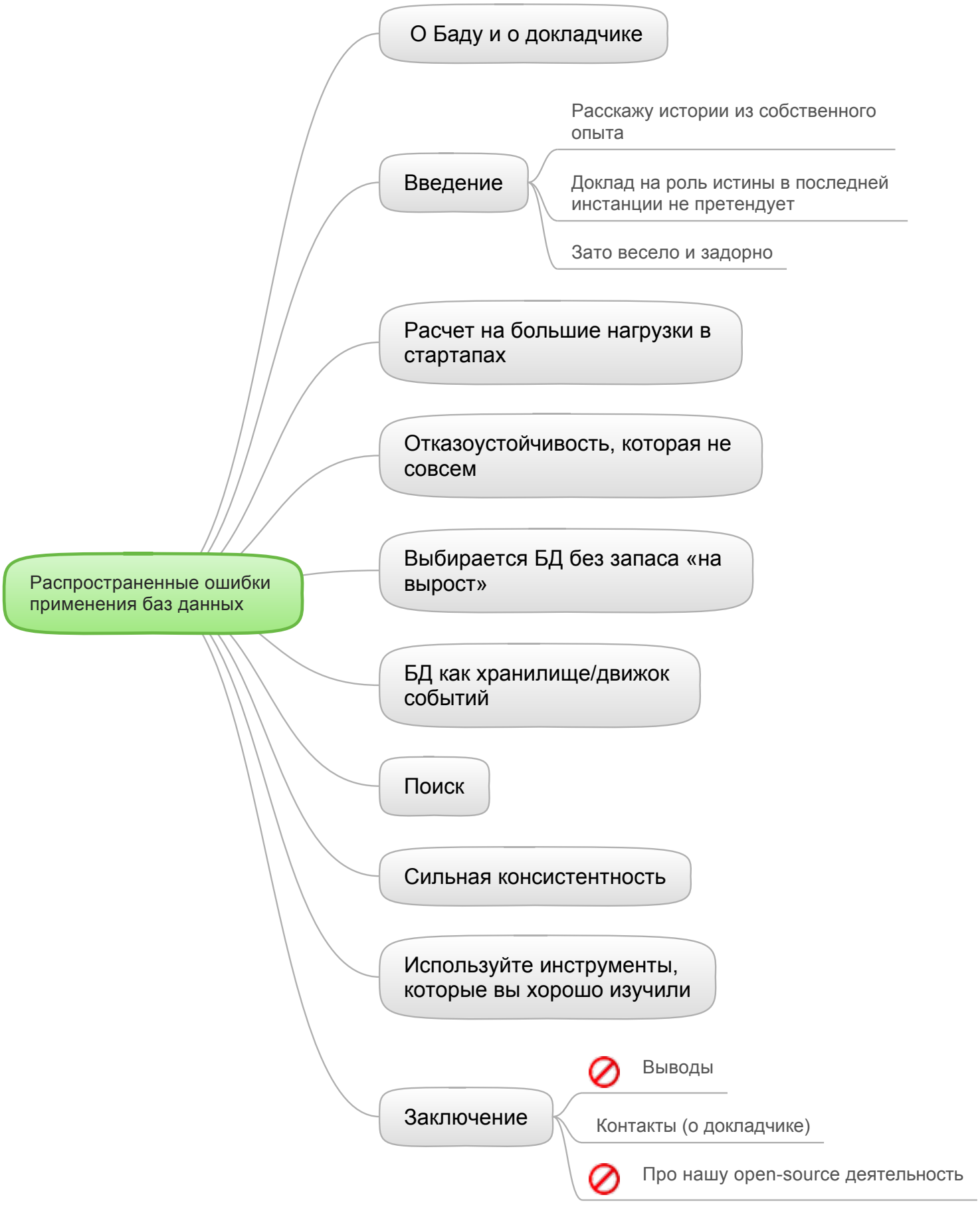
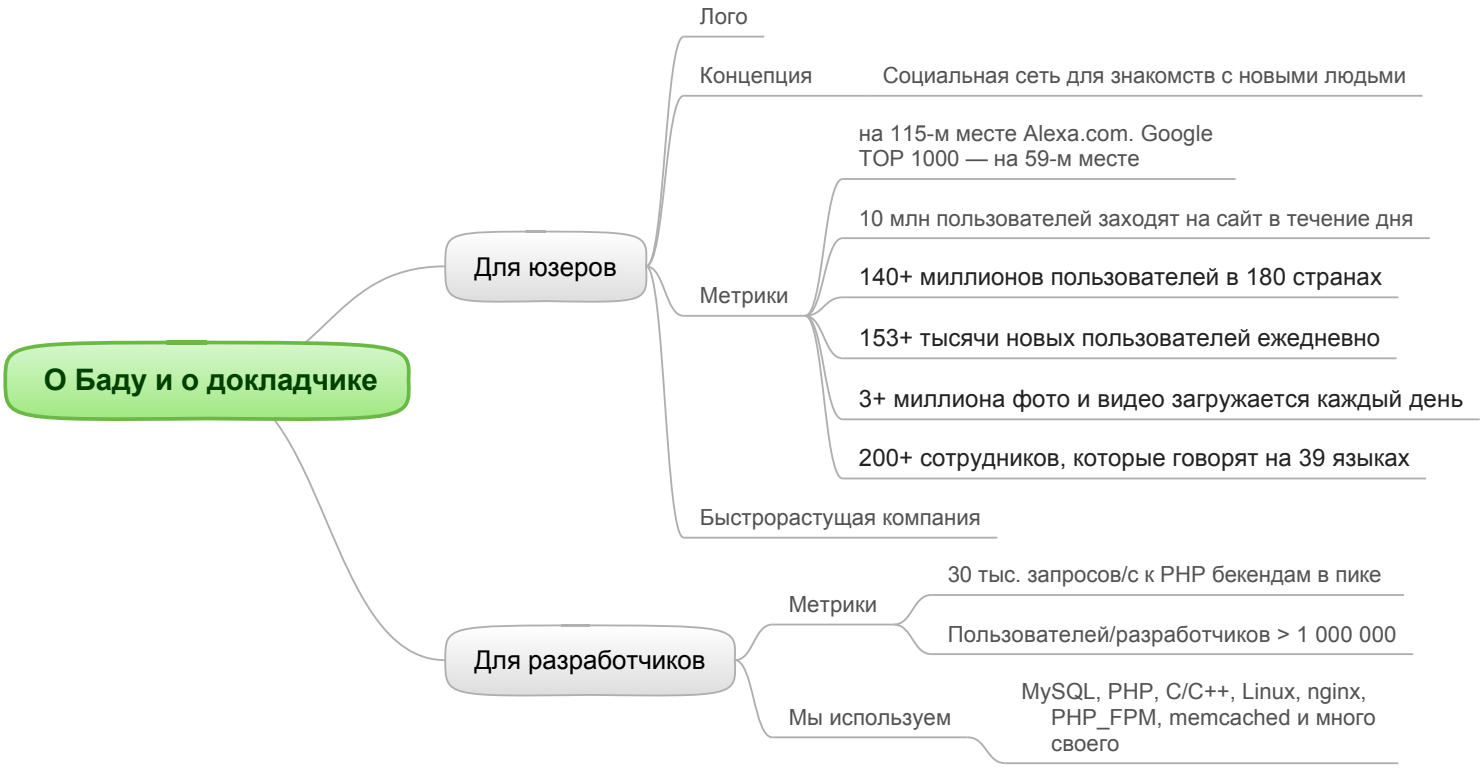

Распространенные ошибки применения баз данных
О Баду и о докладчике
Введение
Расскажу истории из собственного опыта
Доклад на роль истины в последней инстанции не претендует
Зато весело и задорно
Расчет на большие нагрузки в стартапах
Отказоустойчивость, которая не совсем
Выбирается БД без запаса «на вырост»
БД как хранилище/движок событий
Поиск
Сильная консистентность
Используйте инструменты, которые вы хорошо изучили
Заключение
Выводы
Контакты (о докладчике)
