

## 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`:*

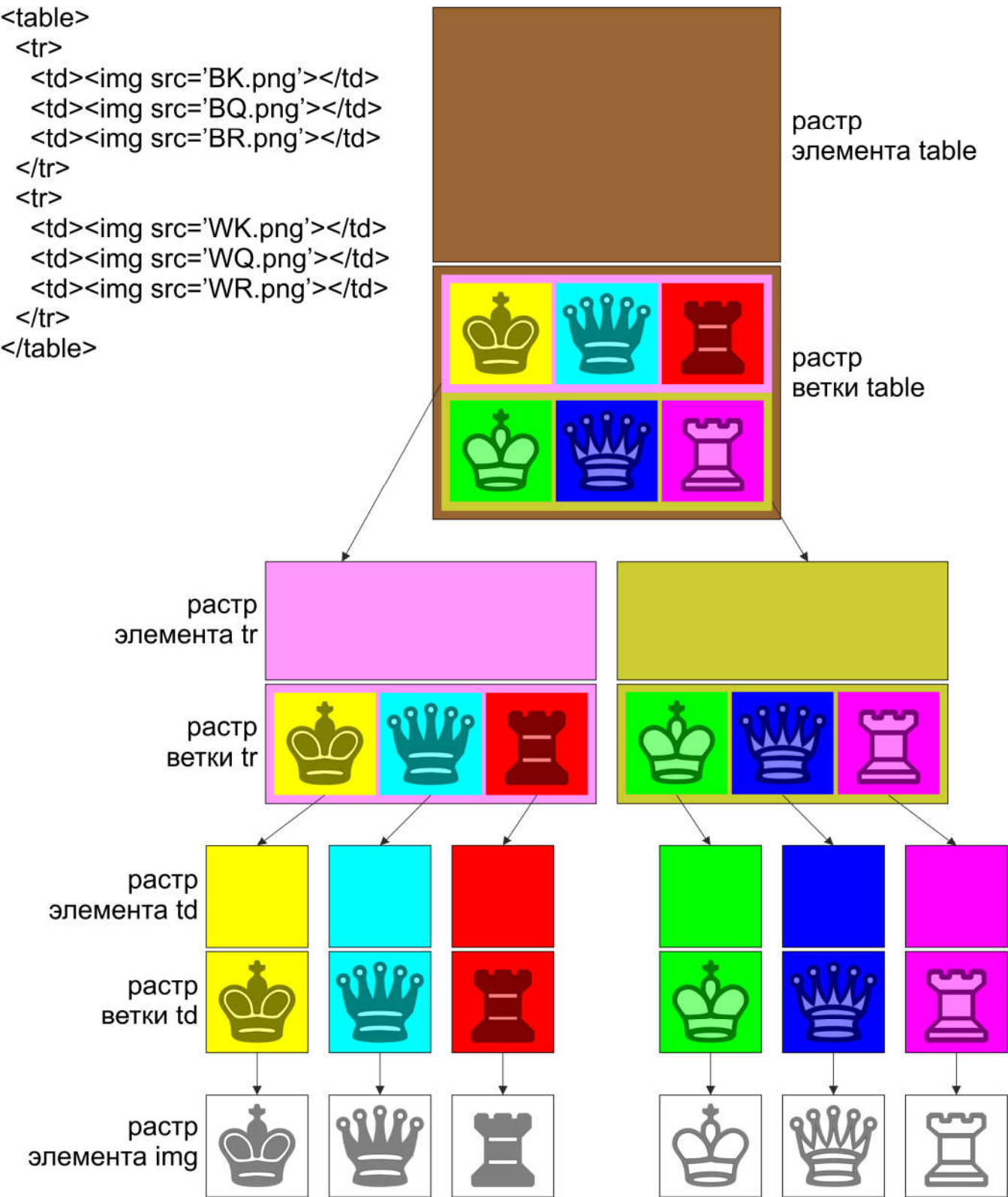
- *рассчитывается новая высота, с учётом высоты и размеров дочерних элементов `DIV` (получившаяся высота может и не совпадать с установленной программно);*
- *пересоздаётся (т.к. изменился размер) и перерисовывается растр элемента;*
- *пересоздаётся и перерисовывается растр ветки (для этого достаточно 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 обчитываемых 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 стилевых свойств 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 стилевых свойств **transform: translateZ(0)**.

Более семантический 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.

