Тема F. Render tree.

Во время отображения страницы браузером часто происходят визуальные изменения, возникающие из-за работы псевдоклассов типа :hover, работы CSS-анимаций и JavaScript-анимаций, программного изменения дерева DOM (создания, удаления, изменения элементов DOM), изменения размеров окна браузера и др.

Браузеру необходимо иметь механизм, позволяющий эффективно перерисовать в окне браузера изменившиеся элементы. Этот механизм основан на использовании **render tree**.

Приводимые далее сведения не являются документальными, а являются скорее упрощённой моделью, позволяющей понять принципы функционирования браузера и сделать обоснованные и проверяемые выводы касательно производительности работы браузера.

Render tree — дерево элементов, по структуре почти совпадающее с деревом DOM.

Для каждого элемента render tree хранит:

- позицию этого элемента относительно его родителя (или окна/документа/предка);
- размер элемента;
- вычисленные стилевые свойства элемента;
- растр элемента (растровое изображение того же размера что и элемент, на котором отрисовывается собственно элемент; например, для элемента IMG это растр с картинкой, для элемента P это прозрачный растр, на котором нарисован текст);
- растр ветки (растровое изображение того же размера что и элемент, на котором наложены растр самого элемента и растры его дочерних элементов; например, для элемента P это прозрачный растр, на котором нарисован текст, а также все изображения, вставленные в абзац).

Когда в DOM вносятся какие-либо изменения, браузер помечает некоторые реквизиты элементов render tree как нуждающиеся в обновлении.
Например, при установке для элемента DIV новой высоты (style.height) браузер:

- *помечает его размер как нуждающийся в пересчёте;*
- *помечает растр элемента и растр ветки как нуждающиеся в перерисовке.*

Никаких пересчётов и перерисовок в данный момент не происходит. Если будут внесены другие изменения в DOM, реквизиты других элементов также будут помечены как нуждающиеся в пересчёте и/или перерисовке.

Когда наступает animation frame (а то есть момент, когда необходимо отобразить изменённую страницу в окне браузера), браузер пересчитывает всё что нужно пересчитать и перерисовывает всё что нужно перерисовать.
Например, для вышеуказанного элемента DIV:

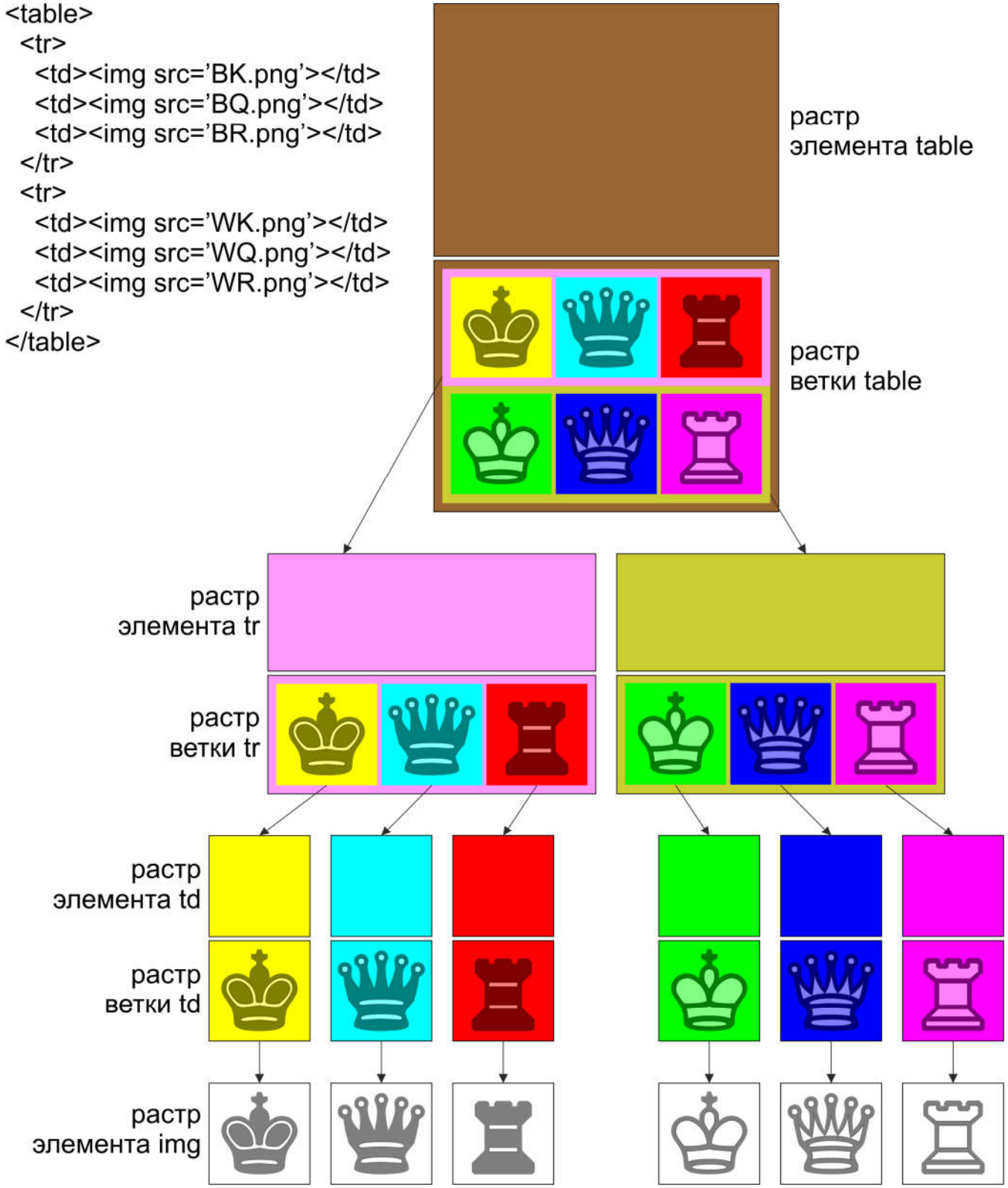
- *рассчитывается новая высота, с учётом высоты и размеров дочерних элементов DIV (получившаяся высота может и не совпадать с установленной программно);*
- *пересоздаётся (т.к. изменился размер) и перерисовывается растр элемента;*
- *пересоздаётся и перерисовывается растр ветки (для этого достаточно взять растр элемента и наложить на него растры веток дочерних элементов этого DIV,* ***не углубляясь*** *в элементы ещё более низких уровней, т.к. они уже входят в растры веток дочерних элементов).*

