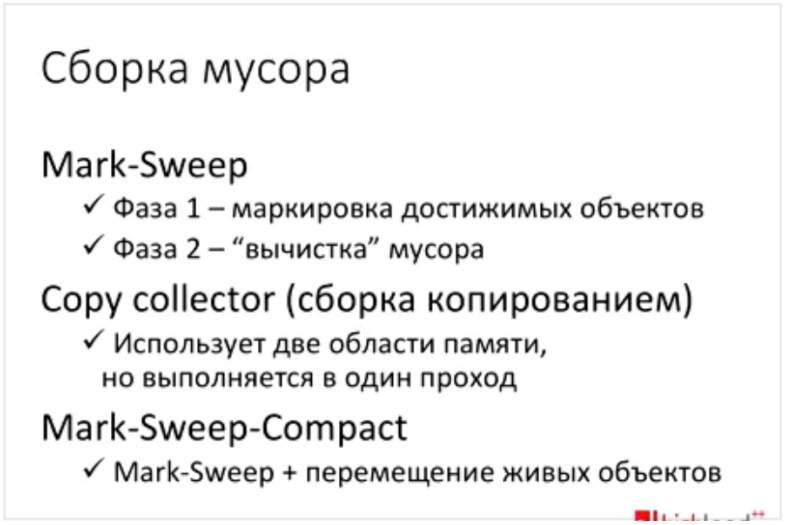
+- 2017 JVM | Алексей Рагозин - Сборка мусора в Java без пауз (Deutsche Bank)

https://www.youtube.com/watch?v=n89CZS0u6dY&ab_channel=HighLoadChannel

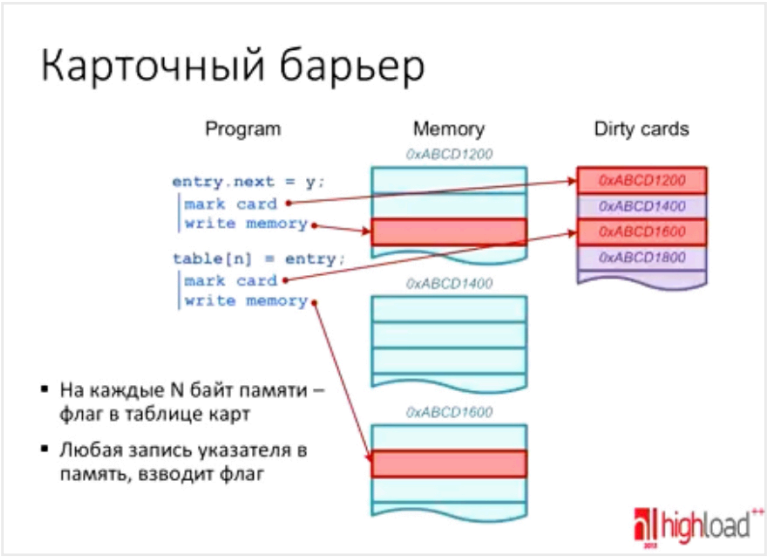
Важное:

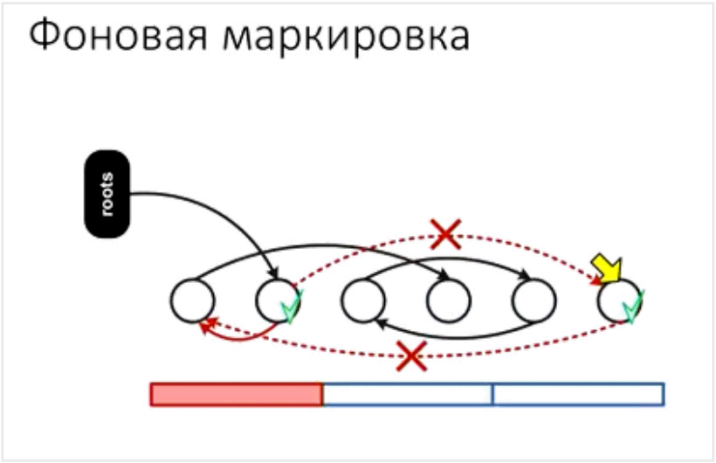
1. Concurrent mark sweep выполняет остановку приложения для завершения маркировки

В докладе рассмотрено создание jvm работающей без пауз.



