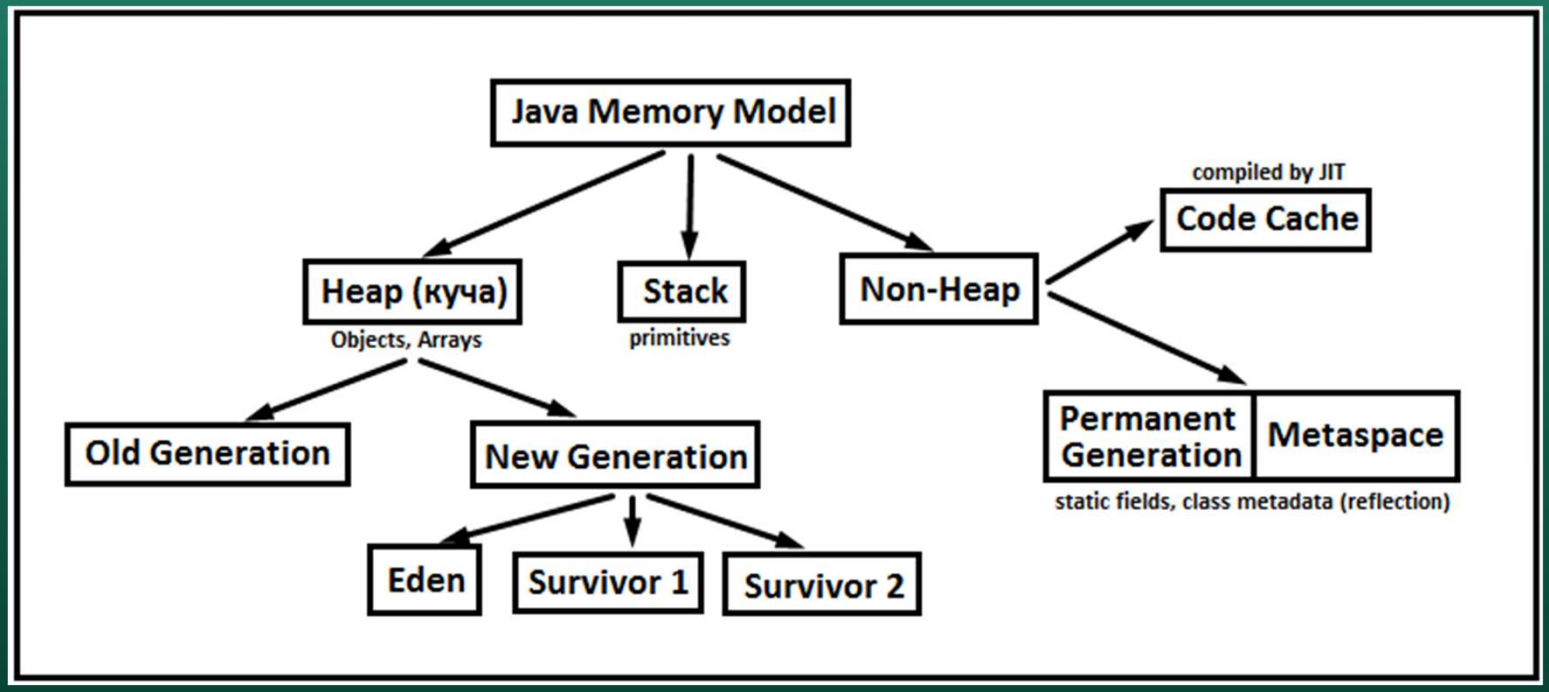
Java Memory Model, Garbage Collectors

КОММУНАЛЬНЫЕ СЛУЖБЫ ВАШИХ ПРИЛОЖЕНИЙ