Таким образом, если внутри данного DIV было скажем три элемента P, а в них — 1000 элементов SPAN и IMG, изменение высоты DIV приведёт к одному пересчёту высоты, одной отрисовке и трём операциям наложения растров.

В некоторых случаях в целях эффективности дерево DOM может быть разбито на несколько render tree, поддерживаемых браузером независимо.

Пример render tree

```
<table>
  <tr>
    <td><img src='BK.png'></td>
    <td><img src='BQ.png'></td>
    <td><img src='BR.png'></td>
  </tr>
  <tr>
    <td><img src='WK.png'></td>
    <td><img src='WQ.png'></td>
    <td><img src='WR.png'></td>
  </tr>
</table>
```



При открытии страницы браузер выполняет следующие действия.

Название этапа: **render tree build**
Действия:
- разбор структуры документа;
- вычисление взаимного порядка и взаимной подчинённости элементов.

От чего зависит расход ресурсов:
- быстродействие и расход памяти зависят от количества элементов в документе.

Название этапа: **layout (или flow)**
Действия:
- вычисление размеров и координат элементов относительно их родителей.

Название этапа: **paint (или style)**
Действия:
- создание растров элементов и растров веток — изображений размера, вычисленного на этапе layout;
- отрисовка элементов на растрах элементов (только сам элемент, не включая его дочерние элементы);
- отрисовка растров веток (на растр элемента накладываются растры дочерних веток).

От чего зависит расход ресурсов:
- расход памяти зависит от площади элементов и глубины дерева;
- быстродействие зависит от количества перерисовываемых элементов, их площади и сложности их отрисовки, а также от глубины дерева (глубина дерева имеет значение, т.к. каждый уровень дерева порождает очередной растр ветки, и каждый элемент в конечном итоге и занимает место, и занимает время на каждом растре ветки).

Название этапа: **composite**
(в конечном счёте каждое render tree компонуется в одно изображение — растр слоя; GPU на этапе «работа с GPU» оперирует именно растрами слоёв)
Действия:
- наложение растров всех элементов на растры слоёв (фактически данный этап, в простом случае, есть просто копирование растра ветки корневого элемента BODY на растр слоя).

Название этапа: **работа с GPU**
Действия:
- отправка растров слоёв в GPU;
- отправка других команд в GPU, в т.ч. команды на отображение слоя в окне браузера.

От чего зависит расход ресурсов:
- быстродействие зависит от скорости шины между CPU и GPU и площади отправляемых слоёв (т.е. чаще всего от размера документа).
  ВАЖНО: мобильные устройства часто имеют медленную шину между CPU и GPU, и шина становится узким местом в производительности.

Рекомендации по повышению эффективности:
не злоупотреблять вынесением элементов на слои GPU — мобильные устройства имеют ограниченную память GPU и её переполнение может вызвать значительное замедление отображения.

При изменении страницы (изменении стилевых свойств элементов, удалении или создании элементов, изменении подчинённости элементов) браузер выполняет следующие действия.

Название этапа: **render tree invalidation**
Действия:

- пометка «пересчитать» элементов, положение или размер которых нужно перерасчитать, т.к. они возможно изменились;
- размер элемента обычно определяется размером его дочерних элементов, и положение элемента зависит от размеров и положений предыдущих элементов того же родителя, поэтому пометка «пересчитать» любого элемента обычно вызывает пометку «пересчитать» всех последующих элементов у того же родителя, самого родителя, а следовательно и следующих после родителя элементов и т.д.;
- пометка «перерисовать» элементов, внешний вид которых изменён (возможно и без изменения положения или размера).