Про нашу open-source деятельность

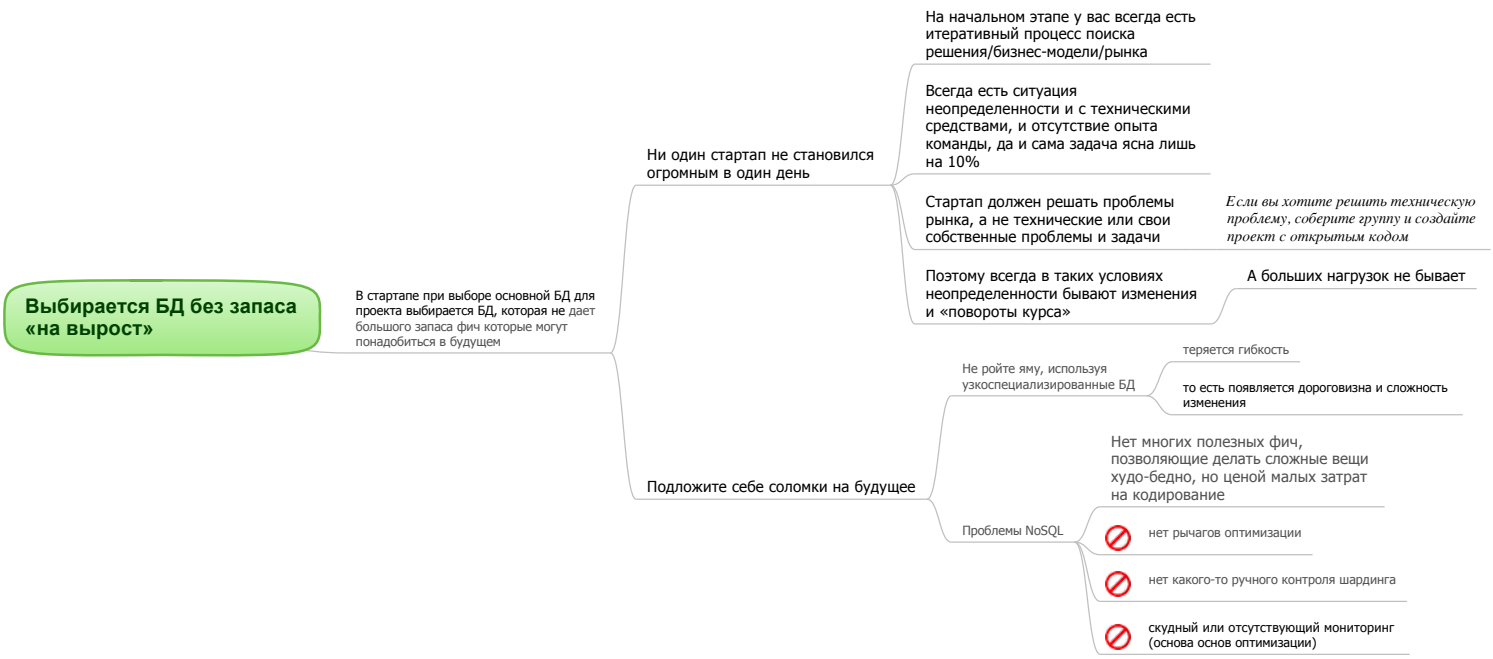

О Баду и о докладчике
Для юзеров
Лого
Концепция
Социальная сеть для знакомств с новыми людьми
Метрики
на 115-м месте Alexa.com. Google TOP 1000 — на 59-м месте
10 млн пользователей заходят на сайт в течение дня
140+ миллионов пользователей в 180 странах
153+ тысячи новых пользователей ежедневно
3+ миллиона фото и видео загружается каждый день
200+ сотрудников, которые говорят на 39 языках
Быстрорастущая компания
Для разработчиков
Метрики
30 тыс. запросов/с к PHP бекендам в пике
Пользователей/разработчиков > 1 000 000
Мы используем
MySQL, PHP, C/C++, Linux, nginx, PHP_FPM, memcached и много своего

Расчет на большие нагрузки в стартапах

- Зачастую в стартапе изначально проектируется архитектура вокруг БД, рассчитанная на огромные нагрузки, на большое масштабирование, которые потом в реальной жизни никогда не понадобится
- И на это тратятся огромные ресурсы, которые по сути не дадут результата
- Например, часто используется такой паттерн, как разбиение приложения на сервисы
- Как, большей частью, молодая команда представляет себе масштабируемость?
