование в случае единичной ошибки

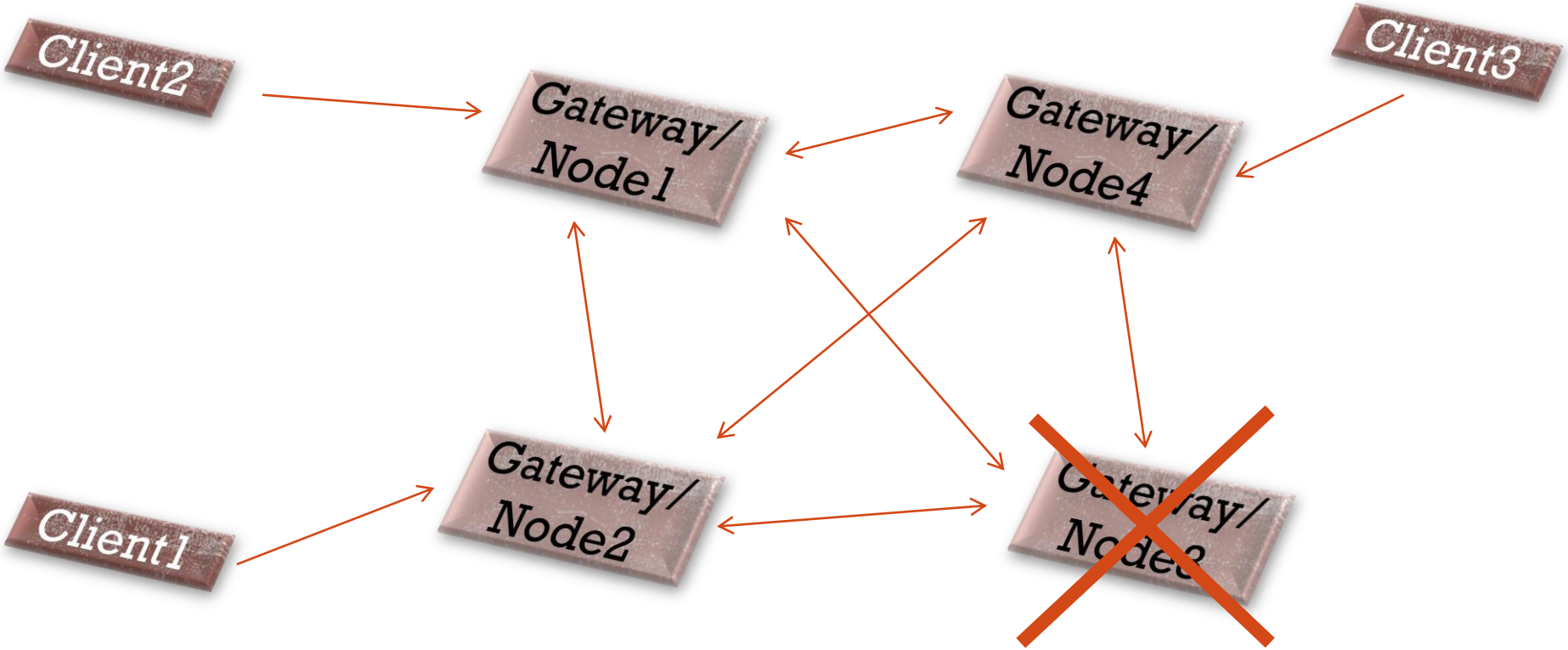
Quorum-based 3-phase commit – несколько больше

Проблема: а что если ошибок больше?
Проблема: откат является корректным завершением для протокола, но не для пользователя
Проблема: а что, если мастер транзакции дал сбой?

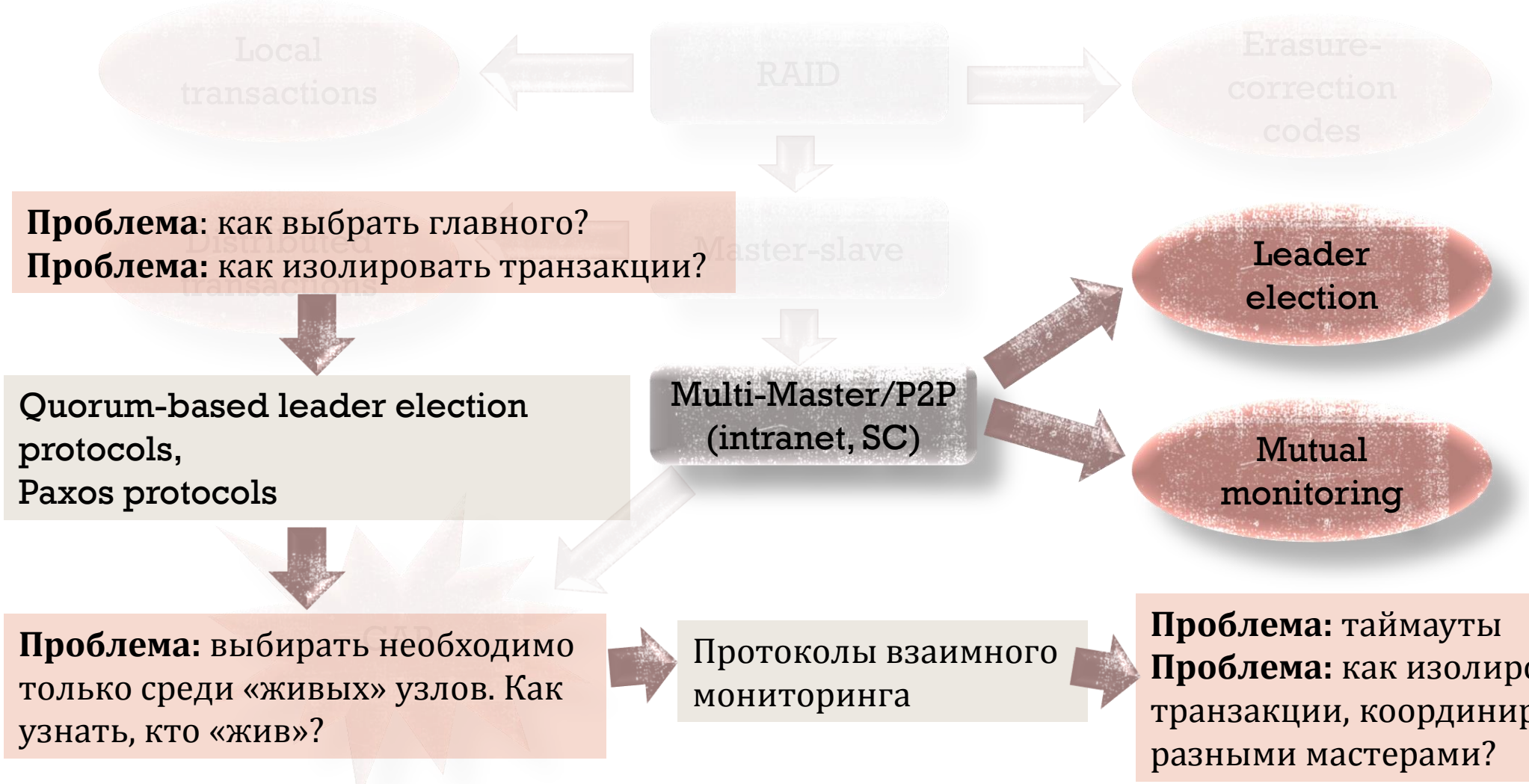
MULTI-MASTER



PEER-TO-PEER


Client2
Client3
Gateway/
Node1
Gateway/
Node4
Client1
Gateway/
Node2

ДЕЦЕНТРАЛИЗАЦИЯ



Теорема Брюера (CAP)