От чего зависит расход ресурсов:

- быстродействие зависит от количества элементов, которые нужно пометить, а то есть от глубины изменившихся элементов в дереве DOM и от количества зависящих от них элементов.

Рекомендации по повышению эффективности:

- к одному родителю не помещать сотни дочерних элементов, объединять их по возможности в промежуточные блочные теги;
- в случае множественных изменений сложной страницы — отсоединять ветку изменяемых элементов от DOM (например через *элемент*.removeChild), делать изменения и снова присоединять ветку к DOM (*элемент*.appendChild); или убирать элемент только из render tree (например через style.display='none' **— что не отображается, то не входит в render tree**), делать изменения и снова возвращать элемент в render tree (style.display='');
- если анимируется (или часто меняется) размер или положение элемента, на уровне вёрстки обеспечить независимость от них положения и размеров других элементов, например через position:absolute.

Никаких дальнейших расчётов и отрисовок браузер не производит, пока не наступит animation frame, т.е. момент для обновления отображаемой страницы. Это позволяет группировать изменения, не дублировать расчёты и отрисовки без необходимости.

Когда наступает animation frame, браузер выполняет следующие действия.

Название этапа: **layout**
Действия:

Для элементов, помеченных «пересчитать»:

- вычисление размеров и координат тех элементов, у которых они могли измениться, т.е. у тех что помечены «пересчитать» на этапе render tree invalidation;
- если в результате вычисления оказалось что размеры и положение элемента не изменились — никаких дополнительных пометок не делается (самый частый вариант);
- если в результате вычисления оказалось, что изменился размер элемента — элемент помечается «перерисовать», т.к. однозначно должен выглядеть иначе чем был;
- если в результате вычисления оказалось, что изменилось положение или размер элемента — он помечается «изменён», т.е. он должен быть перевыведен в окно браузера в конечном счёте;
- в любом случае, пометка «пересчитать» у элементов снимается.

ВАЖНО: этап layout также **частично** отрабатывает, не дожидаясь animation frame, если происходит программное чтение свойств offsetTop/Left/Width/Height, clientTop/Left/Width/Height, scrollTop/Left/Width/Height либо вызов метода getComputedStyle().
От чего зависит расход ресурсов:

- от количества обсчитываемых элементов.

Рекомендации по повышению эффективности:

- группировать вносимые изменения и не перемежать изменения чтением стилевых свойств или положения/размеров, в противном случае частичный layout будет выполнен многократно; т.е. есть смысл сначала прочитать все нужные стилевые свойства и положения/размеры в промежуточные переменные, а потом единовременно вносить изменения;
- стараться анимировать абсолютно и фиксированно позиционированные элементы, т.к. их положение и размер не влияют на размеры их родителя, а следовательно и всех остальных элементов страницы.

Название этапа: **paint**
Действия:

Для элементов, помеченных «перерисовать»:

- изменение размеров растров элементов, если размеры элемента изменились на этапе layout;
- отрисовка элементов на растрах элементов (только сам элемент, не включая его дочерние элементы);
- отрисовка на растрах веток (растр элемента с наложением растров веток дочерних элементов);
- в любом случае, пометка «перерисовать» у элементов снимается.

От чего зависит расход ресурсов:

- быстродействие зависит от количества перерисовываемых элементов, их площади и сложности их отрисовки.

Название этапа: **composite**
Действия:

- копирование растра ветки элемента BODY (т.е. растра всей веб-страницы) в растр слоя.

От чего зависит расход ресурсов:

- в некоторой степени быстродействие зависит от площади веб-страницы (почти вся работа уже выполнена на этапе paint).

Название этапа: **работа с GPU**
Действия:

- отправка растров изменившихся слоёв в GPU;
- отправка других команд в GPU, в т.ч. команды на отображение слоя в окне браузера.

От чего зависит расход ресурсов:

- быстродействие зависит от скорости шины между CPU и GPU и площади отправляемых (изменившихся) слоёв (т.е. чаще всего от размера документа).
  ВАЖНО: мобильные устройства часто имеют медленную шину между CPU и GPU, и шина становится узким местом в производительности.

Рекомендации по повышению эффективности:

- минимизировать площадь часто изменяемых элементов и вынести их на отдельные слои;
- по возможности, анимировать только стилевые свойства transform, opacity, filter.

Элемент выделяется на отдельный GPU-слой, если:

- для него задана 3D-трансформация стилевым свойством transform: translate3d/translateZ/scale3d/scaleZ/rotate3d/rotateZ/matrix3d/perspective;
- к нему применены CSS-фильтры (стилевое свойство filter);
- для него **сейчас работает** CSS-анимация стилевого свойства opacity или transform;
- он является элементом <video> и поддерживается его аппаратное ускорение;
- он является элементом <canvas> с 3D (WebGL), или элементом <canvas> с 2D и поддерживается его аппаратное ускорение (см. например chrome://gpu/);
- для его отображения требуется плагин браузера (flash, silverlight);
- он оказался отделён от остальных элементов страницы другим GPU-слоем по z-index.