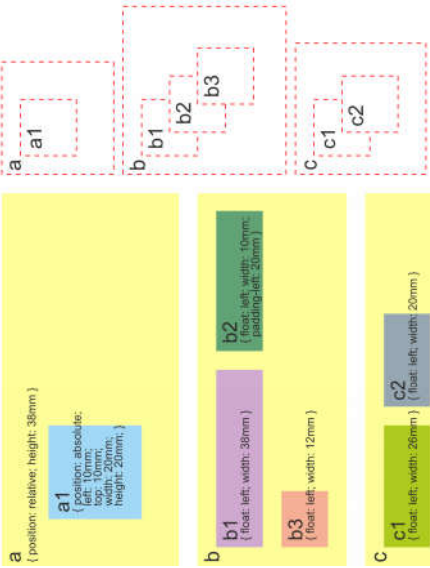
# page1.html — обычная веб-страница, открытие страницы

render tree хранит информацию о взаимном порядке и взаимной подчинённости элементов

открываем новую страницу

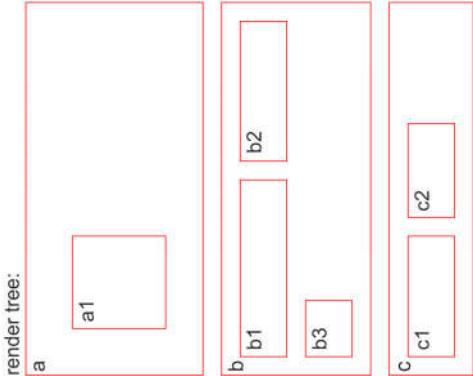
0. этап render tree build: разбор структуры документа

макет страницы и код html/css:

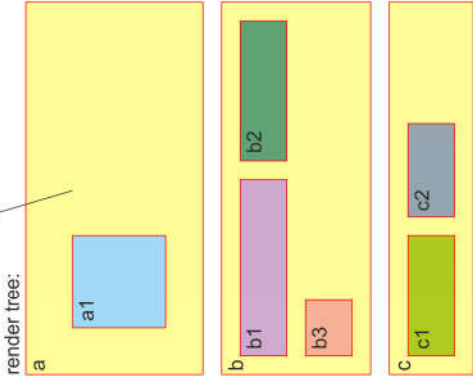


созданы и отрисованы растры элементов и растры веток (раздельно не показаны)

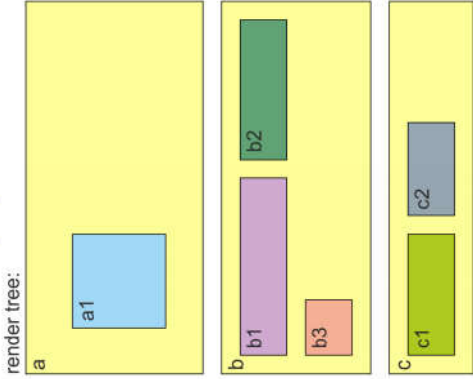
1. этап layout: вычисление размеров и координат элементов



2. этап paint: отрисовка элементов на растрах элементов

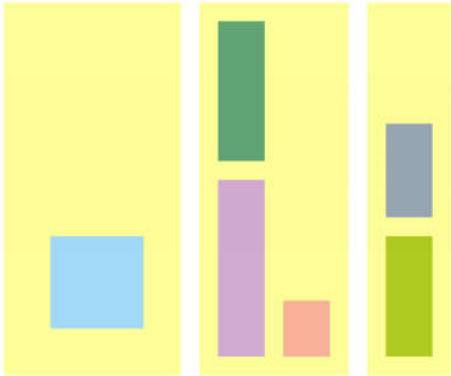


3. этап composite: отрисовка растров элементов на растре слоя



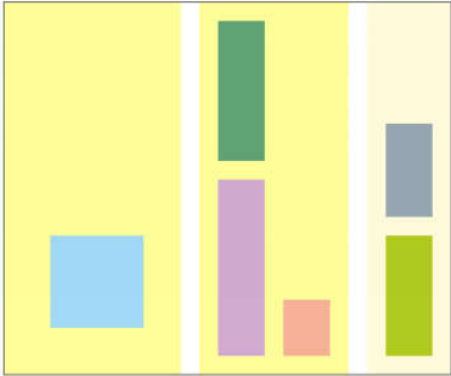
5. команда в GPU: отобразить слой

окно браузера:

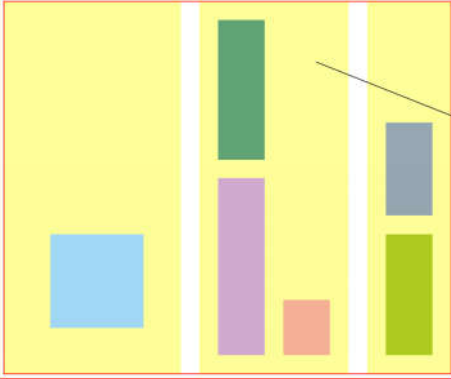


4. отправка в GPU растра слоя

слой:



слой:



растр слоя по сути есть растр ветки BODY

элемент	условные обозначения
	штриховая рамка - "пересчитать", элемент нуждается в пересчёте положения и размеров
	белый фон - "перерисовать", элемент нуждается в перерисовке на растре элемента
	красная рамка - "изменён", элемент изменён и его растр должен быть перерисован на растре слоя
	слой
	красная рамка - слой изменён и должен быть отправлен в GPU

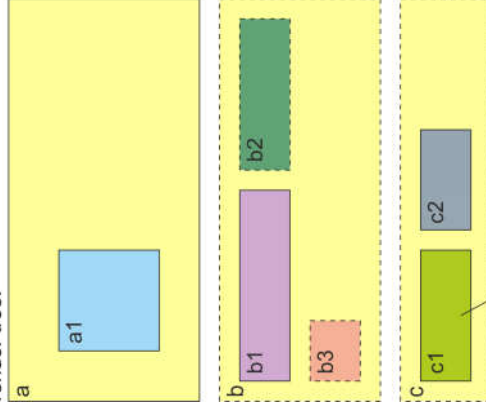
## page1.html — изменяем размер элемента

а например при изменении ширины окна **всем** элементам проставится "пересчитать" никаких 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

`b2.style.paddingLeft=0;`

0. помечаются элементы, для которых требуется layout или paint

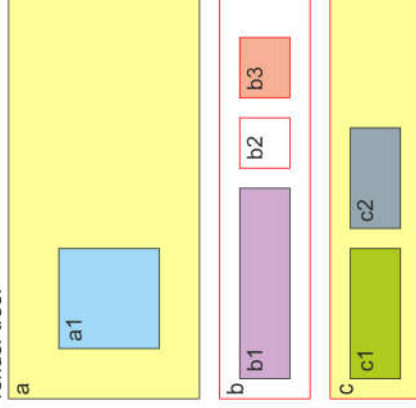
render tree:



ожидание animation frame

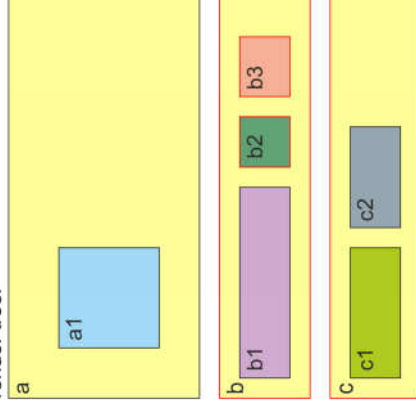
1. этап layout: вычисление размеров и координат некоторых элементов

render tree:



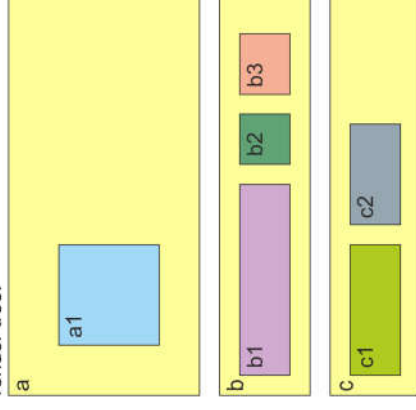
2. этап paint: отрисовка некоторых элементов на растрах элементов

render tree:



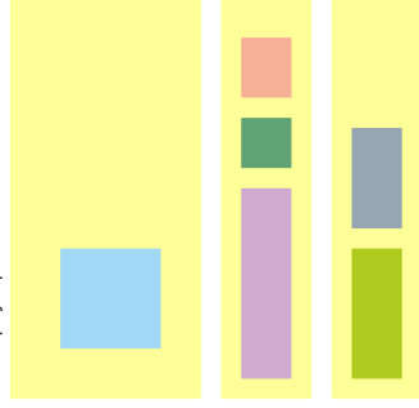
3. этап composite: перерисовка растров изменившихся элементов на растре слоя

render tree:



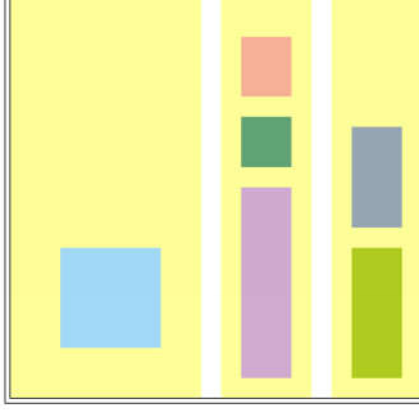
5. команда в GPU: отобразить слой

окно браузера:



4. отправка в GPU растра слоя

слой:



даже если весь контейнер сдвинется, координаты его элементов не потребуют пересчёта, т.к. координаты элементов всегда отсчитываются относительно контейнера