Крайне сложно найти живые объекты в графе по мере сборки мусора, поскольку граф постоянно меняется.
Реализация фоновой маркировки(половина алгоритма concurrent mark sweep) в HotSpotJvm основывается на **карточном барьере**. На каждые 512 байт памяти есть флажок, каждый раз когда код что-то пишет в память, он зануляет этот флажок.

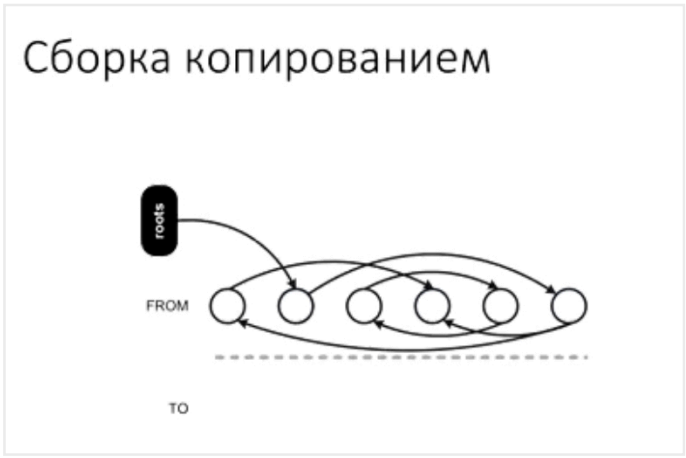


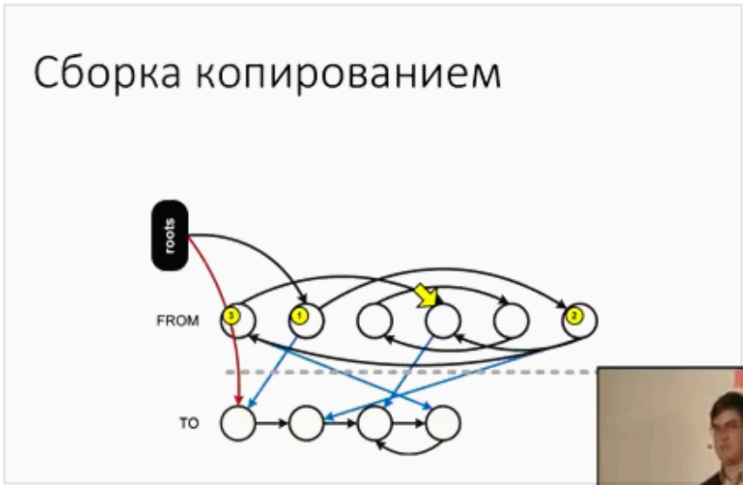


После выполнения маркировки проверяется таблица карт и те участки памяти, которые были модифицированы помечаются заново.

Поскольку мы делаем барьер на запись, а он срабатывает уже пост фактум, приходится иногда останавливать приложение, чтобы закончить маркировку.

Для алгоритма работающего без пауз нужен другой тип барьера - **барьер на чтение**.