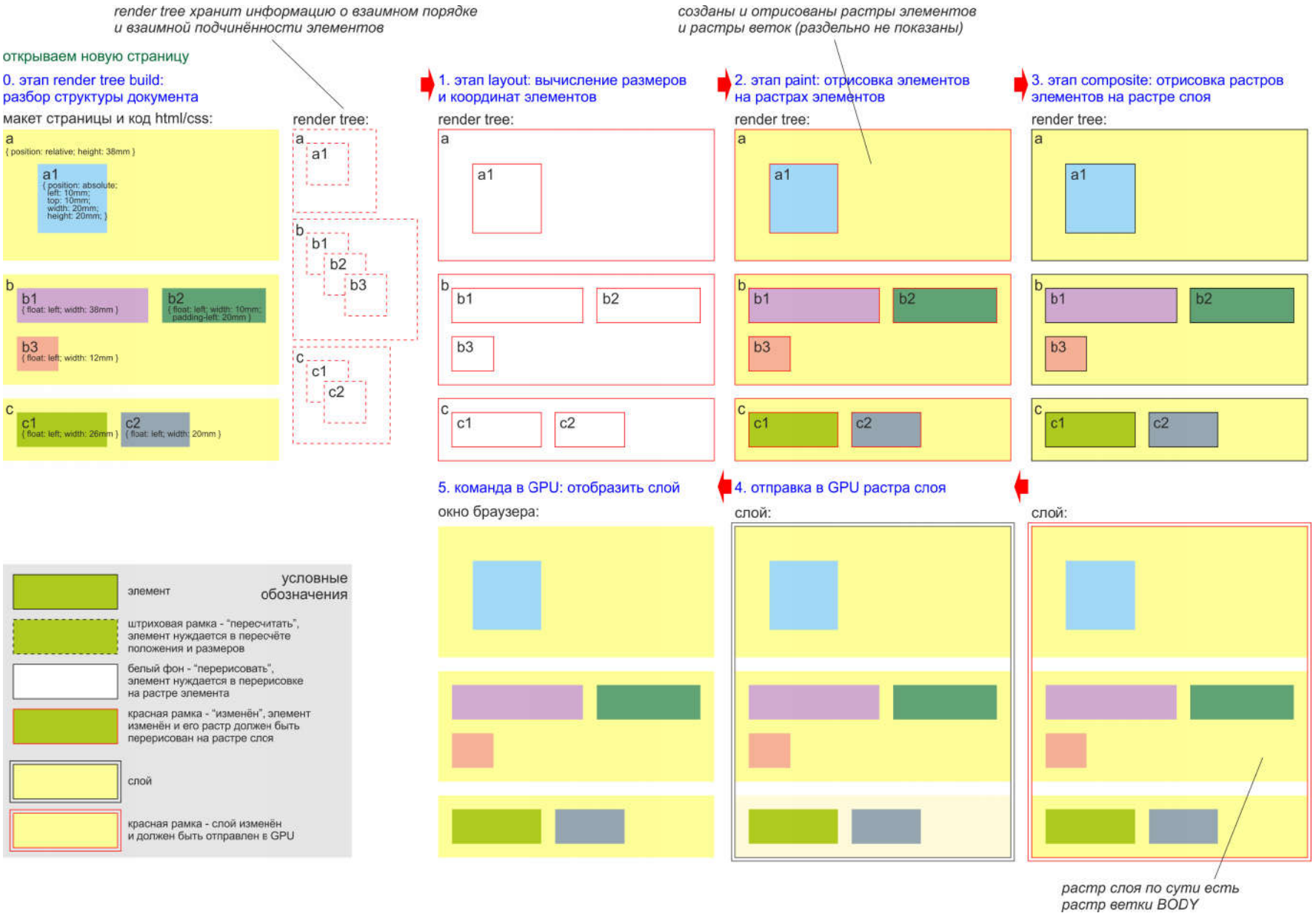
Следовательно, любой элемент может быть искусственно вынесен на отдельный GPU-слой применением к нему стилевого свойства **transform: translateZ(0).**
Более семантичный способ, работающий в современных браузерах **— will-change: transform.**
Лучше всего использовать оба способа сразу:

```
transform: translateZ(0);
will-change: transform;
```