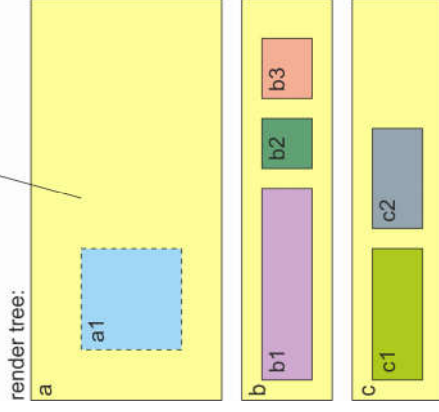


## page1.html — изменяем размер абсолютно позиционированного элемента

размер контейнера "a" не зависит  
от положения и размера элемента "a1",  
т.к. элемент "a1" спозиционирован абсолютно

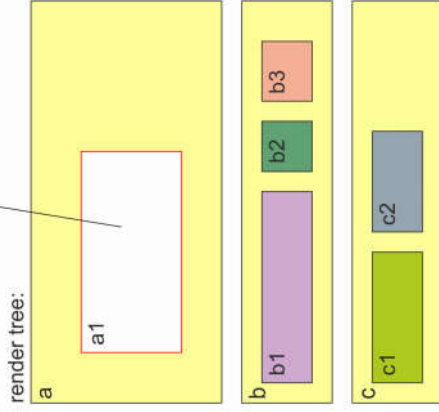
a1.style.width='40mm';

0. помечаются элементы, для которых  
требуется layout или paint

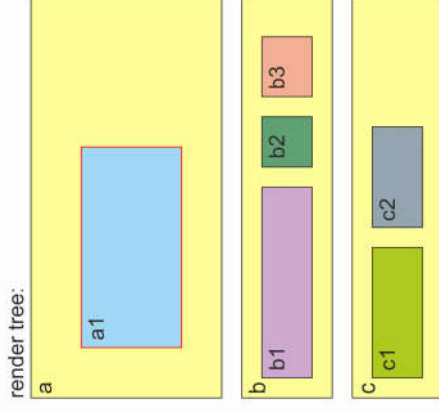


ожидание animation frame

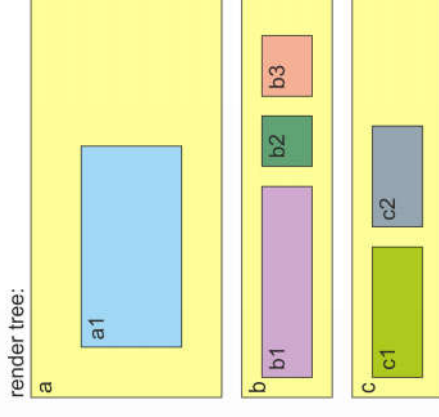
1. этап layout: вычисление размеров  
и координат некоторых элементов



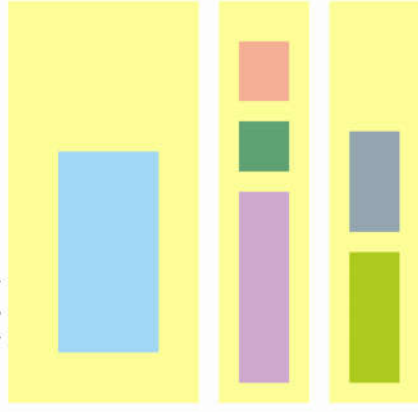
2. этап paint: отрисовка некоторых  
элементов на растрах элементов



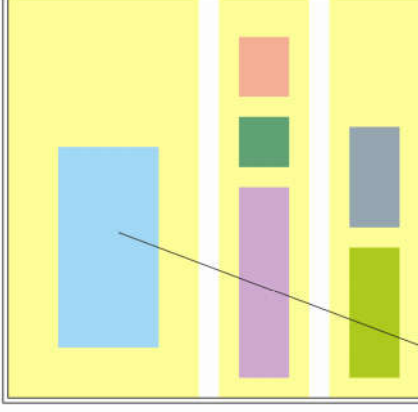
3. этап composite: перерисовка растров  
изменившихся элементов на растре слоя



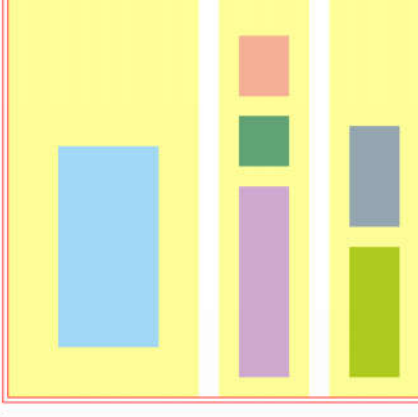
5. команда в GPU: отобразить слой  
окно браузера:



4. отправка в GPU растра слоя



слой:



несмотря на то что изменился небольшой  
кусочек слоя, слой заново полностью отправляется в GPU