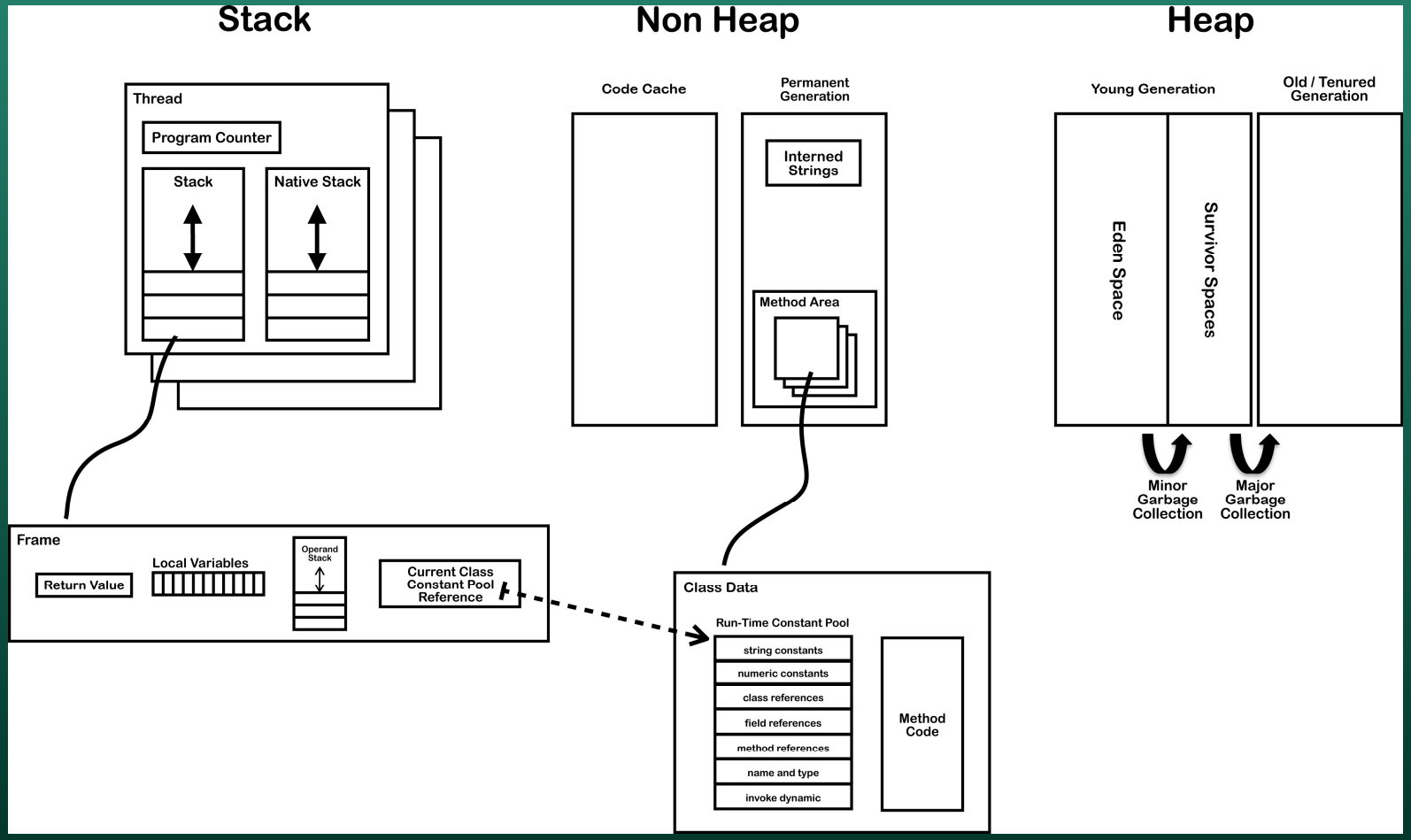
Коротко о JMM



*

* https://eyakubovskiy.ru/2020/06/08/pamyat-prilojeniya-v-java/

И ~~за что~~ зачем нам все это?