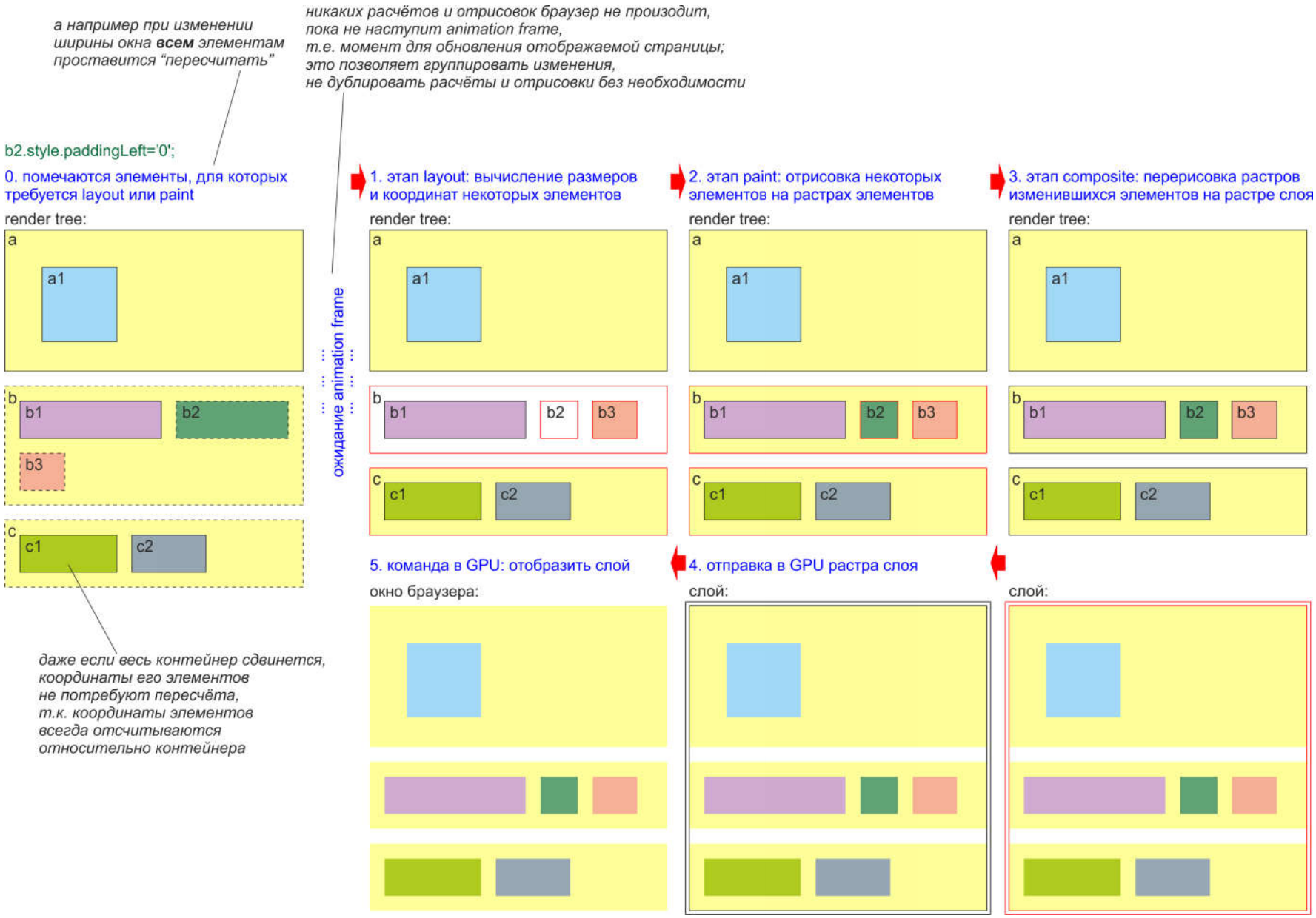
Далее на иллюстрациях, для упрощения схем, приняты следующие условности:

- показаны элементы только двух уровней; в реальности уровней больше, и всегда есть элемент самого верхнего уровня BODY, который охватывает всё render tree;
- в render tree вместо растра элемента и растра ветки показано одно условное изображение, характеризующее метки «пересчитать», «перерисовать», «изменён» данного элемента.