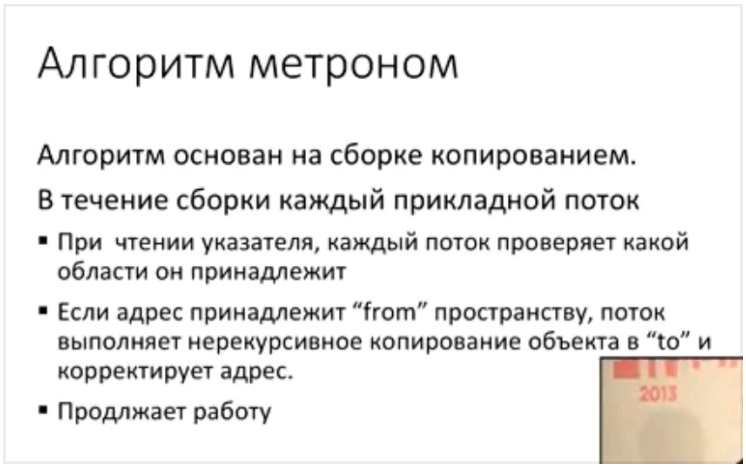




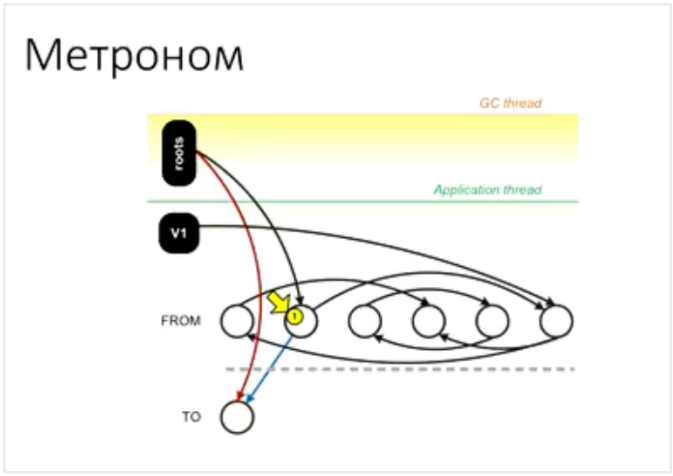
Сборка копирование выполняется для молодого поколения в hotSpotJvm.
Выполняется проход по объектам, к которым есть доступ из корневых ссылок, выполняется их копирование в другое пространство и вместе с этим чинятся ссылки на скопированные объекты. На практике данный алгоритм хорошо распараллеливается. Всю старую область памяти можно очищать. Однако этот алгоритм требует stop the world.

Большая часть академических работ по алгоритмам сборки мусора проводилась в 70-80 годах прошло века и

Алгоритм метроном - позволяет собирать мусор без пауз, основан на сборке копированием.



Отличие метронома от копирующего сборщика заключается в том, что после того как начался цикл сборки мусора все прикладные потоки работают только с данными в пространстве to.

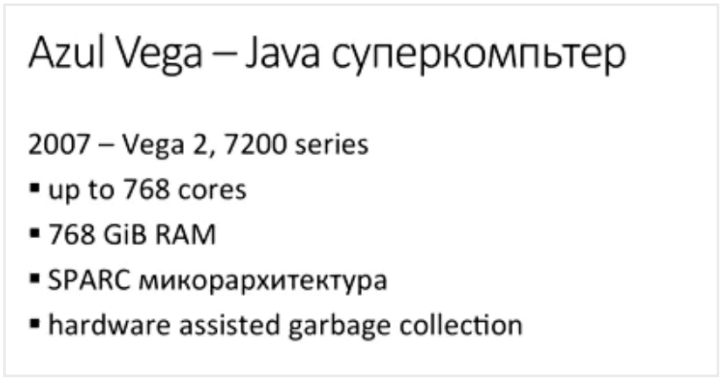


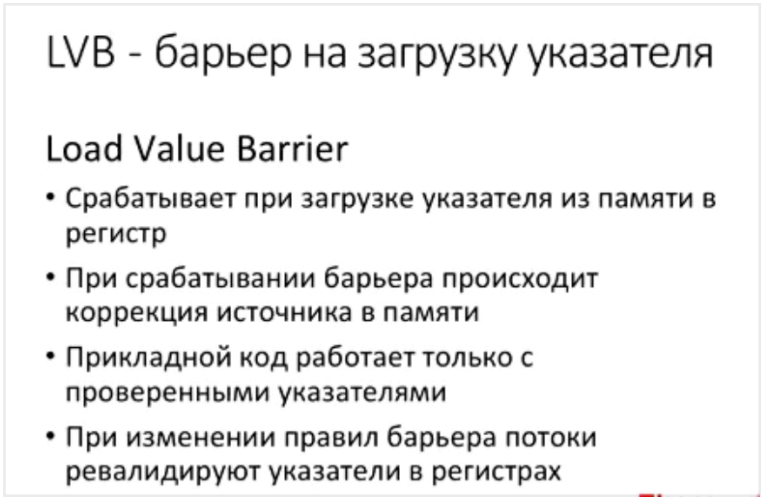
Проблемы алгоритма метроном:

1. Низкая скорость работы
2. Сложно обеспечить транзакционность в системе без блокировок
3. Сборка мусора с одним пространством не эффективная. Должно быть несколько поколений сборки.