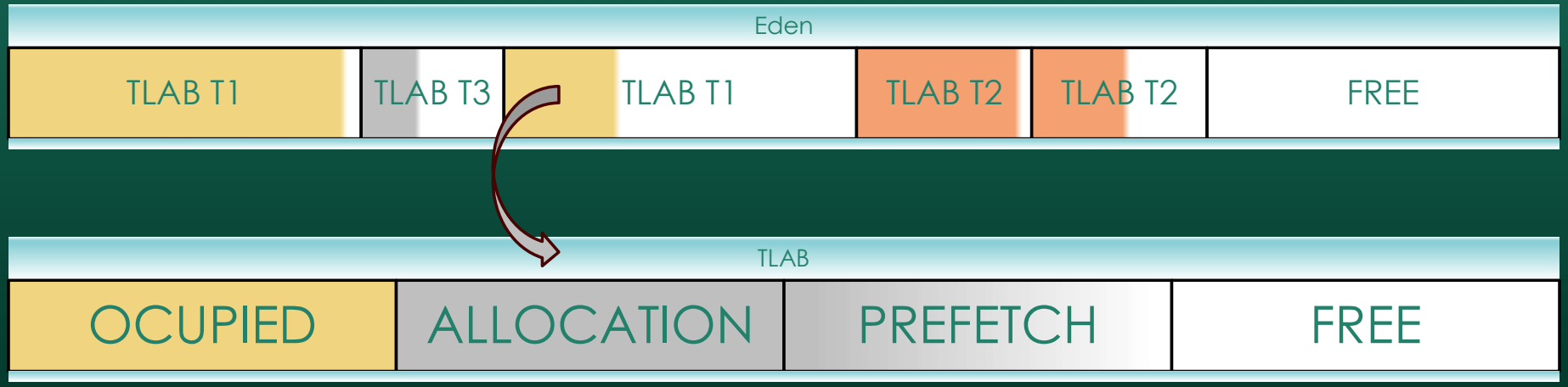


* https://habr.com/ru/articles/739338/

Подробнее про кучу или как происходит аллокация объектов.

Аллокация новых объектов в куче

TLAB – thread-local allocation buffer – область памяти в куче, зарезервированная ЭКСКЛЮЗИВНО для конкретного потока. Это значит, что доступ к TLAB происходит при отсутствии синхронизации.



* https://habr.com/ru/articles/332708/

Сборка мусора или управление памятью?

- Все объекты в куче делятся на младшее поколение(young generation/nursery) и старшее поколение (old generation).
- В процессе малой сборки мусора (minor GC) очищается только младшее поколение.
- В процессе основной сборки мусора (major GC) очистка происходит только в старшем поколении.
- Во время полной сборки мусора (full GC) удаляются объекты обоих поколений
- В безопасных точках (safepoint) программы GC может инициировать stop-the-world (STW) – остановку всех потоков программы для безопасной сборки мусора.

Методы сбора мусора

Метод подсчёта ссылок. Для каждого объекта в куче создаётся счётчик ссылок. Его показания меняются, когда ссылки на объект добавляются или удаляются. Сборщик мусора обходит все объекты и оставляет только те, у которых счётчик больше нуля. Остальные отправляются в мусорную корзину.

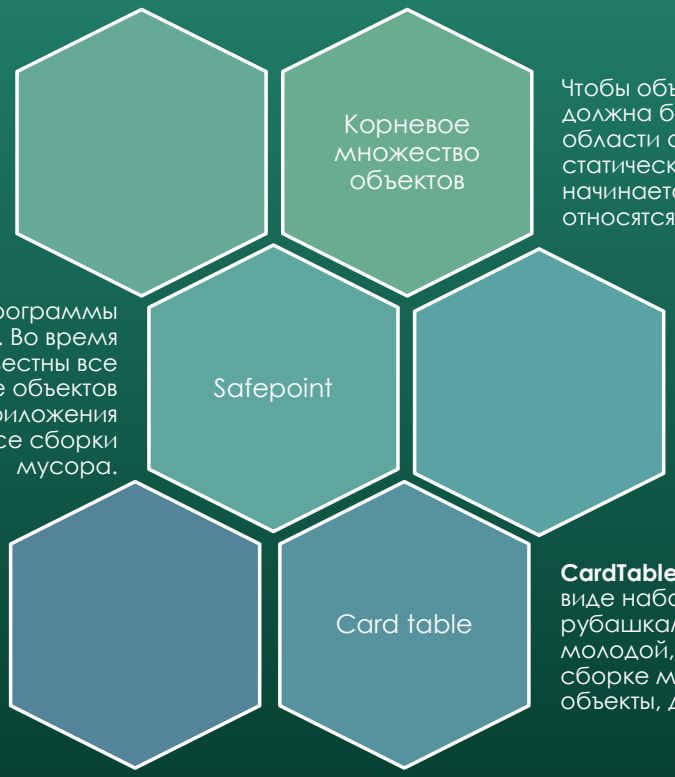
Метод флагов. Его ещё называют методом маркировки и выметания (от англ. *mark and sweep*). Его суть заключается в том, что для каждого объекта в памяти хранится бит, указывающий на достижимость.

Объекты помечаются достижимыми если к ним имеется доступ из объектов **корневого множества (GC roots)**.

Метод карточного стола (от англ. *Card table*). Этот метод считается наиболее эффективным на текущий момент.