Из трех свойств распределенных систем:

Consistency (согласованность)

Availability (доступность)

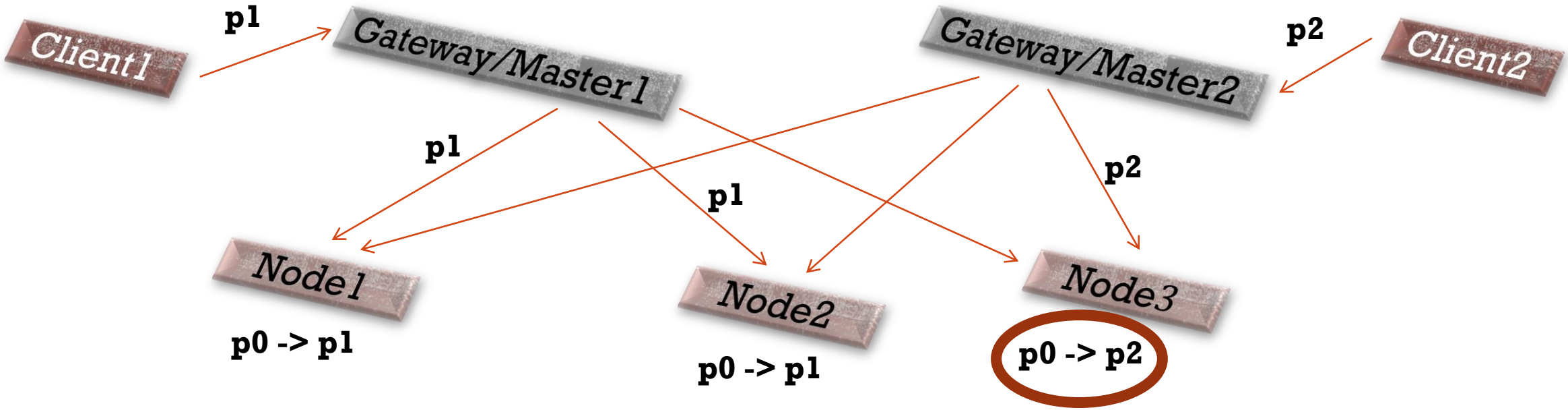
Partitioning tolerance (отказоустойчивость)

одновременно можно реализовать только два

CONSISTENCY & REPLICATION

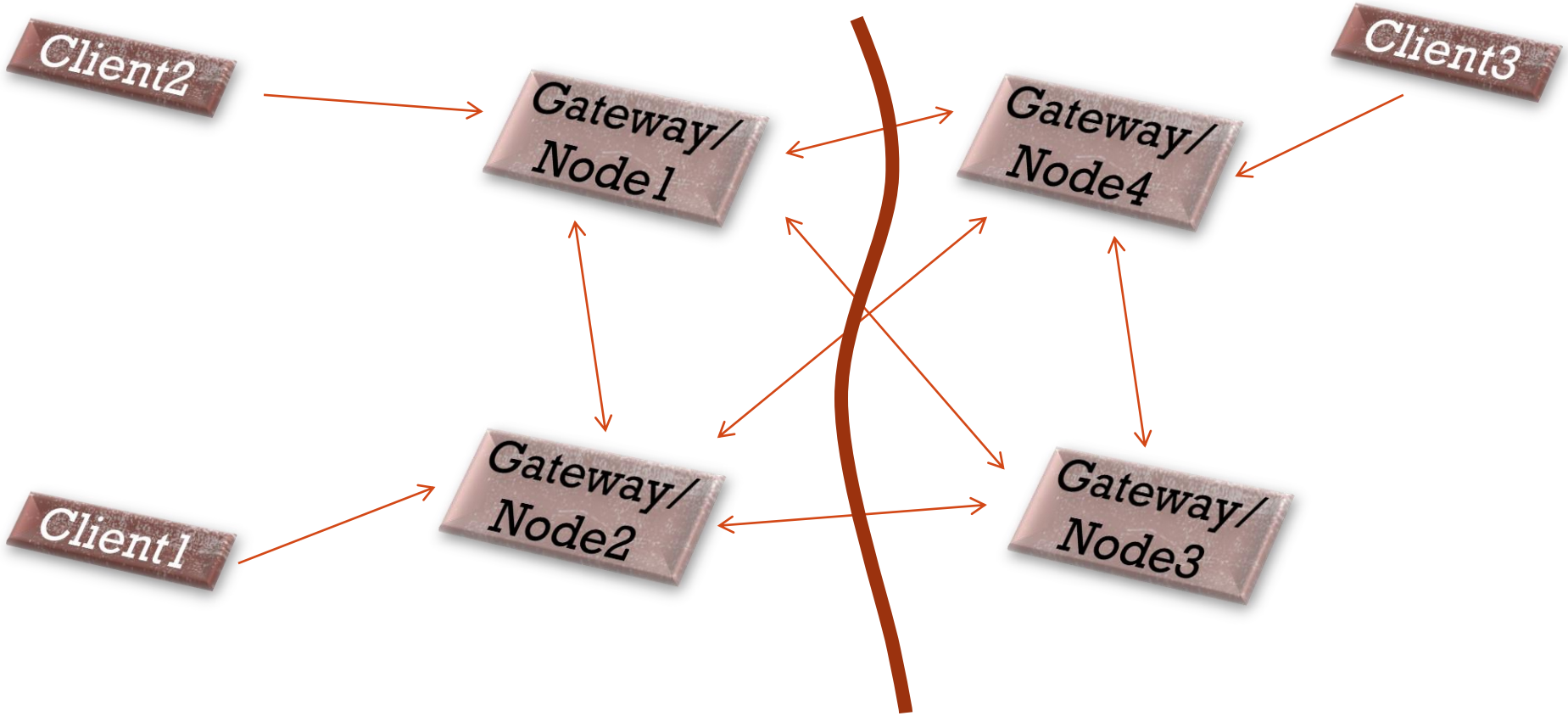
Модель данных: key-value (ключ-значение, словарь, ассоциативный массив)

Репликация – каждое значение хранится в нескольких экземплярах (репликах) на разных узлах