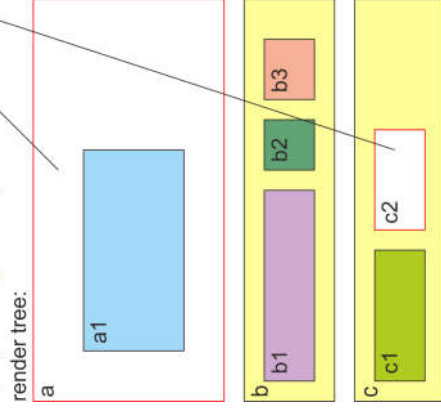


этап пропускается, т.к.  
нет элементов с пометкой "пересчитать",  
и даже если до прихода *animation frame* будут  
программно читаться положение или размеры  
элемента, *ge-layout* не будет выполняться

до прихода *animation frame* сделано сразу несколько изменений,  
они накапливаются, никаких расчётов и отрисовок  
до прихода *animation frame* не делается

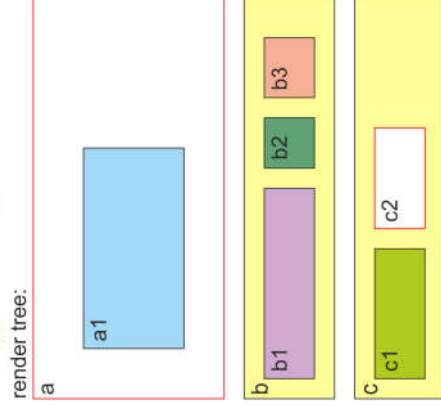
`a.style.backgroundColor='magenta';`  
`c2.style.backgroundColor='cyan';`

0. помечаются элементы, для которых  
требуется *layout* или *paint*

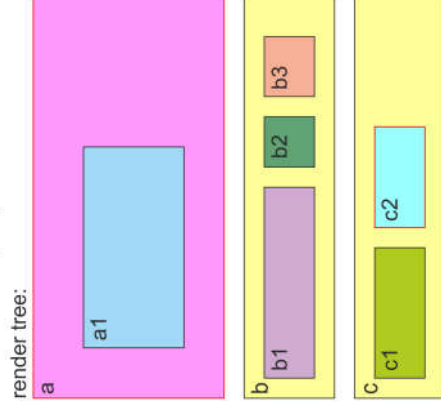


ожидание *animation frame*

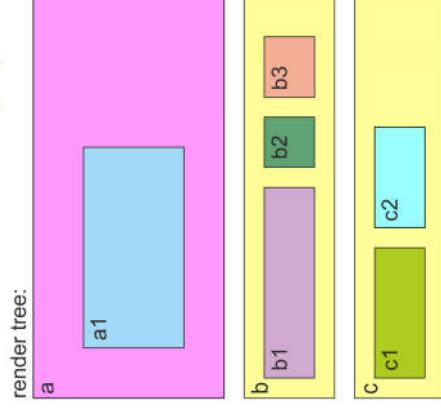
1-этап *layout*: вычисление размеров  
и координат некоторых элементов



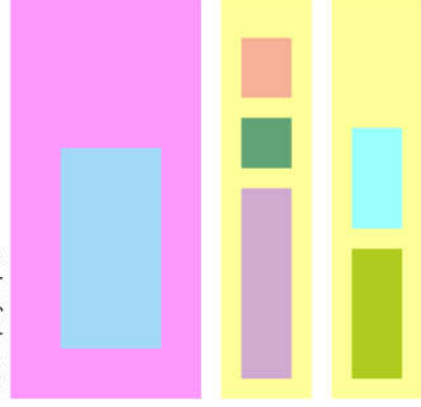
2. этап *paint*: отрисовка некоторых  
элементов на растрах элементов



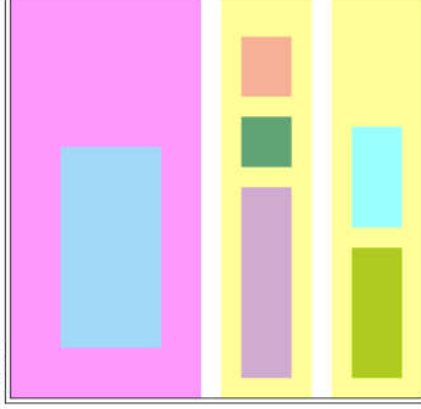
3. этап *composite*: перерисовка растров  
изменившихся элементов на растре слоя



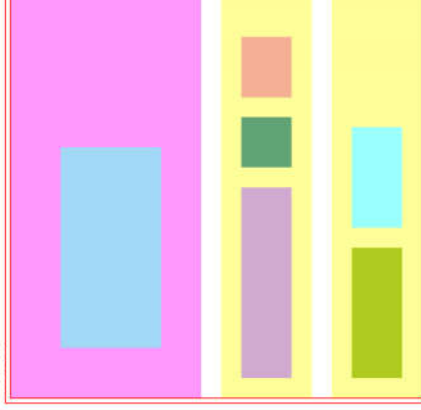
5. команда в GPU: отобразить слой  
окно браузера:



4. отправка в GPU растра слоя  
слой:



слой:



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

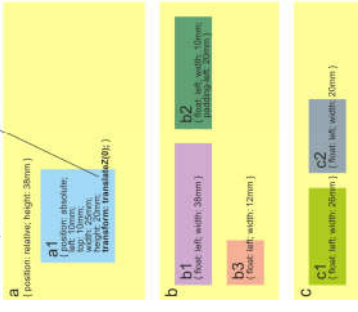
элементы, имеющие `transform: translateZ`,  
идут в отдельное `render tree`,  
т.к. в конечном счёте попадают на отдельный слой

открываем новую страницу

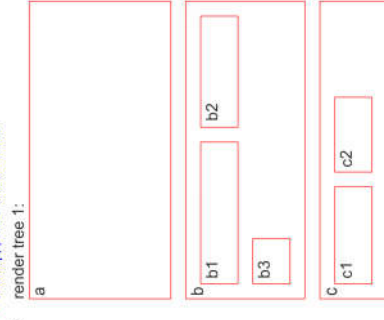
0. этап `render tree build`:

разбор структуры документа

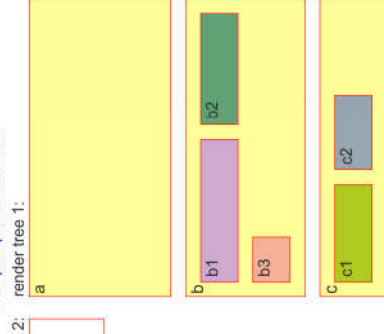
макет страницы и код `html/css`:



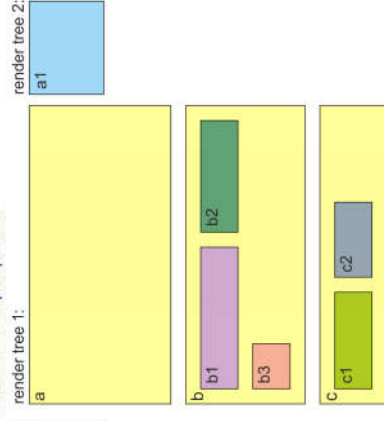
1. этап `layout`: вычисление размеров и координат элементов



2. этап `paint`: отрисовка элементов на растрах элементов

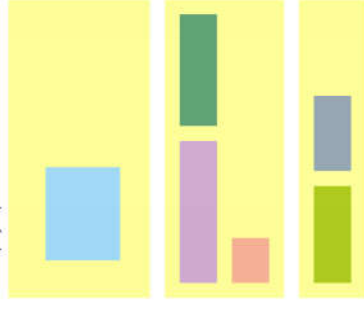


3. этап `composite`: отрисовка растров элементов на растре слоя



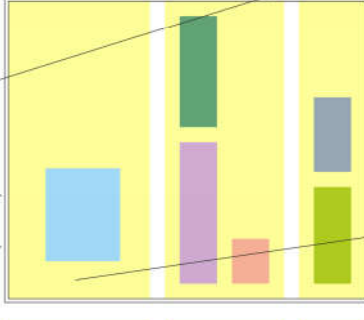
6. команда в GPU: отобразить слой3

окно браузера:



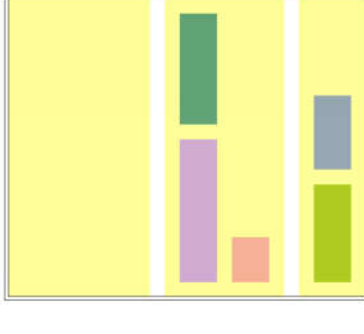
5. команда в GPU: `слой3 = слой1 * x0y0 + слой2 * x1y1`

слой3 (в GPU):

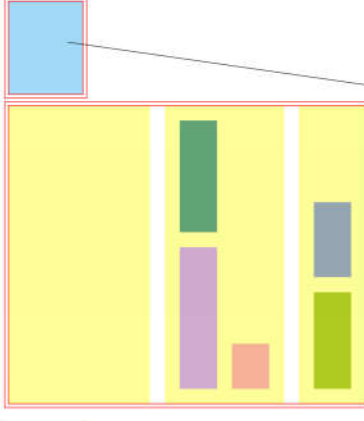


4. отправка в GPU растров слоёв

слой1:



слой2:



итоговое изображение  
браузер вообще  
не формирует,  
оно формируется  
в памяти GPU

слой3 формируется  
наложением слой2 на слой1  
с нужным сдвигом

слой2 значительно меньше по площади чем слой1,  
браузер помнит с каким сдвигом  
надо отображать слой2 относительно слоя1