Сборщик мусора может убирать не всю кучу, а только её часть. Но при этом ему нужно понять, ссылаются ли объекты старшего поколения на объекты младшего поколения.

…Помедленнее, я записываю!



Чтобы объект считался достижимым, на него должна быть ссылка или из стека, или из области старшего поколения, или из области статической памяти. Именно в этих областях начинается поиск достижимых объектов. Они относятся к **корневому множеству объектов.**

Остановки происходят в специальных местах программы — **безопасных точках** (от англ. *safepoint*). Во время выполнения программы в таких точках известны все корневые множества объектов и содержимое объектов кучи согласовано. Таким образом, данные приложения не будут потеряны или повреждены в процессе сборки мусора.

CardTable представляет объекты старшего поколения в виде набора карт. Изначально карты повёрнуты рубашками вверх. Если старший объект ссылается на молодой, карта переворачивается. И при следующей сборке мусора на достижимость проверяются только объекты, доступные «из-под этой карты».

Реализации GC *

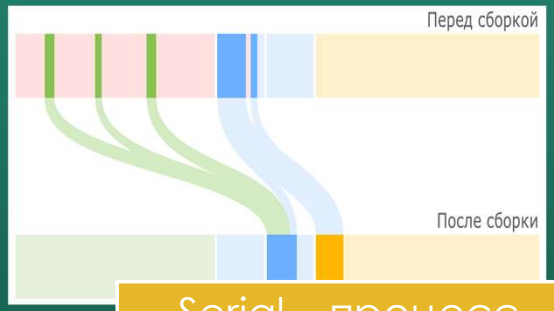
Serial
&
Parallel

CMS
&
G1

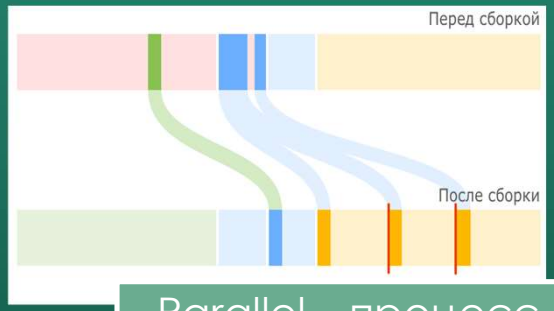
ZGC &
Shenandoah

* https://habr.com/ru/articles/269621/

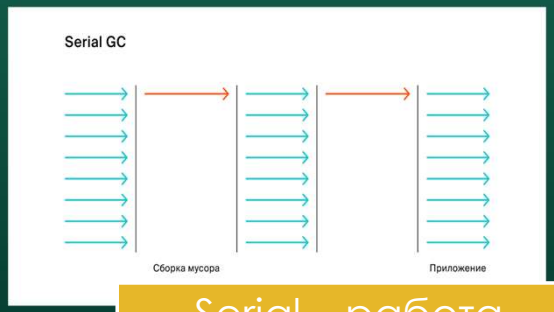