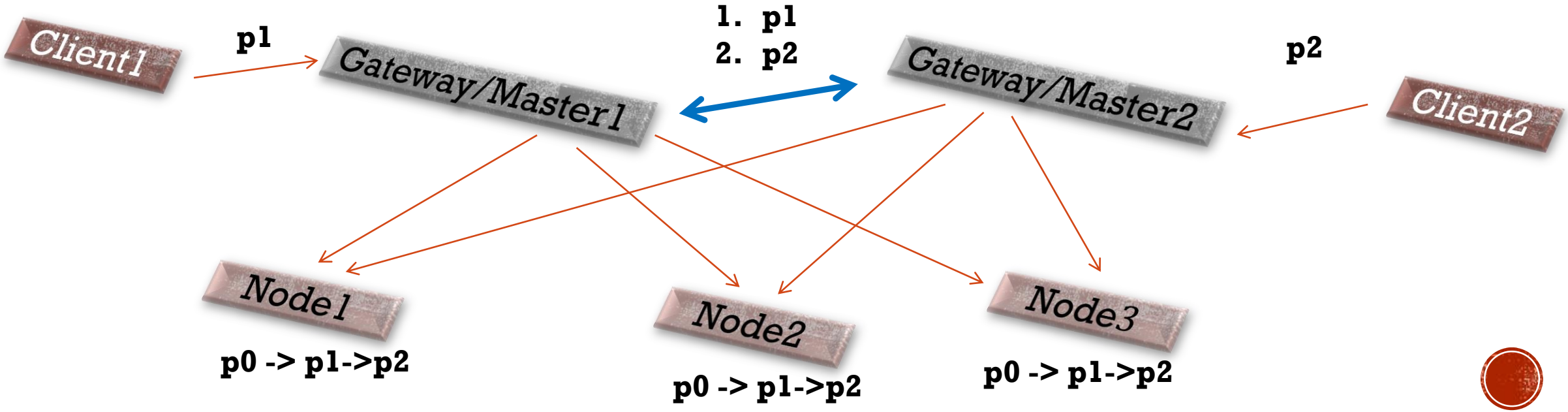
PARTITIONING TOLERANCE

Partitioning tolerance – сколько можно потерять узлов без потери доступности данных



(HIGH) AVAILABILITY

High availability – качественная характеристика системы, масштабируемость системы с точки зрения количества обрабатываемых запросов в единицу времени.
При росте количества узлов в системе количество запросов должно расти линейно/квазилинейно, а при потере узлов – деградировать квазилинейно.



CAP: ОСНОВНЫЕ КЛАССЫ СИСТЕМ

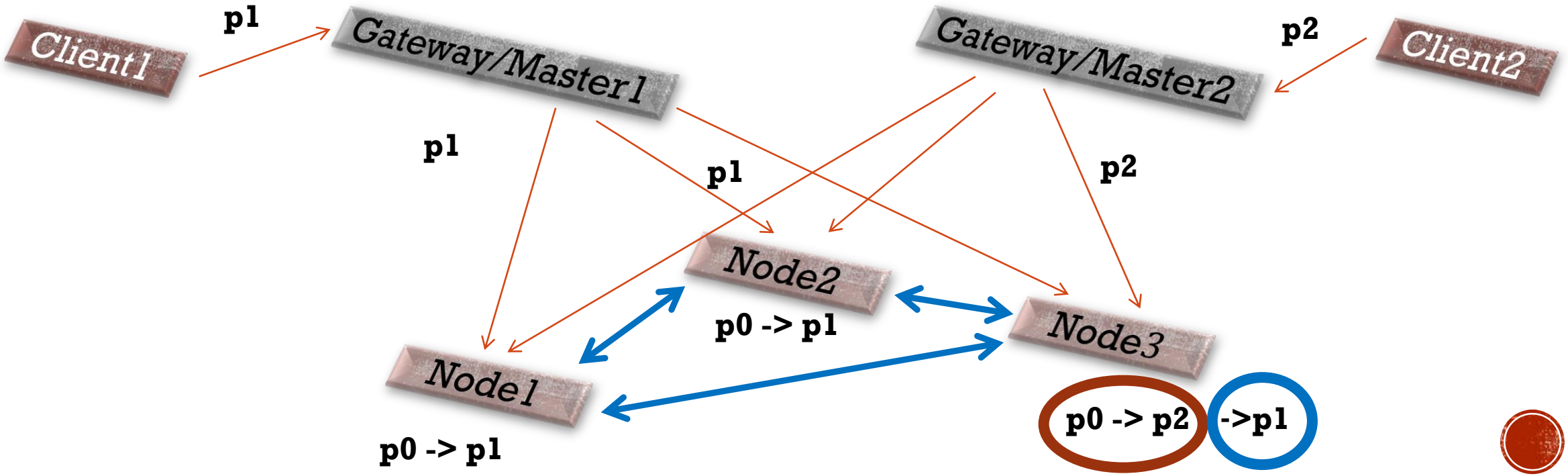
CA – системы, нетерпимые к сбоям. Для хранения сколько-нибудь важных данных неприменимы.

CP – системы со строгой согласованностью (**Strong consistency, SC**), «пессимистичные системы». Банковское ПО и т.п.

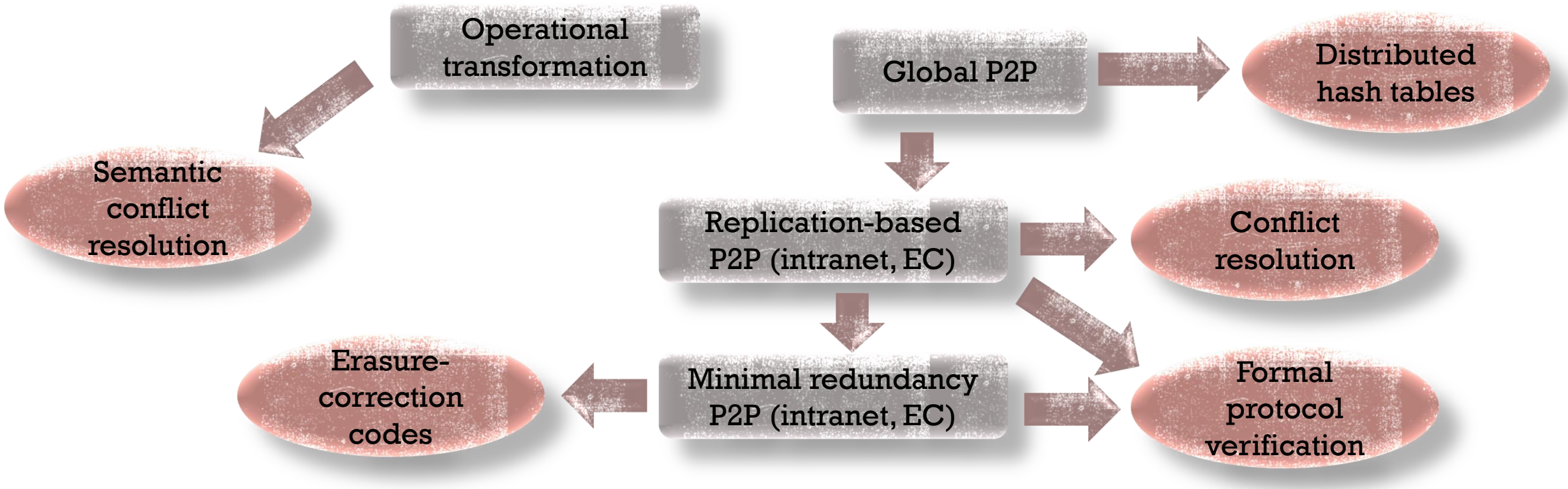
AP – системы, до некоторой степени жертвующие согласованностью в пользу доступности.