## page2.html — изменяем элемент, находящийся на отдельном слое

изменения, вносимые в элементы одного *render tree*, никак не влияют на другое *render tree*

```
a1.style.backgroundColor='green';  
a1.style.left='300px';
```

0. помечаются элементы, для которых требуется layout или paint

1. этап layout: вычисление размеров и координат некоторых элементов

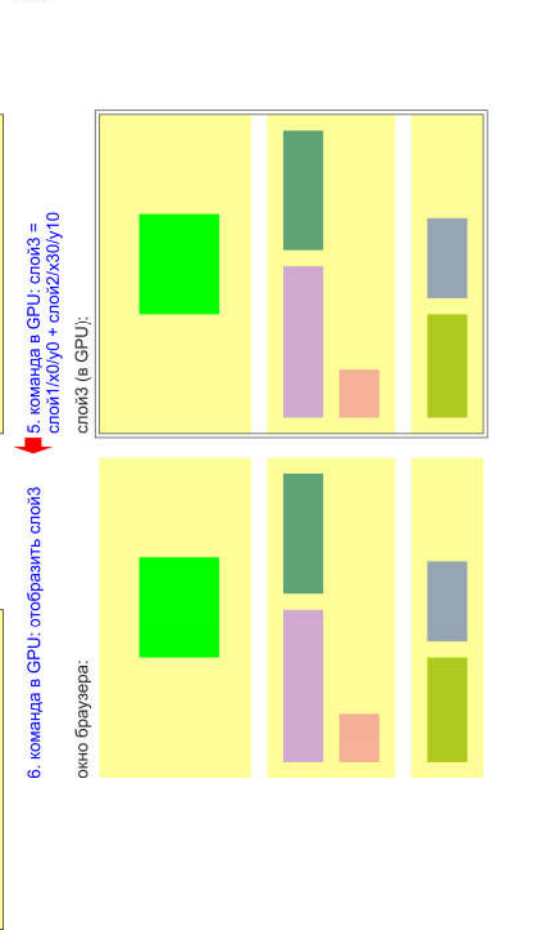
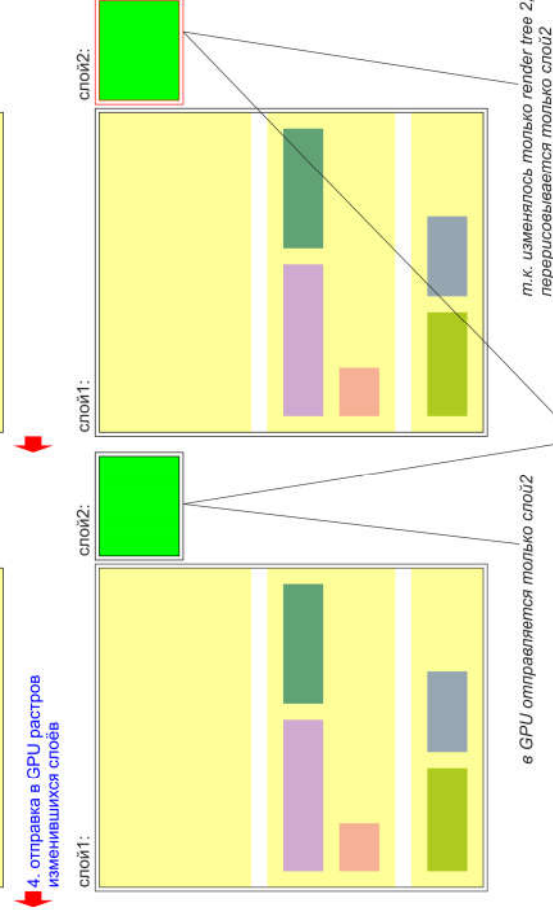
3. этап composite: перерисовка растров изменившихся элементов на растре слоя

2. этап paint: отрисовка некоторых элементов на растрах элементов

6. команда в GPU: отобразить слой3

4. отправка в GPU растров изменившихся слоев

5. команда в GPU: слой3 = слой1/x0/y0 + слой2/x30/y10



если бы был только сдвиг элемента "a1", без изменения фона, элемент всё равно был бы помечен как изменённый, слой2 также считался бы изменённым и перерисовывался бы в GPU, несмотря на то что его содержание не изменилось

в GPU отправляются только слой2

т.к. изменилось только render tree 2, перерисовывается только слой2

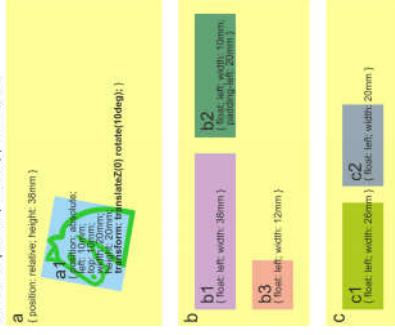
page3.html — веб-страница с вынесением элемента на отдельный слой и применением к нему трансформации