В 2007 году компания Azul разработала суперкомпьютер, использующий java от HotSpot, но с нестандартным барьером на

чтение



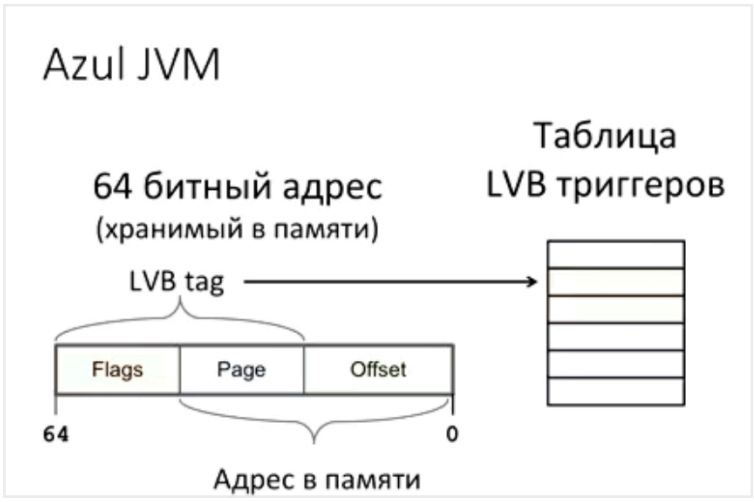


В HotSpot все потоки должны остановиться, у azul все потоки должны отметится, что прочитали актуальное состояние, перефильтровали свои регистры. Данная операция может выполняться асинхронно.

Адрес хранимый в памяти представляет собой своеобразную структуру.
Вся память разбита на 2МБ страницу, есть смещение внутри страницы, адрес страницы, биты не участвующие в адресации, например маркирующие.

Флаг и номер страницы представляет собой тэг, в памяти существует таблица тегов, в которой на каждый тег true или false.

Как работает барьер? Читает указатель, смотрит в таблицу, если true, выполняет барьер, иначе ничего не делает. Таким образом процессор выполняет проверку барьера и при этом исполняет код. Барьер срабатывает редко, поэтому не накладывает ограничений на производительность.

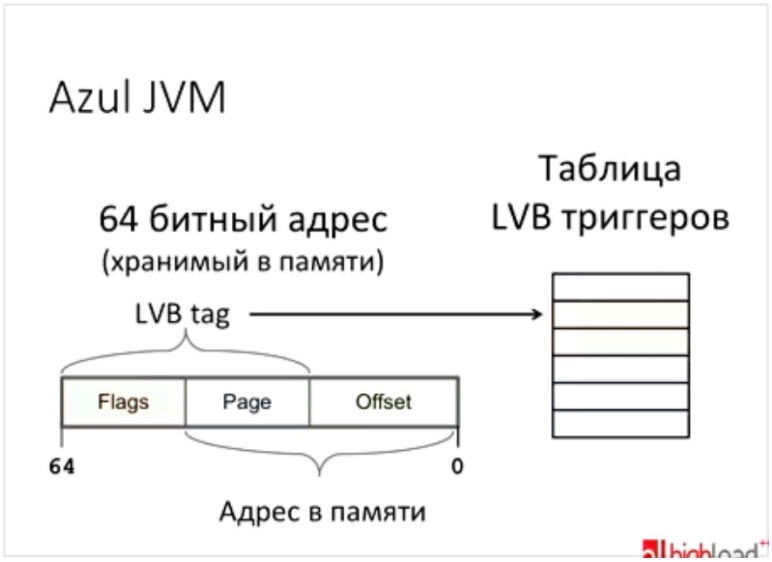


Однако для проверки необходимости запуска цикла сборки по старому поколению необходимо выполнить маркировку, которая требует остановки приложения, пусть и гораздо более быстрого, чем в HotSpot