EVENTUAL CONSISTENCY (AP-CLASS)

Eventual consistency (EC, согласованность в конечном счете) – если в системе не происходит никаких событий (в т.ч. сбоев), то она придет в согласованное состояние за некоторое гарантированное время.



ДИВНЫЙ НОВЫЙ МИР: СОГЛАСОВАННОСТЬ В КОНЕЧНОМ СЧЕТЕ



GLOBAL P2P

Основная проблема: обеспечение работоспособности в сетях в тысячи узлов и более без перегрузки сетевой инфраструктуры

Способы решения:

1. Частичная централизация
2. Distributed Hash-Table Protocols (DHT)

Примеры:

Domain Name Services (DNS)

Файлообменные сети (Torrent, Direct Connect)

Block-chain системы (криптовалюты)

INTRANET P2P STORAGE

Основные задачи:

1. Обеспечить высокую доступность
2. Обеспечить масштабируемость дискового пространства
3. Обеспечить отказоустойчивость на заданном пользователем уровне

Основные элементы протокола:

1. На нижнем уровне: пользовательские операции **put** и **get** по ключу (модель ключ-значение), через любой узел кластера.
2. Взаимный мониторинг на предмет сбоев и конфликтов
3. Разрешение конфликтов (синтаксическое)
4. Распределение данных, в т.ч. перераспределение при сбоях

Известные реализации:

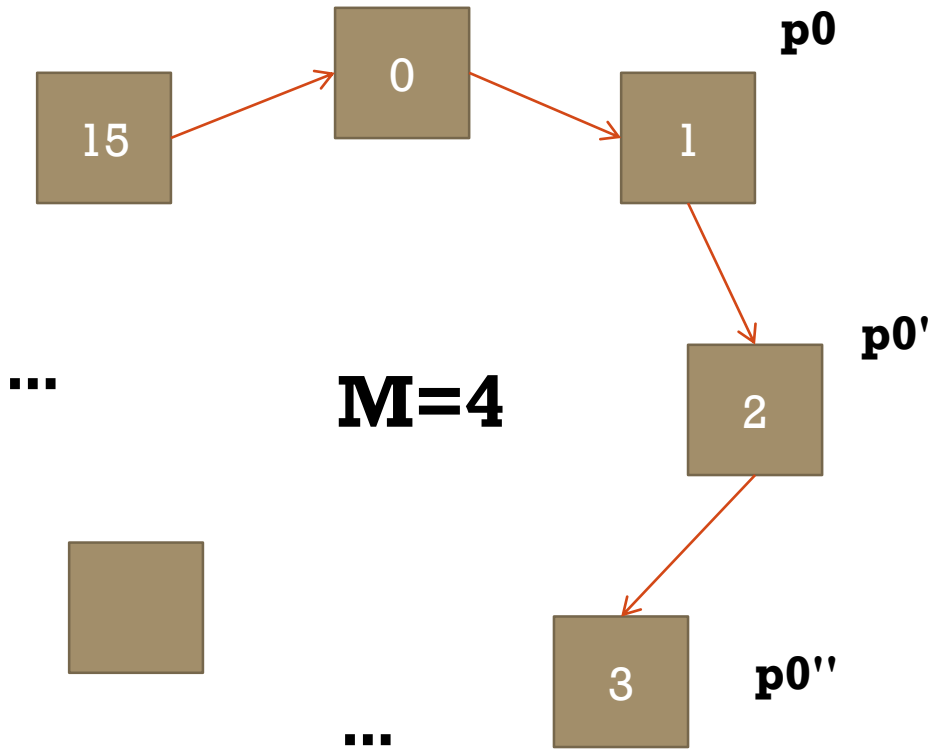
1. Amazon DynamoDB (S3)
2. Cassandra
3. MongoDB
4. Riak
5. …

ПРОТОКОЛ DYNAMO: ПРОСТРАНСТВО ДАННЫХ

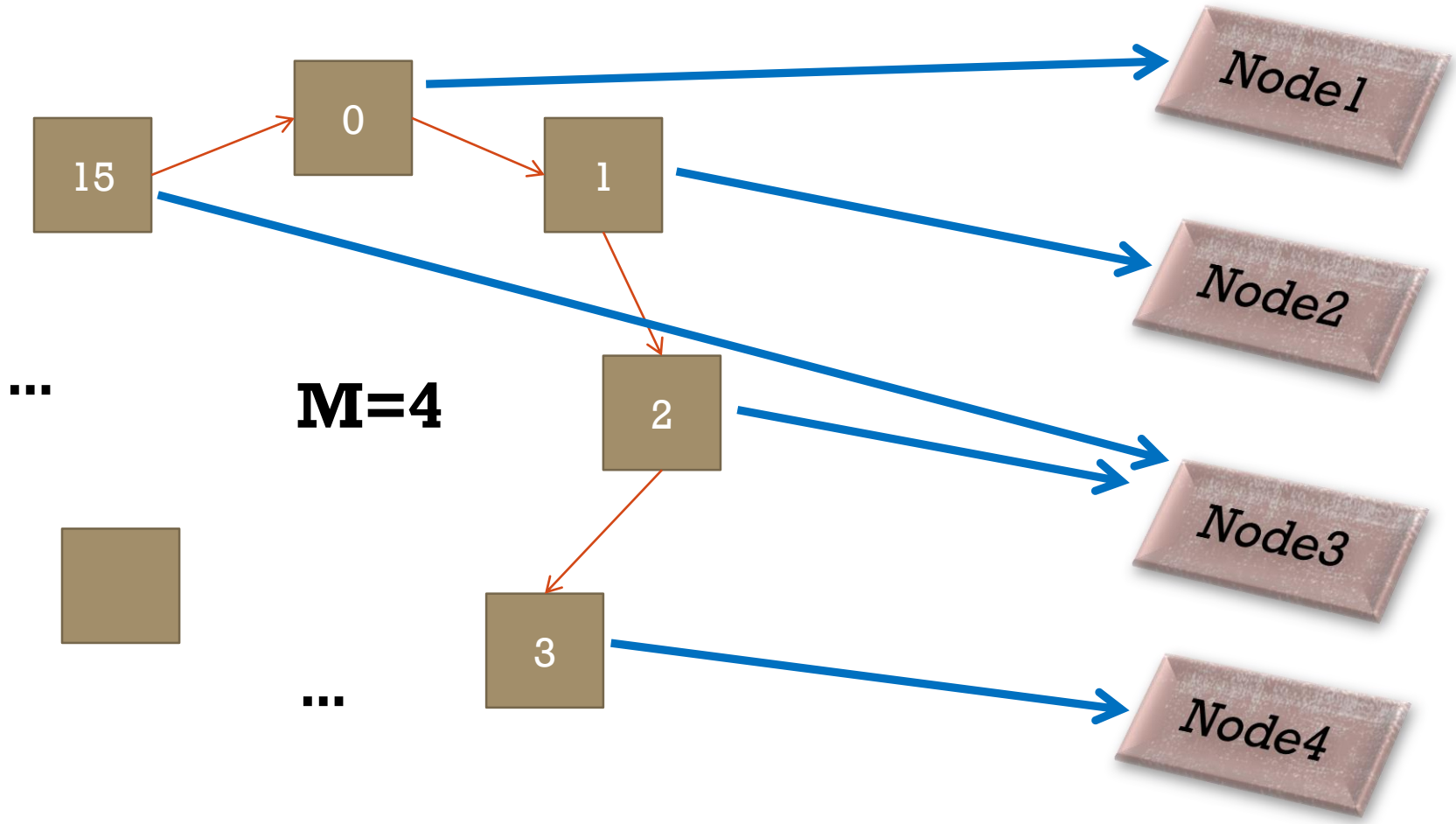
Ключ – произвольная строка
Значение – произвольная последовательность байт