page1.html — обычная веб-страница, открытие страницы

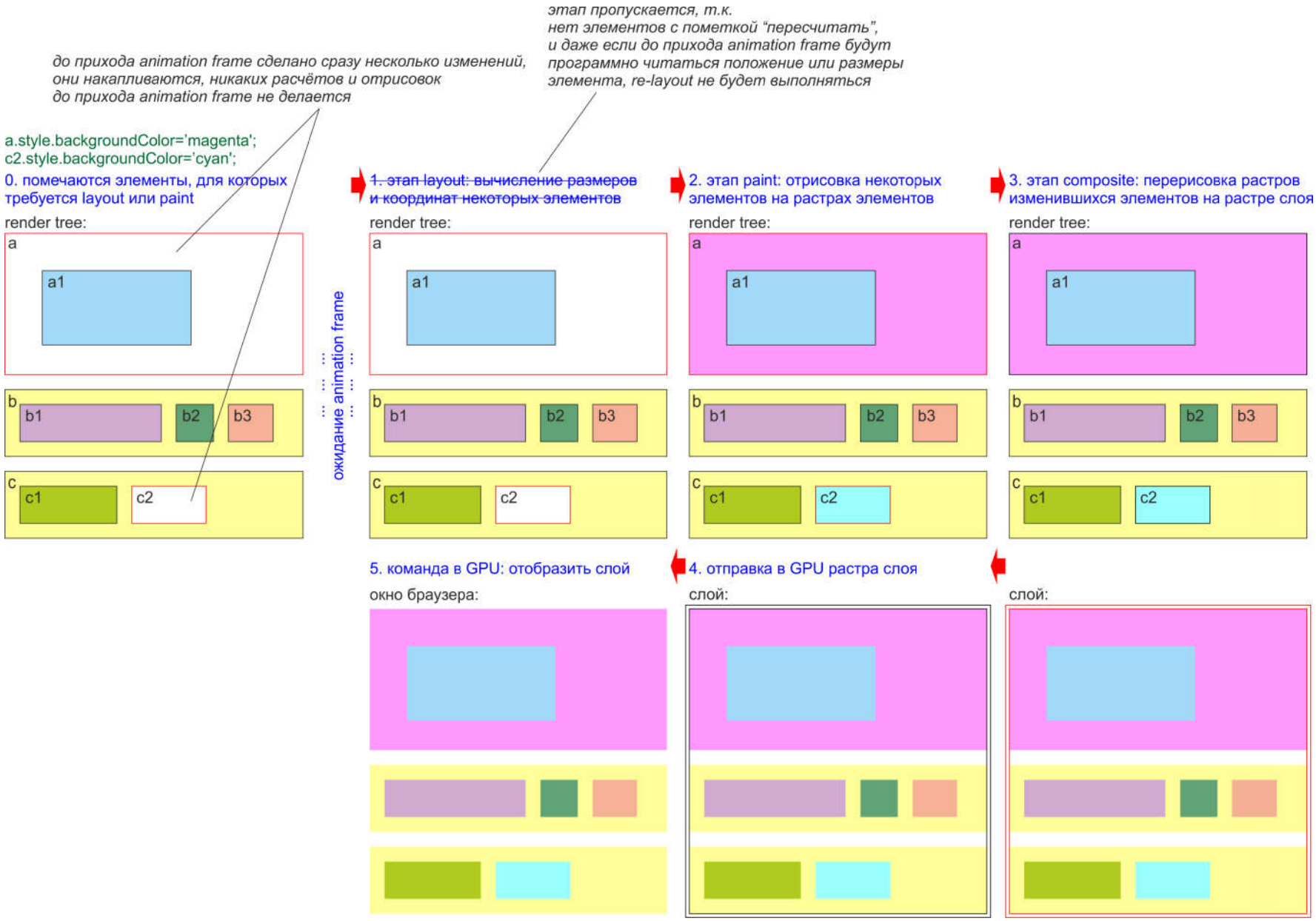


page1.html — изменяем размер элемента

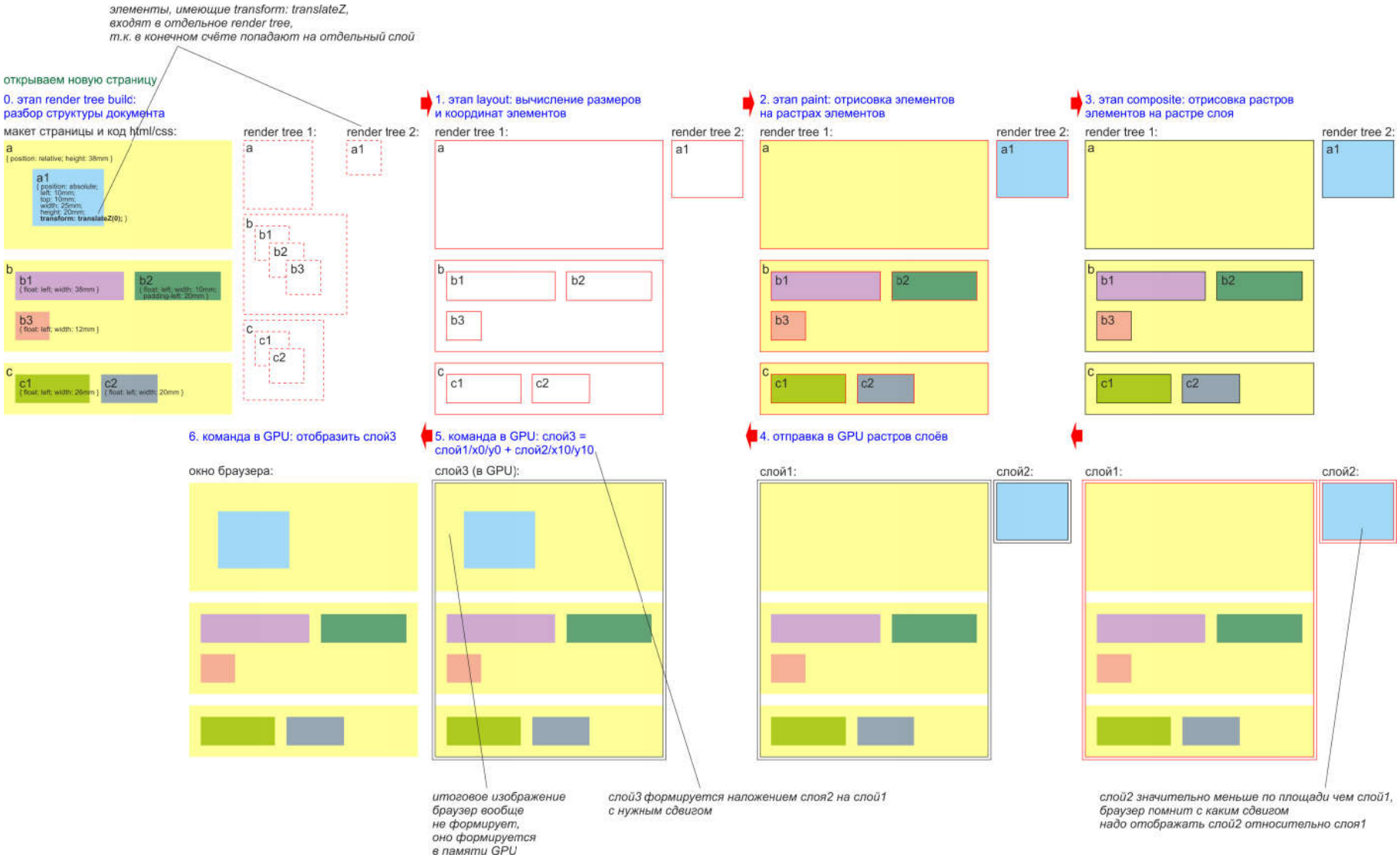


page1.html — изменяем размер абсолютно позиционированного элемента