Serial & Parallel



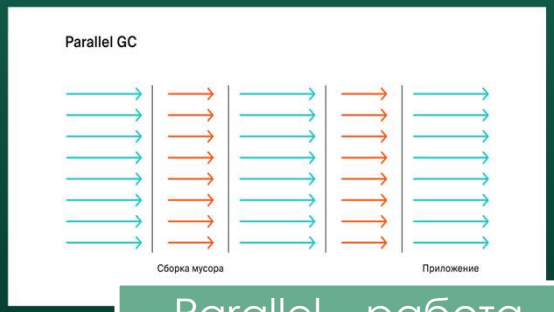
Serial – процесс сборки



Parallel – процесс сборки

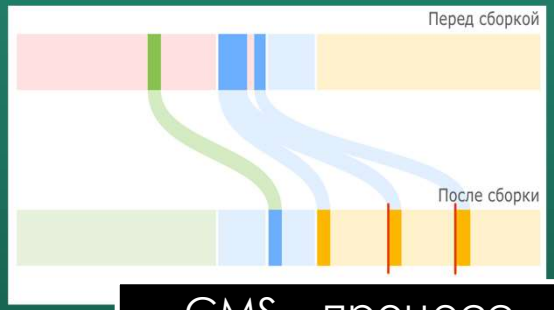


Serial – работа потоков

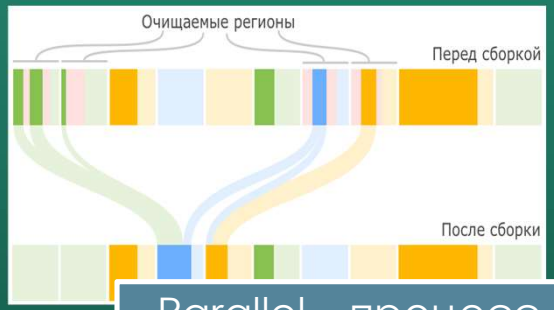


Parallel – работа потоков

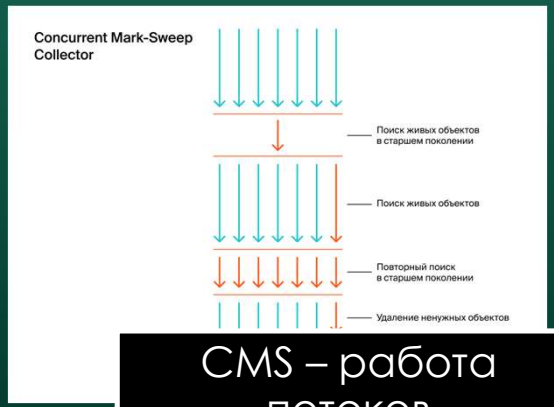
CMS & G1



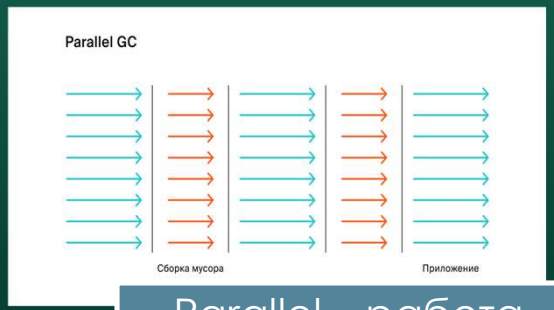
CMS – процесс сборки



Parallel – процесс сборки



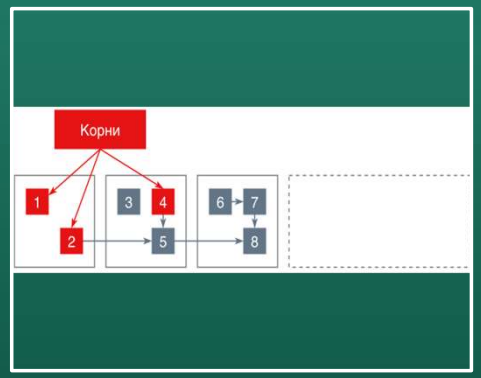
CMS – работа потоков



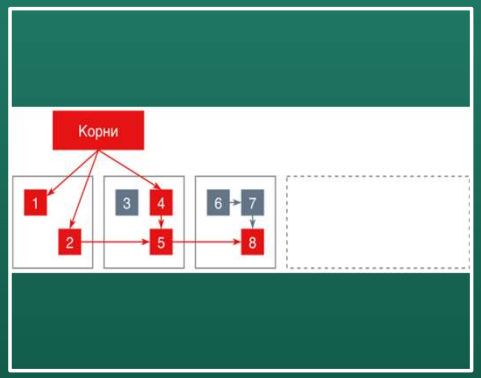
Parallel – работа потоков

ZGC

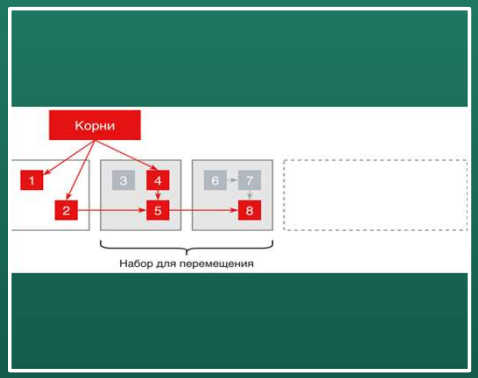
Pause Mark Start



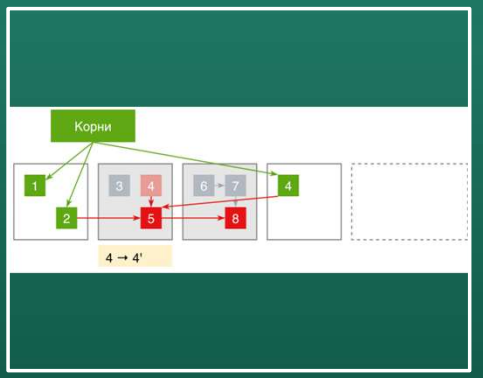
Concurrent Map, Pause Mark End



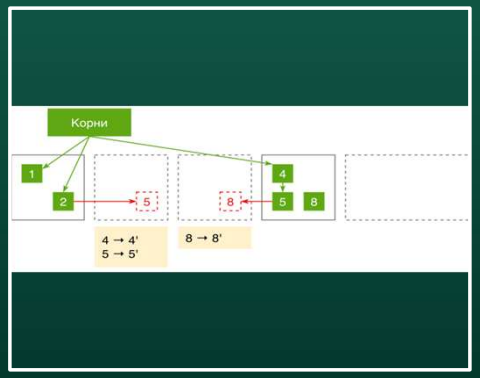
Concurrent Prepare for Relocate



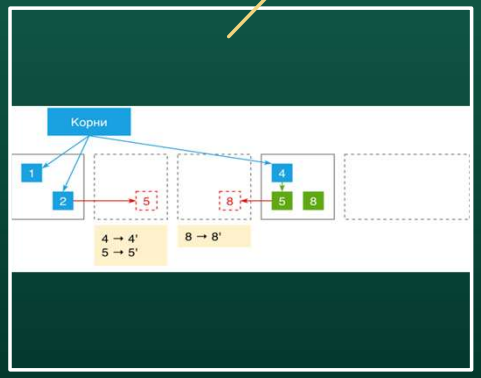
Pause Relocate Start



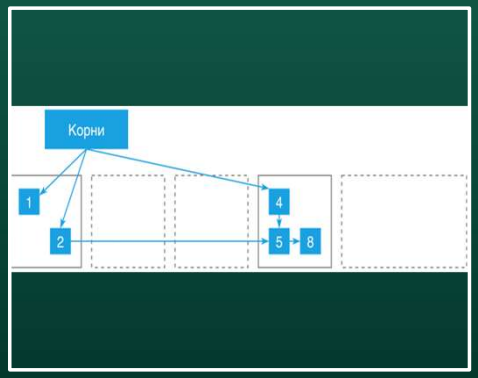
Concurrent Relocate



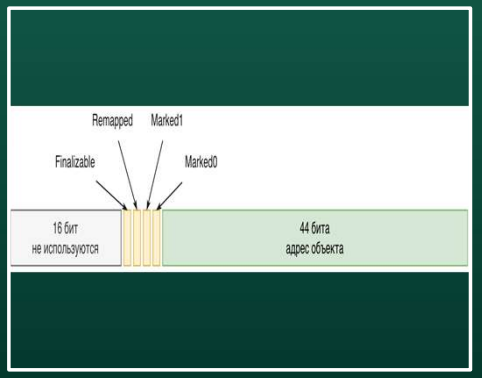
Concurrent Remap



Начало следующего цикла



Структура ссылки



Оператор new

- При использовании оператора new JVM выделяет место и создает объект в куче.
- Из-за этого сравнение по ссылке экземпляров некоторых классов может давать различные результаты.

JMM и многопоточность*

- **Instructions Reordering:** инструкции в потоке необязательно выполняются в том же порядке, в каком они написаны в коде, однако JAVA дает гарантию их **as-if-serial** выполнения.
- **Memory reordering:** существует несколько типов memory reodering: LoadLoad, LoadStore, StoreStore, StoreLoad
  - Если программа не синхронизирована, то разрешены все переупорядочивания. Если программа правильно синхронизирована, запрещены все переупорядочивания
  - Если программа не синхронизирована, то memory order, неконсистентный с program order, валиден с точки зрения JMM. Если программа правильно синхронизирована, то валиден только консистентный порядок
- **Data race** возникает тогда, когда с shared данными работает одновременно два или больше тредов, где как минимум один из них пишет и их действия не синхронизированы. Для действий в гонке не гарантируется никакого консистентного memory order, поэтому не стоит удивляться неожиданным результатам.
- **Happens-before** определяется как отношение между двумя действиями:
  - Пусть есть поток T1 и поток T2 (необязательно отличающийся от потока T1) и действия x и y, выполняемые в потоках T1 и T2 соответственно
  - Если x happens-before y, то во время выполнения y треду T2 будут видны все изменения, выполняемые в x тредом T1

* https://habr.com/ru/articles/685518/

Спасибо за внимание!