  - вынесение БД на отдельный сервер
  - клонирование бекендов
  - клонирование баз данных
  - SPOF нет
  - Все данные аккуратно нарезаются на шарды и не имеют зависимостей между шардами
  - шарды работают магически, их никогда не надо будет нарезать/ сливать/переносить
  - Сложные вопросы не рассматриваются по причине того, что слишком рано/мало опыта/ проблемы еще непонятны
- Закладываются возможности быстрого и большого роста кластера БД
- на самом деле, это предполагает, что ваши бизнес-метрики тоже вырастут в десятки и сотни раз, а архитектура сохранится
  - История про дома
    - Вот, что есть (избушка)
    - Вот, что видится в грезах (empire state building)
    - А вот, как собираемся это масштабировать (много избушек)
  - поскольку «серебряной пули» масштабирования нет
    - вы столкнетесь с целым классом новых уникальных для вашего проекта проблем
    - которые требуют творческого решения
    - что все равно придется многое переделывать
  - само по себе 'mongodb is web scale' — это хорошо
    - но на самом деле все это вершина айсберга проблем под названием highload
    - Которые все равно надо будет решать
- Для стартапа главными ценностями являются быстрый старт и дешевизна изменений
  - а вы зачастую, получается, идете вопреки этим ценностям
  - Мешая стартапу, отъедая ресурсы
  - Правильную архитектуру вы сможете сделать, когда найдете нужную бизнес-модель, определитесь с объемами рынка и т. д.
- Поэтому, на мой взгляд, если вы начинаете делать старт-ап
  - Начните с простых решений «по рецепту», быстрых и несложных
  - Одна БД скорее всего вас устроит
  - К моменту, когда вы перестанете влезать в один сервер, вы будете гораздо четче представлять, что вам нужно и какие специфичные проблемы вас преследуют

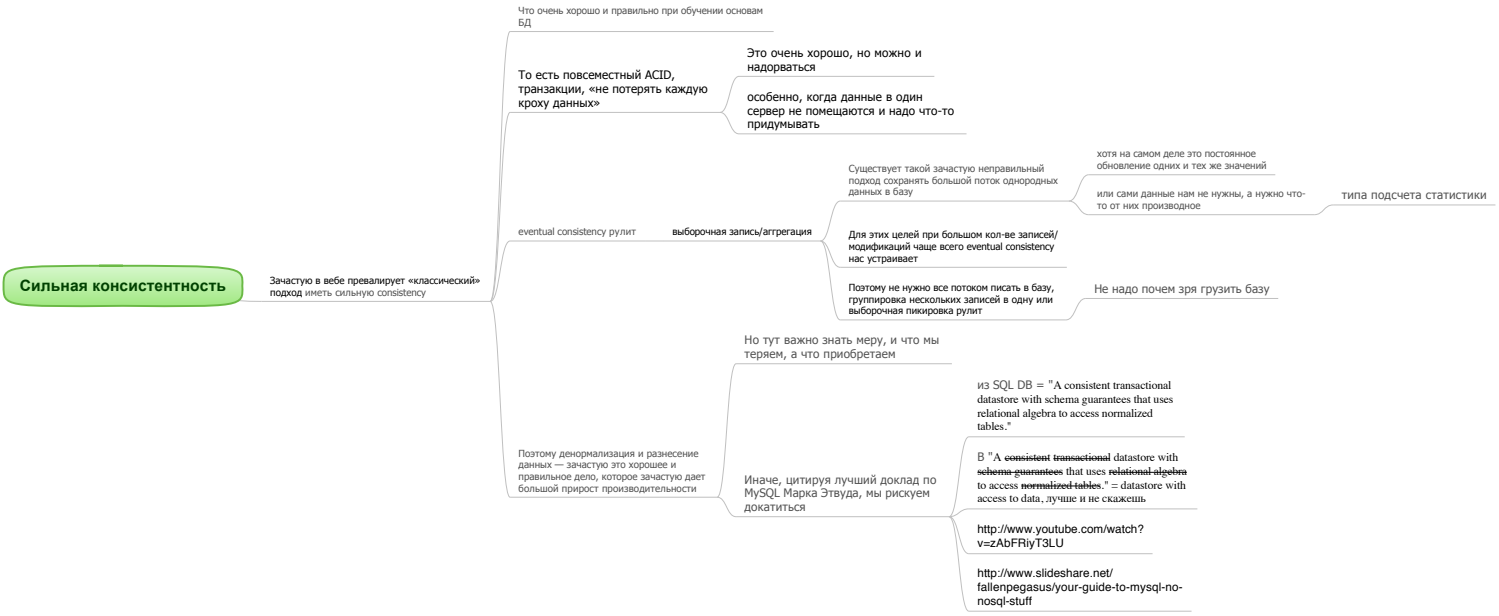
Отказоустойчивость, которая не совсем

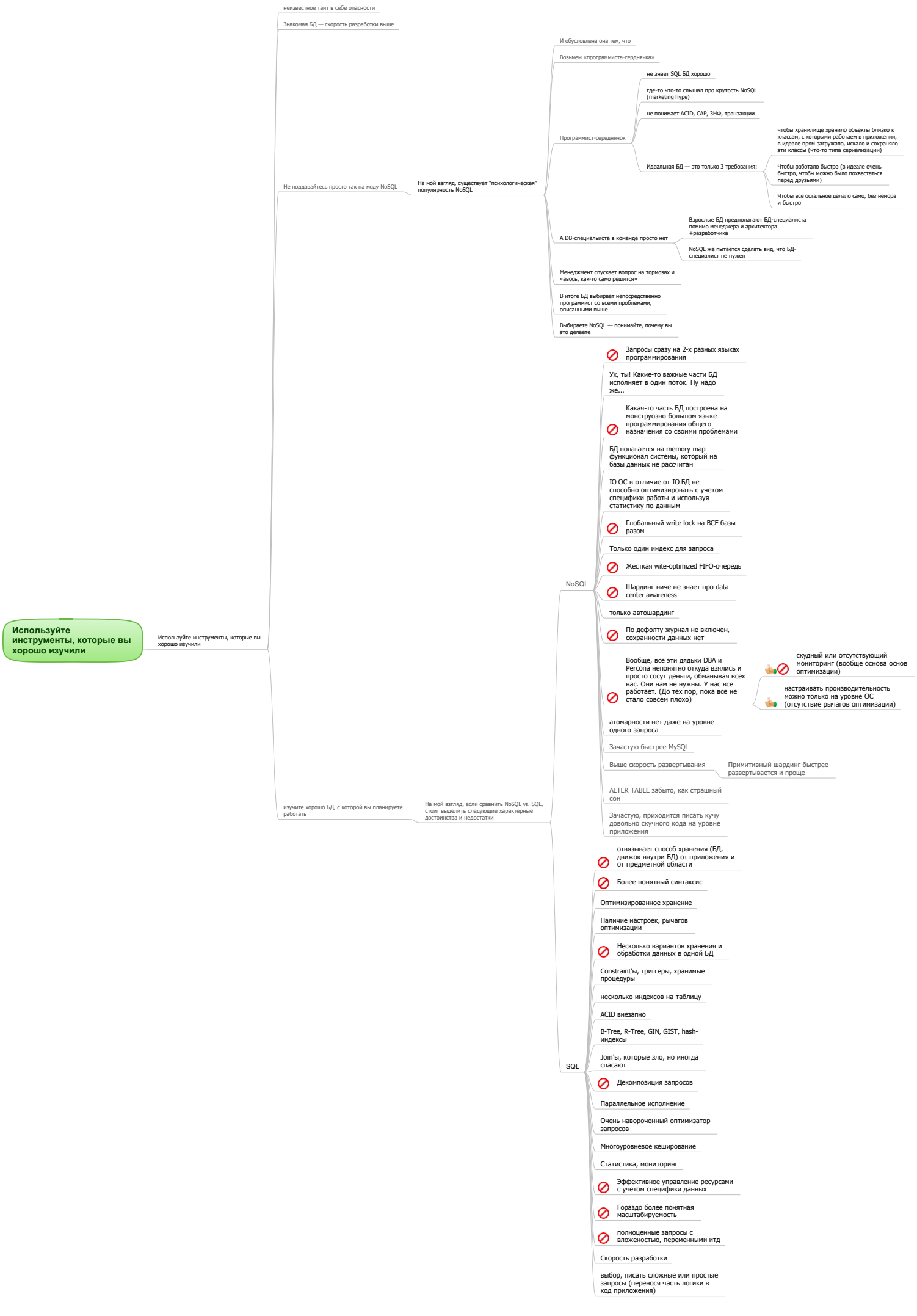
- Проектируется архитектура, которая якобы дает отказоустойчивость
  - Например, используется full ACID БД, включается журнал, делается реплика
  - но при этом проблемы нижних уровней во внимание не принимаются, а именно:
    - насколько отказоустойчиво железо, что в нем дублировано
      - волшебный raid-5 вопрос: что происходит при отказе? можно ли дальше быстро и стабильно работать?