Хэш-значение ключа – целое размера N бит. (N=32, 64, 128…)

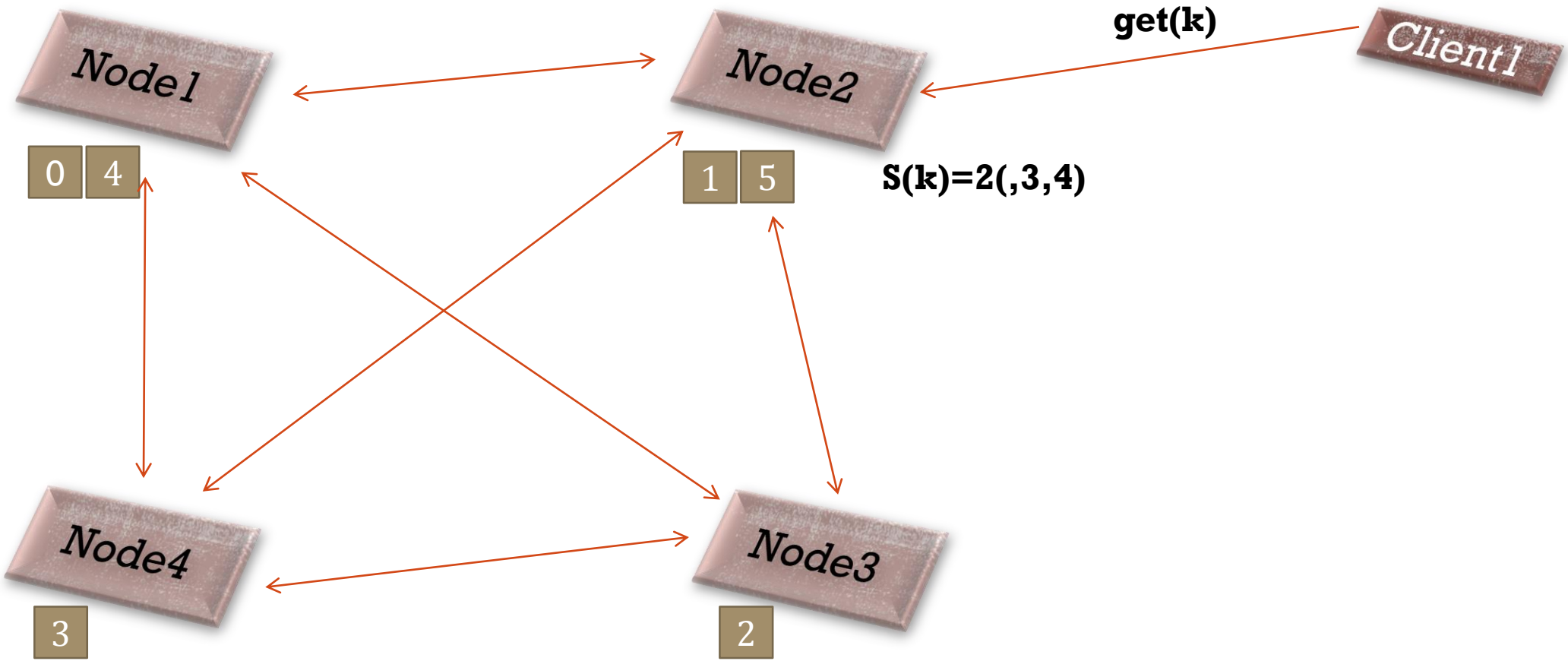
Виртуальный узел – подмножество хэш-значений ключа, идентифицируется целым размера M<<N (типично M = 6..12)



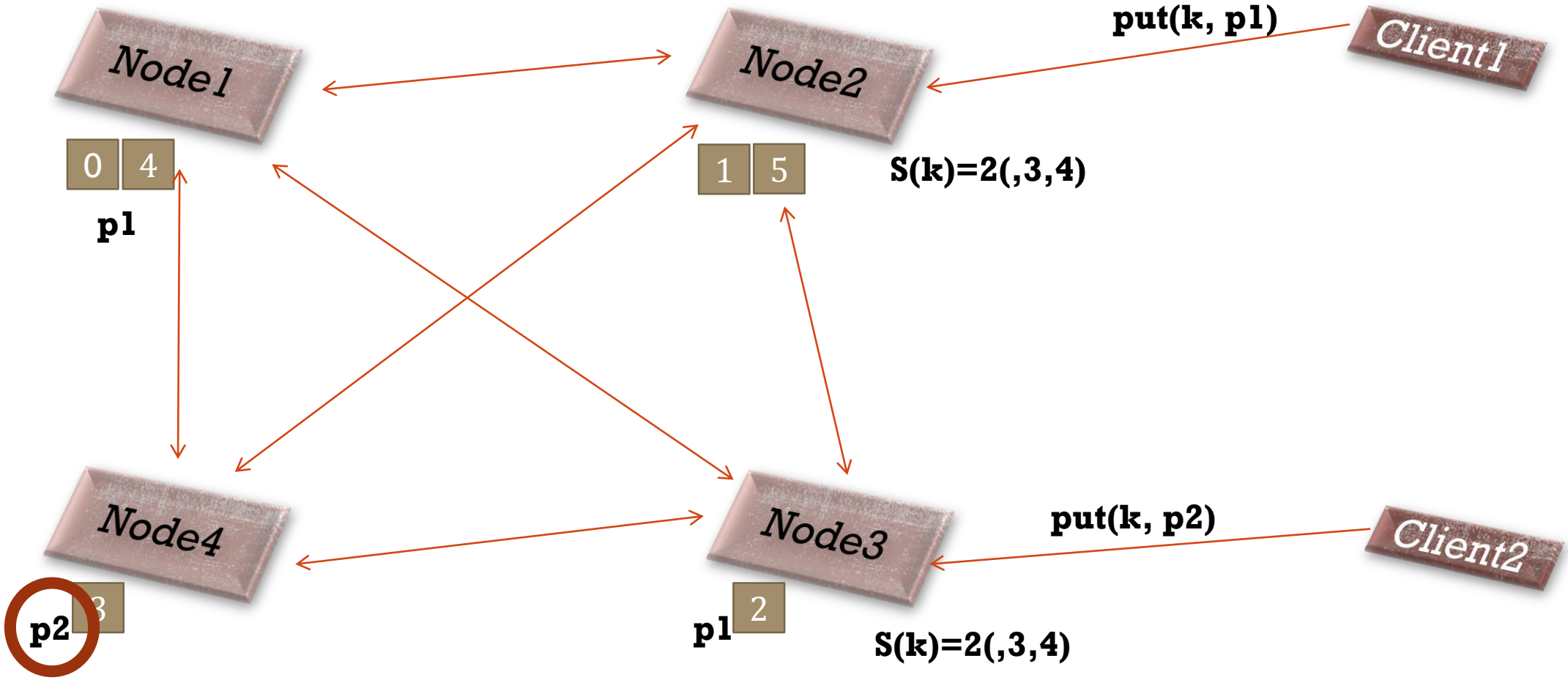
ПРОТОКОЛ DYNAMO: ФИЗИЧЕСКИЕ И ВИРТУАЛЬНЫЕ УЗЛЫ