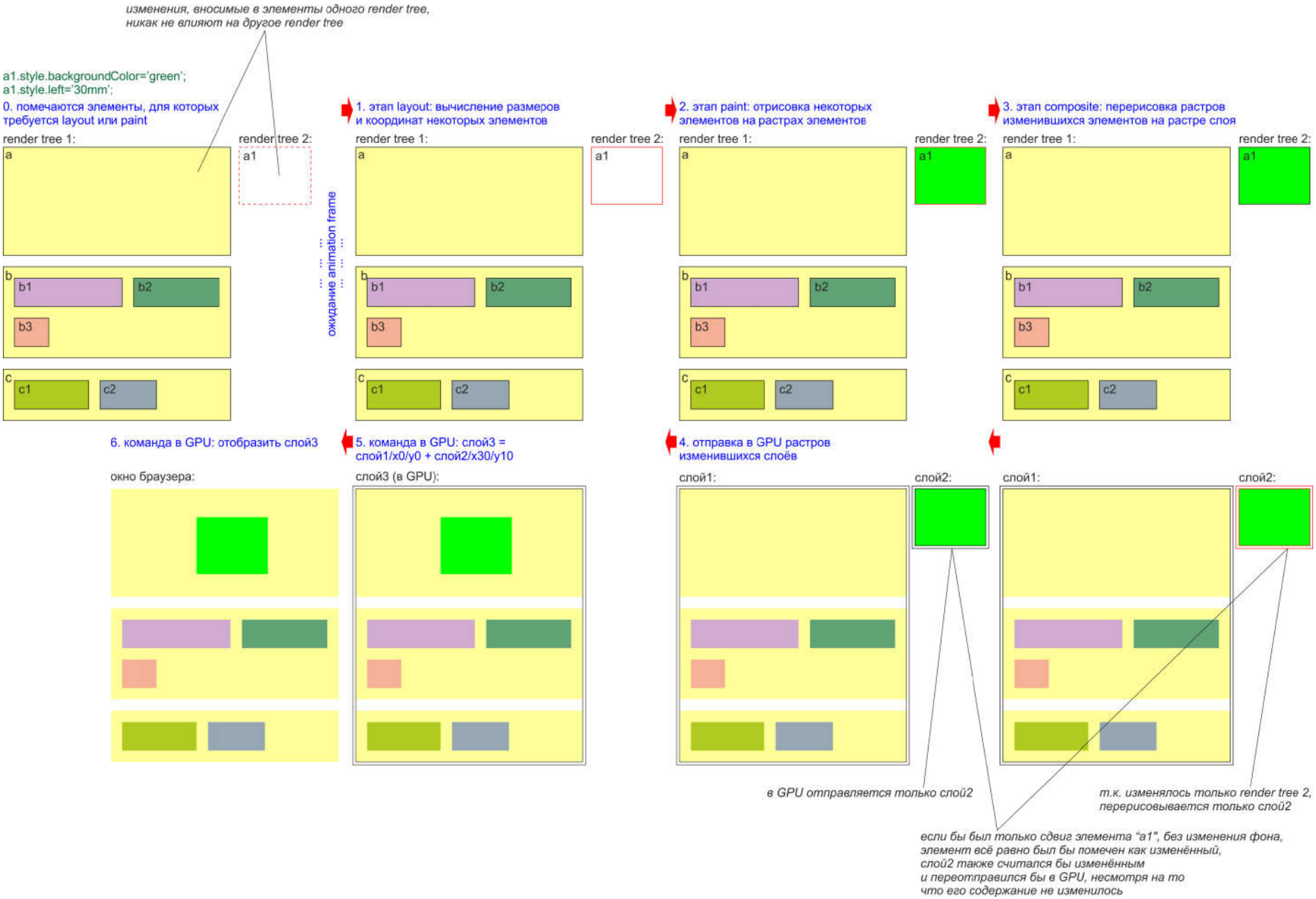
page1.html — изменяем заливку



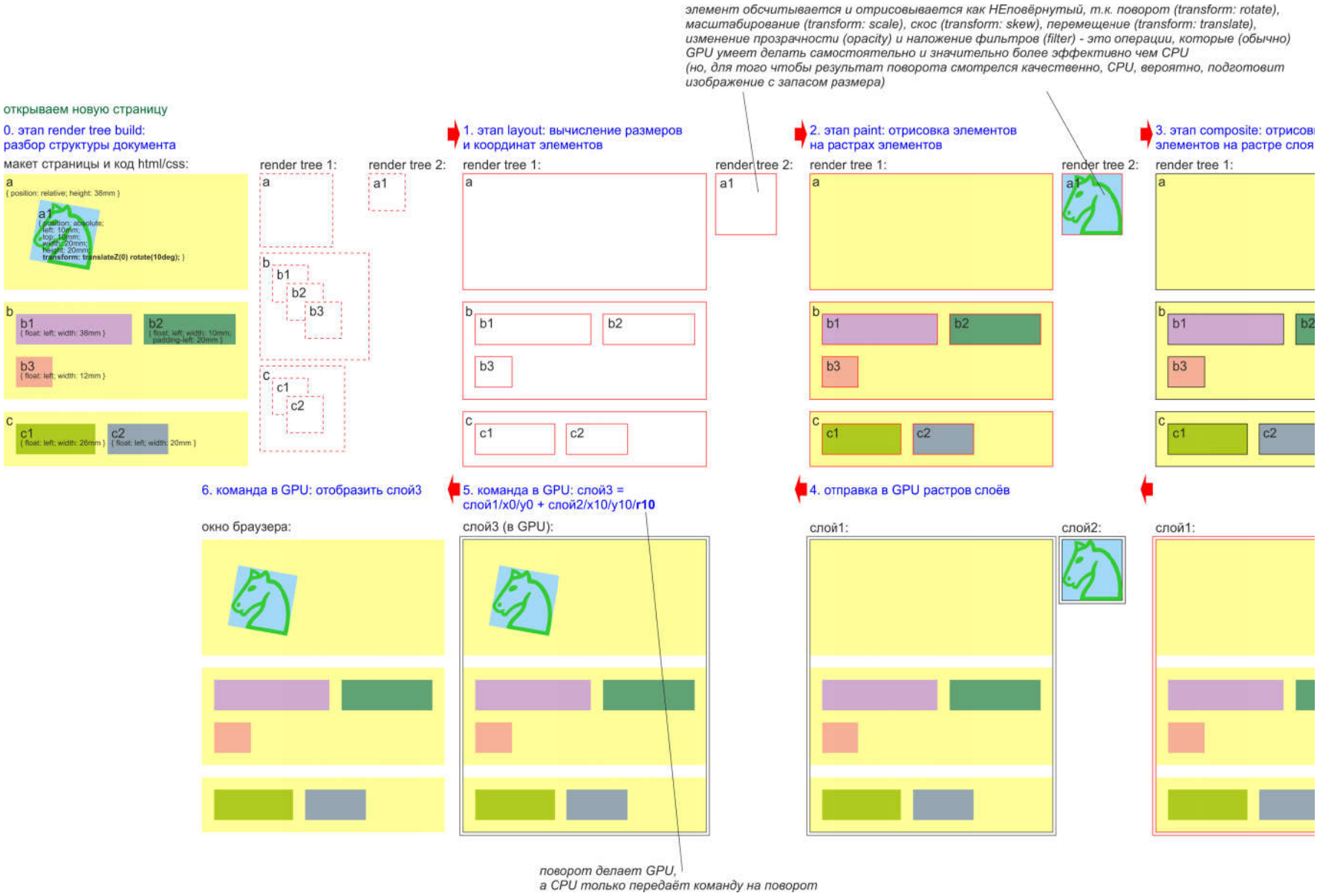
page2.html — веб-страница с вынесением элемента на отдельный слой



page2.html — изменяем элемент, находящийся на отдельном слое



page3.html — веб-страница с вынесением элемента на отдельный слой и применением к нему трансформации



page3.html — изменяем трансформацию элемента, находящегося на отдельном слое