    - Прошлогодняя история: а может ли сгореть ваш ДЦ вместе со всем содержимым?
    - насколько сильное влияние на вас может иметь human factor
    - насколько отказоустойчива ваша сеть. Будут ли у вас проблемы, если сеть ляжет или отвалится часть сети?
    - Насколько вероятно возникновение нескольких проблем одновременно
    - от каких внешних рисков еще вы зависите
    - Самая мешающая мне проблема: есть ли какая-то взаимосвязанность. Что будет, если, например, начнутся проблемы со скоростью работы БД, не ляжет ли все остальное?
  - Не все проблемы можно предотвратить полностью
  - Но парой вещей нужно поработать
    - в первую очередь надо работать над железом (серверами, сетью)
      - ломайте сами, смотрите, что получается
    - и потом конечно же над архитектурой приложения
      - В рамках одного **сервера** на практике отказоустойчивости не бывает
- Следующая известная проблема, связанная с отказоустойчивостью
- Как это сделано в Баду
  - для самых важных данных
    - юзерских данных
  - Специфика: данных очень много
  - Никакой магии
  - Выделенные сервера
    - проверенного вендора с мировым именем
    - резервирование по питанию
    - raid 1+0
  - Самосборный линукс на базе SLES
    - минимум пакетов
    - фаерволл
    - коммерчески поддерживаемая и протестированная сборка mySQL от Percona
    - несколько разных пользователей в базе, имеющих разные права доступа
    - chrooted environment
  - Новые данные изначально пишутся на один сервер в транзакции
    - Но очень быстро попадают в общую очередь
    - и синхронизируется через нее с другим ДЦ


Выбирается БД без запаса «на вырост»
В стартапе при выборе основной БД для проекта выбирается БД, которая не дает большого запаса фич которые могут понадобиться в будущем
Ни один стартап не становился огромным в один день
На начальном этапе у вас всегда есть итеративный процесс поиска решения/бизнес-модели/рынка
Всегда есть ситуация неопределенности и с техническими средствами, и отсутствие опыта команды, да и сама задача ясна лишь на 10%
Стартап должен решать проблемы рынка, а не технические или свои собственные проблемы и задачи
Если вы хотите решить техническую проблему, соберите группу и создайте проект с открытым кодом
Поэтому всегда в таких условиях неопределенности бывают изменения и «повороты курса»
А больших нагрузок не бывает
Подложите себе соломки на будущее
Не ройте яму, используя узкоспециализированные БД
теряется гибкость
то есть появляется дороговизна и сложность изменения
Проблемы NoSQL
Нет многих полезных фич, позволяющие делать сложные вещи худо-бедно, но ценой малых затрат на кодирование
нет рычагов оптимизации
нет какого-то ручного контроля шардинга
скудный или отсутствующий мониторинг (основа основ оптимизации)


БД как хранилище/движок событий
Использование БД как хранилища/движка обработки событий чаще всего не оправдано
Существует на мой взгляд 3 распространенных use case'а
события, порожденные транзакциями — 100% consistency
события, которые должны надежно доставляться - может прийти 2 раза, порядок не важен, главное не потерять
события, можно потерять - вообще ничего не важно
Вполне можно использовать специализированный движок
в 2х сценариях из трех
кроме первого
Что обычно дает бенефиты в
скорости
простоте и понятности
масштабируемости
доп. фичах, которые не надо реализовывать ручками и они могут понадобиться в будущем
Для некоторых задач мы используем систему, построенную вокруг Scribe
Своя обертка с агрегацией данных, вставкой в БД
Scribe-демон можно поставить на каждую машину — меньше соединений по сети
Отлично настраивается, как хранить пришедшие сообщения