ПРОТОКОЛ DYNAMO: ПОИСК



КОНФЛИКТ


Node1
Node2
Client1
put(k, p1)
0
4
p1
1
5
S(k)=2(,3,4)
Node4
Node3
Client2
put(k, p2)
p2
3
p1
2
S(k)=2(,3,4)

СИНТАКСИЧЕСКОЕ РАЗРЕШЕНИЕ КОНФЛИКТОВ

Основной принцип – кто последний, тот и прав

Проблема релятивизма – порядок событий в распределенной системе зависит от точки наблюдения

Решения:
1. Синхронизация часов в кластере сравнение временных меток
2. Часы Лампорта – логические часы (счетчик событий)
3. Векторные часы – счетчик событий с частичным порядком.

Проблема: потерянные обновления.

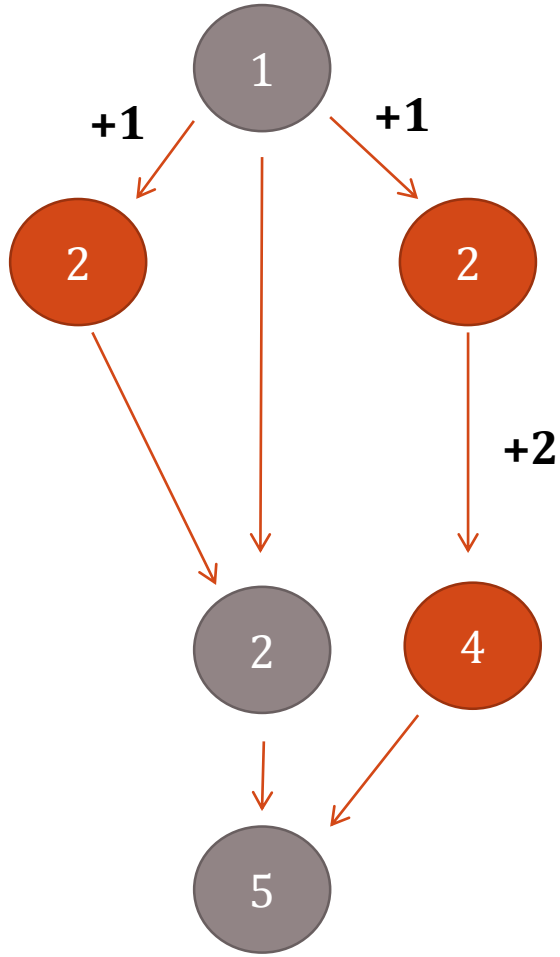
СЕМАНТИЧЕСКОЕ РАЗРЕШЕНИЕ КОНФЛИКТОВ

Основной принцип – помним операции над объектом и пытаемся разрешить конфликты путем объединения

Требуется переупорядочить операции в соответствии с их семантикой

Полезные свойства
- Коммутативность
- Обратимость
- Идемпотентность
- Предусловия и постусловия (логика Хоара)
- …

Основные применения
- Системы контроля версий (CVS, SVN, Git, …)
- Multi-Version Concurrency Control, Transactional Memory
- Operational Transformation (Google Docs)



DYNAMO: ПРОБЛЕМЫ

Высокая избыточность: эффективный объем пространства 1/**K**, где **K –** уровень репликации

Ограниченная масштабируемость: при росте кластера **K** следует повышать для сохранения уровня отказоустойчивости.

ERASURE CORRECTION CODES

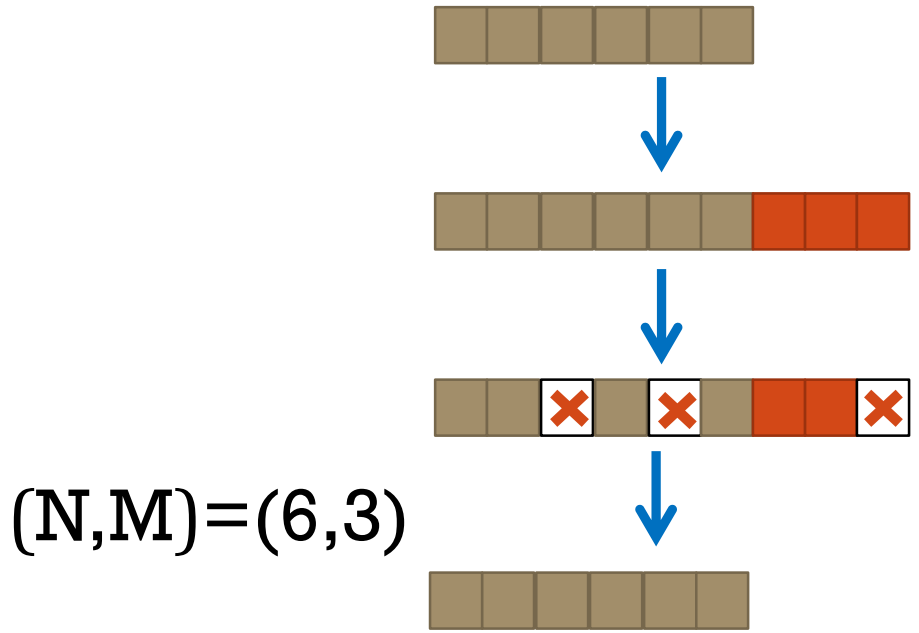
Error correction code – код, детектирующий и исправляющий ошибки (коды Хэмминга и др.)

Erasure correction code – код, исправляющий ошибки, которые должны быть детектированы внешним образом. Простейший пример – контроль четности (RAID 5)

КОДЫ РИДА-СОЛОМОНА

Блочный линейный код

Оптимальный код:
исходный объект разбиваем на **N** фрагментов, дополняем **M** контрольными фрагментами (схема **(N, M)**), далее по любым **N** фрагментам можно восстановить исходные данные.