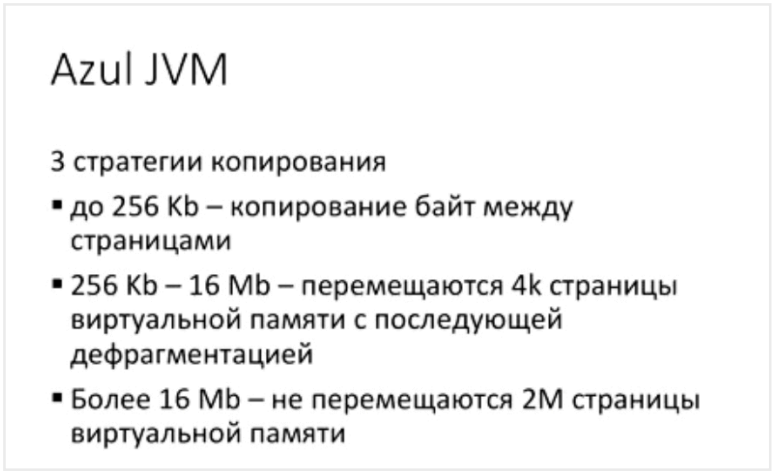
На AZUL нельзя было запускать ничего кроме java и использовались специфичные процессоры.

В последующих версиях они это решили используя процессоры intel.

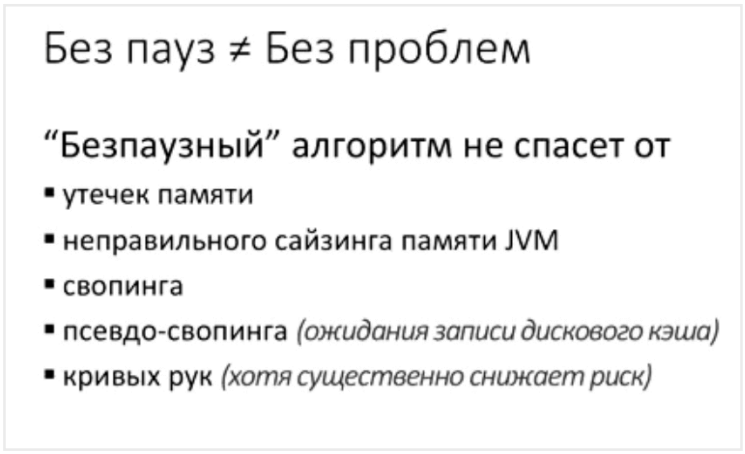
Была проблема при переходе с процессоров vega на intel, что процессоры vega позволяли хранить в адресе несколько незначащих битов. В архитектуре x86 такого нет.
Чтобы решить эту проблему азул берет все возможные комбинации этого поля флагов и реплицирует адресное пространство виртуальной памяти. Независимо от того какие биты будут в поле флагов, после преобразования через таблицу адресов микропроцессор получит один и тот же физический адрес в памяти.



Также при такой архитектуре есть проблема копирования больших объектов.



Можно подытожить, что этот алгоритм не является алгоритмом сборки в режиме реально времени



Почему все используют hotSpot а не zing? Zing сложнее настраивать, уходит больше времени на настройку. Вероятны сложности при переходе с одной jvm на другую. Zing - путь вертикально масштабирования. Если зинг не будет успевать очищать память без пауз, то они начнутся. Сборщик мусора предугадывать есть ли мусор в куче. Это достигается за счет анализа динамики. Если приложение аллоцирует 1гб, из которых долгощивущих 10%, то при сбросе кэша с этим возникает проблема.
Один из способов избавиться от проблем со сборкой мусора - добавить больше памяти.