При сбоях в точке сборки Scribe умеет сохранять пришедшие данные локально, пока не восстановится точка сборки
Очень быстро, в отличии от БД, потому что все основано на push а не poll

- **Поиск**
  - Поиск
    - Поиск в базе
      - либо быстро, просто, плохо
      - либо используя сложный фреймворк для организации поиска в базе
      - либо делаем сами такой фреймворк
    - Быстро, просто, плохо — 99%
      - Отсылаю к http://www.percona.com/files//presentations/opensql2008_sphinx.pdf
      - начинается все как комбинация из:
        - SELECT id, body FROM entries WHERE body LIKE '%one%'
      - Потом быстро надоедает, и делается так:
        - SELECT id, body FROM entries WHERE body RLIKE '[[:<:]]one[[:>:]]'
        - Some people, when confronted with a problem, think “I know, I’ll use regular expressions.” Now they have two problems. — Jamie Zawinsky
      - Потом используем встроенный в MySQL FULLTEXT Index
      - К чему это я — для простых решений, типа поиска по тегам или городов/улиц по первым буквам
        - В принципе, прекрасно работает поиск в базе через LIKE или обратный индекс
        - Но с полноценным поиском по тексту проблема даже не в скорости работы, скудности фич и плохом масштабировании — просто плохо ищет =)
          - мало фич
          - медленно
          - хуже масштабируется
        - А для каких-то задач просто неприменимо
          - Percona test: 2,5 млн записей, 15 Гб текста на одном сервере
            - Sphinx: 20 минут
            - MySQL: админ уснул через 6 часов, так и не дождавшись
          - По сравнению скорости работы отсылаю к http://www.slideshare.net/billkarwin/practical-full-text-search-with-my-sql
    - Что касается больших и сложных фреймворков
      - готовых, в open-source и известных я не знаю
      - написать свой, чтобы искал хорошо — из области rocket science
      - В данном случае БД отходит на второй план и ее использование скорее всего не оправдано (выглядит велосипедом)
    - Поиск в тексте — сильно специфичная задача
      - к БД относящаяся постольку-поскольку
    - Используйте специализированные поисковые БД/движки
      - Это и дешевле в разработке
      - быстрее работает
      - больше фич
      - отлично масштабируется
      - а главное — лучше ищет
      - Часто, наоборот, поисковый движок можно использовать как БД для своих задач


Сильная консистентность
Зачастую в вебе превалирует «классический» подход иметь сильную consistency
Что очень хорошо и правильно при обучении основам БД
То есть повсеместный ACID, транзакции, «не потерять каждую кроху данных»
Это очень хорошо, но можно и надорваться
особенно, когда данные в один сервер не помещаются и надо что-то придумывать
eventual consistency рулит
выборочная запись/аггрегация
Существует такой зачастую неправильный подход сохранять большой поток однородных данных в базу
хотя на самом деле это постоянное обновление одних и тех же значений
или сами данные нам не нужны, а нужно что-то от них производное
типа подсчета статистики
Для этих целей при большом кол-ве записей/модификаций чаще всего eventual consistency нас устраивает
Поэтому не нужно все потоком писать в базу, группировка нескольких записей в одну или выборочная пикировка рулит
Не надо почем зря грузить базу
Поэтому денормализация и разнесение данных — зачастую это хорошее и правильное дело, которое зачастую дает большой прирост производительности
Но тут важно знать меру, и что мы теряем, а что приобретаем
Иначе, цитируя лучший доклад по MySQL Марка Этвуда, мы рискуем докатиться
из SQL DB = "A consistent transactional datastore with schema guarantees that uses relational algebra to access normalized tables."