КОДЫ РИДА-СОЛОМОНА

1. Строим конечное поле (размер 64, 128, 256, …)
2. Строим кодирующую матрицу **N x (N+M)**, где все миноры размера **N** невырождены (матрица Коши, Вандермонда и т.п.)
3. Приводим матрицу к виду с верхним единичным минором
4. При кодировании получаем вектор длины **N+M**
5. При восстановлении выбираем вектор из любых **N** «живых» фрагментов, обращаем соответствующий минор

$$\begin{pmatrix} 1 & \cdots & 0 \\ \vdots & 1 & \vdots \\ 0 & \cdots & 1 \\ x_{1,N+1} & \cdots & x_{N,N+1} \\ \vdots & \cdots & \vdots \\ x_{1,N+M} & \cdots & x_{N,N+M} \end{pmatrix} \begin{pmatrix} p_1 \\ \vdots \\ p_N \end{pmatrix} = \begin{pmatrix} p_1 \\ \vdots \\ p_N \\ q_1 \\ \vdots \\ q_M \end{pmatrix}$$

ПРОТОКОЛ HYDRA SHARDS

Идея: адаптировать **Dynamo** (или аналог), вместо репликации использовать коды Рида-Соломона, по виртуальным узлам раскладывать не реплики, а фрагменты

Плюсы:
- Степень избыточности ниже
- При масштабировании можно не повышать степень избыточности
- Потенциальная применимость для глобальных систем

Минусы:
- Данные не локализованы (всегда требуется сборка по кластеру)
- Ниже доступность
- Сложнее протокол

Кластер 12 узлов	Эффективная емкость	Допустимые потери
Репликация, степень 3	4 узла	2 узла
Кодирование (4,2)	8 узлов	2 узла
Кодирование (8,4)	8 узлов	4 узла

ПРИМЕРЫ ЗАДАЧ

1. **Резервное копирование данных в сеть**

2. **«Облачный сетевой диск»**

3. **Инфраструктура для хранения и обработки данных**

РЕЗЕРВНОЕ КОПИРОВАНИЕ

Дано: файлы 100 шт. по 10МБ, которые иногда модифицируется (очень ценные!)

Требуется: обеспечить периодически резервное копирование

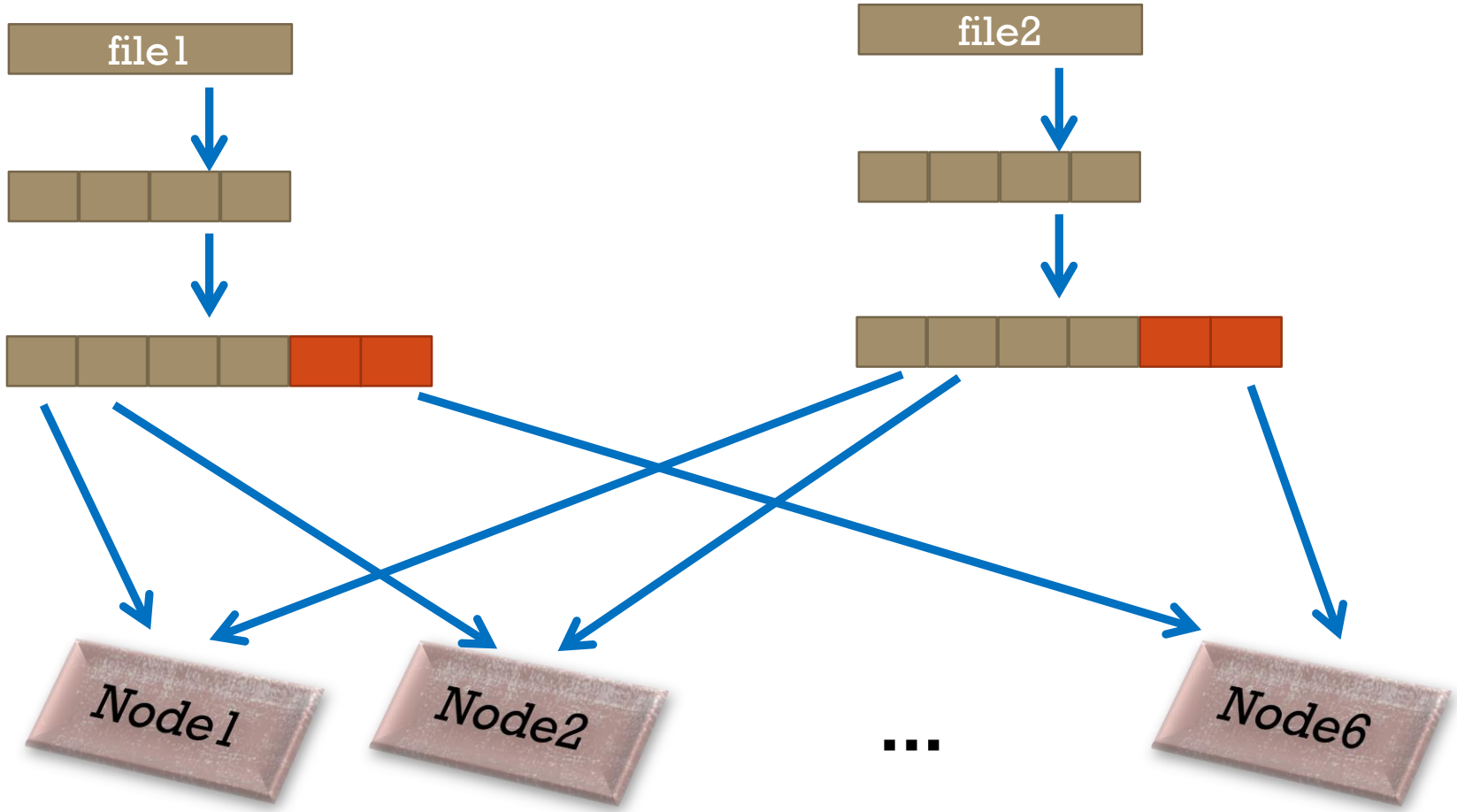
Решение 1: берем флэш-карту (или сетевой диск) и периодически копируем туда файлы.
Избыточность x2+

Решение 2: берем **p2p** сервис для резервного копирования, платим за это **x1.5+** дополнительным дисковым пространством.
Избыточность x2.5+