открываем новую страницу

0.37s render tree build:

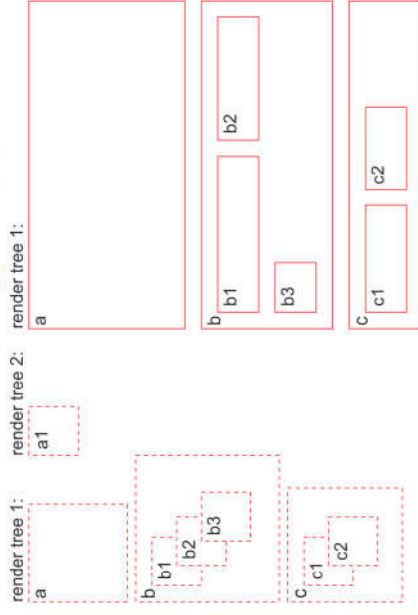
## разбор структуры документа

макет страницы и код `html/css`:



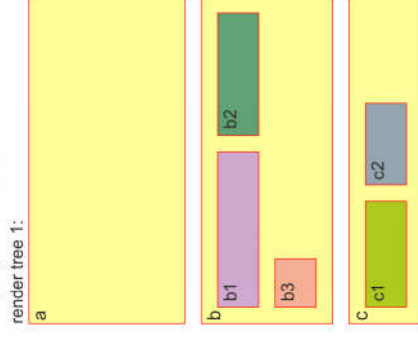
1. этап layout: вычисление размеров

и координат элементов



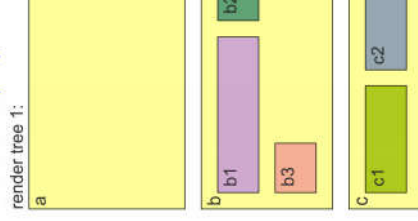
2. этап paint: отрисовка элементов

на растрах элементов

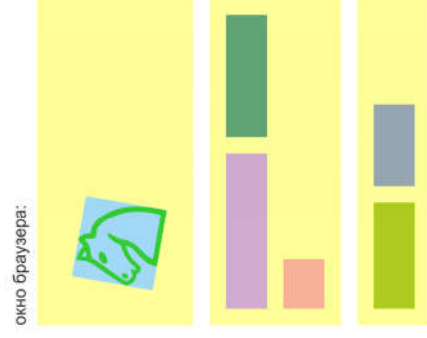


3. этап composite: отрисовки

элементов на

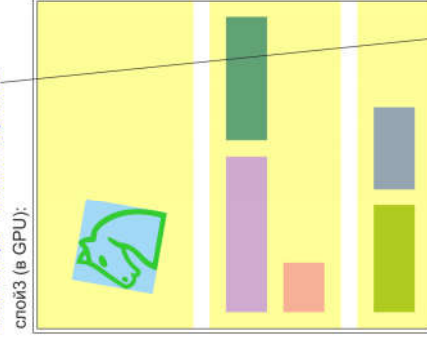


6. команда в GPU: отобразить слой3



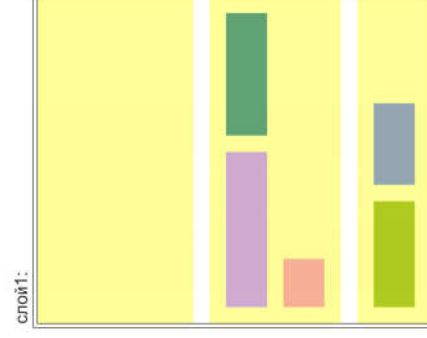
5. команда в GPU: слой3 =

слой1/x0/y0 + слой2/x10/y10/r10

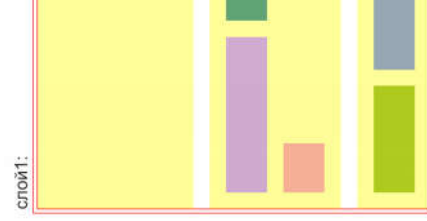


#### 4. отправка в GPU растров слоев

1



↓



поворот делает GPU, а CPU только передаёт команду на поворот

элемент обчислюється як *НПоворотный*, т.к. поворот (*transform: rotate*), масштабирование (*transform: scale*), скос (*transform: skew*), перемещение (*transform: translate*), изменение прозрачности (*opacity*) и наложение фильтров (*filter*) - это операции, которые (обычно) GPU умеет делать самостоятельно и значительно более эффективно чем CPU (но, для того чтобы результат поворота смотрелся качественно, CPU, вероятно, подготовит изображение с запасом размера)