В "A ~~consistent transactional~~ datastore with ~~schema guarantees~~ that uses ~~relational algebra~~ to access ~~normalized tables~~." = datastore with access to data. лучше и не скажешь
http://www.youtube.com/watch?v=zAbFRiyT3LU
http://www.slideshare.net/fallenpegasus/your-guide-to-mysql-no-nosql-stuff


Используйте инструменты, которые вы хорошо изучили
Используйте инструменты, которые вы хорошо изучили
неизвестное таит в себе опасности
Знакомая БД — скорость разработки выше
Не поддавайтесь просто так на моду NoSQL
На мой взгляд, существует "психологическая" популярность NoSQL
И обусловлена она тем, что
Возьмем «программиста-середнячка»
Программист-середнячок
не знает SQL БД хорошо
где-то что-то слышал про крутость NoSQL (marketing hype)
не понимает ACID, CAP, 3НФ, транзакции
Идеальная БД — это только 3 требования:
чтобы хранилище хранило объекты близко к классам, с которыми работаем в приложении, в идеале прям загружало, искало и сохраняло эти классы (что-то типа сериализации)
Чтобы работало быстро (в идеале очень быстро, чтобы можно было похвастаться перед друзьями)
Чтобы все остальное делало само, без немора и быстро
А DB-специалиста в команде просто нет
Взрослые БД предполагают БД-специалиста помимо менеджера и архитектора +разработчика
NoSQL же пытается сделать вид, что БД-специалист не нужен
Менеджмент спускает вопрос на тормозах и «авось, как-то само решится»
В итоге БД выбирает непосредственно программист со всеми проблемами, описанными выше
Выбираете NoSQL — понимайте, почему вы это делаете
изучите хорошо БД, с которой вы планируете работать
На мой взгляд, если сравнить NoSQL vs. SQL, стоит выделить следующие характерные достоинства и недостатки
NoSQL
Запросы сразу на 2-х разных языках программирования
Ух, ты! Какие-то важные части БД исполняет в один поток. Ну надо же...
Какая-то часть БД построена на монструозно-большом языке программирования общего назначения со своими проблемами
БД полагается на memory-map функционал системы, который на базы данных не рассчитан
IO ОС в отличие от IO БД не способно оптимизировать с учетом специфики работы и используя статистику по данным
Глобальный write lock на ВСЕ базы разом
Только один индекс для запроса
Жесткая wite-optimized FIFO-очередь
Шардинг ниче не знает про data center awareness
только автошардинг
По дефолту журнал не включен, сохранности данных нет
Вообще, все эти дядьки DBA и Percona непонятно откуда взялись и просто сосут деньги, обманывая всех нас. Они нам не нужны. У нас все работает. (До тех пор, пока все не стало совсем плохо)
скудный или отсутствующий мониторинг (вообще основа основ оптимизации)
настраивать производительность можно только на уровне ОС (отсутствие рычагов оптимизации)
атомарности нет даже на уровне одного запроса
Зачастую быстрее MySQL
Выше скорость развертывания
Примитивный шардинг быстрее развертывается и проще
ALTER TABLE забыто, как страшный сон
Зачастую, приходится писать кучу довольно скучного кода на уровне приложения
SQL
отвязывает способ хранения (БД, движок внутри БД) от приложения и от предметной области
Более понятный синтаксис
Оптимизированное хранение
Наличие настроек, рычагов оптимизации
Несколько вариантов хранения и обработки данных в одной БД
Constraint'ы, триггеры, хранимые процедуры
несколько индексов на таблицу
ACID внезапно
B-Tree, R-Tree, GIN, GIST, hash-индексы
Join'ы, которые зло, но иногда спасают
Декомпозиция запросов
Параллельное исполнение
Очень навороченный оптимизатор запросов
Многоуровневое кеширование
Статистика, мониторинг
Эффективное управление ресурсами с учетом специфики данных
Гораздо более понятная масштабируемость
полноценные запросы с вложеностью, переменными итд
Скорость разработки
выбор, писать сложные или простые запросы (перенося часть логики в код приложения)