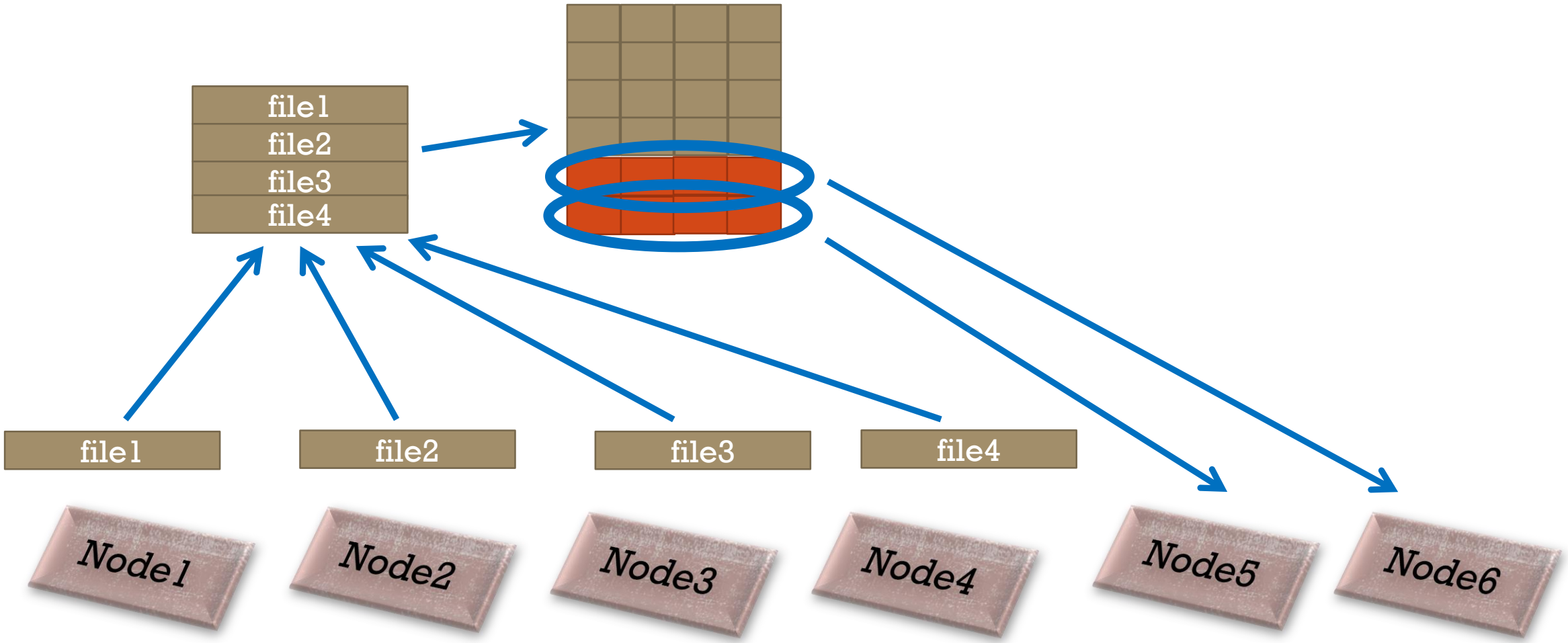
Решение 3: файл никуда не копируется, берем другой **p2p** сервис (гипотетический), который обеспечивает только хранение контрольных сумм. Платим за него **x0.5** дополнительным пространством.
Избыточность x1.5+
Т.е. ценой **x0.5** можем восстановить целое (**x1**)

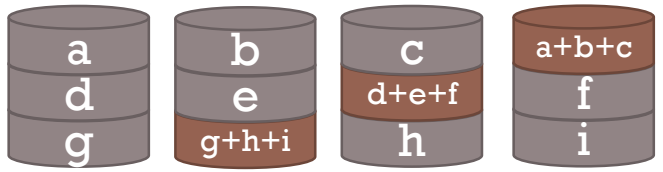
Как из части можно получить целое???

ПРОТОКОЛ HYDRA SHARDS



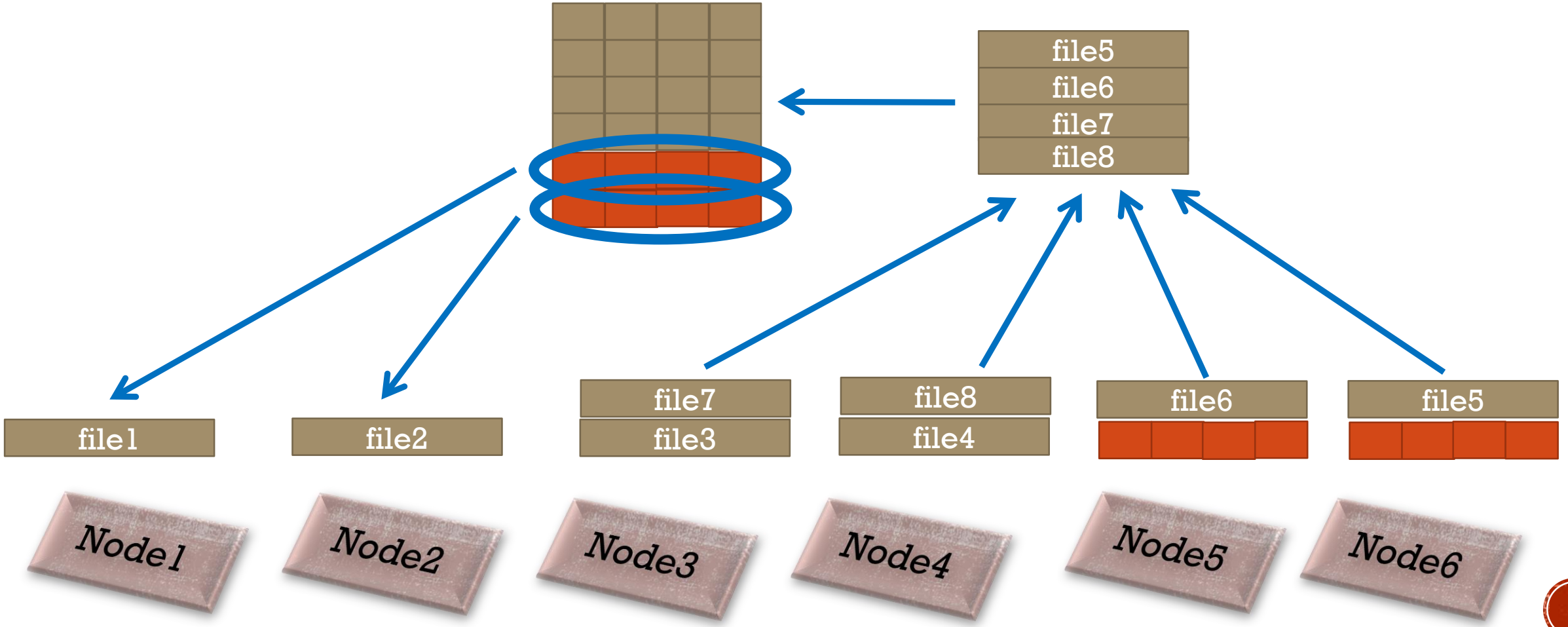
ПРОТОКОЛ HYDRA CHECKSUMS





Déjà vu?

ПРОТОКОЛ HYDRA CHECKSUMS



НЕОБХОДИМОСТЬ ФОРМАЛЬНОЙ ВЕРИФИКАЦИИ

Протокол операции put (Dynamo):

1. Вычислить виртуальный узел
2. Вычислить физические узлы
3. Послать мультикаст-запрос
4. Дождаться ответа (одного или всех)

Действия, если один из узлов не доступен:

1. …
2. …

А что делать, если в это же время:

- тот же объект модифицируется с другого узла?
- один из узлов сломался, и про это еще не известно?
- началась миграция нужного виртуального узла?
- включился протокол мониторинга, обнаружил конфликт и начал его исправлять?
- все это произошло одновременно?

ФОРМАЛЬНАЯ ВЕРИФИКАЦИЯ

Базовая модель для strong consistency:

- автоматная модель (детерминированный автомат) каждого узла
- автоматная модель (недетерминированный автомат) всей системы
- темпоральная логика (какая-нибудь) для утверждений о корректности
- критерий корректности - согласованность не должна нарушаться ни при каких сценариях!

Eventual consistency:

- в вышеописанном смысле все EC-протоколы некорректны (всегда можно придумать сценарий, где согласованность нарушается необратимо)
- НО вероятность такого сбоя может быть очень малой, так что на практике не реализуется
- модель должна оперировать вероятностными утверждениями о корректности
- выбор модели и верификатора – открытые вопросы