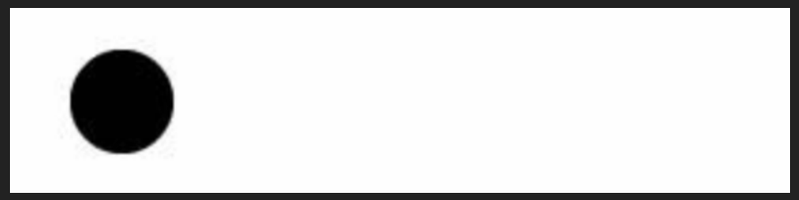
网页动画浅析

武宁

什么是动画?

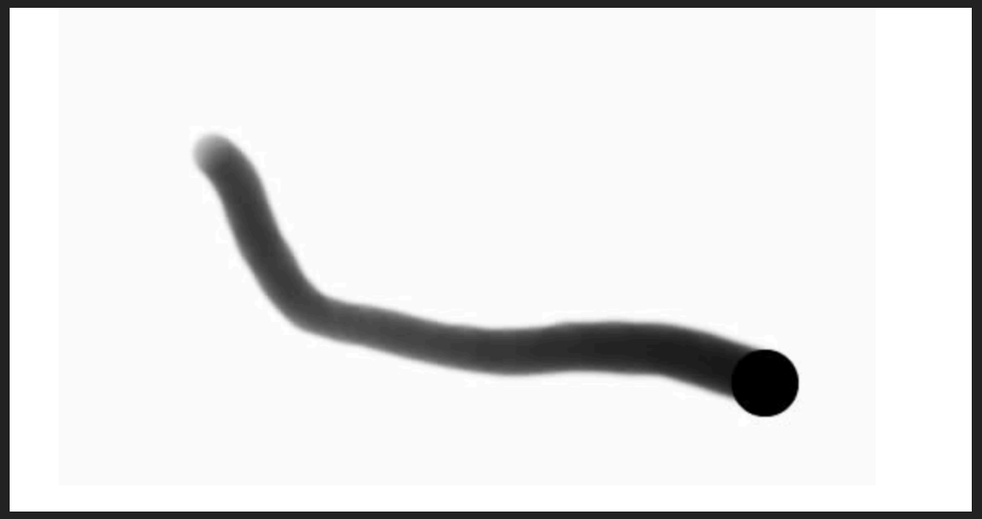
- 动画是创造运动假象的过程
- 电影保持在每秒24帧，人眼就会识别帧为运动的图像
- 网页或游戏中要保持每秒60帧，才能产生动画假象

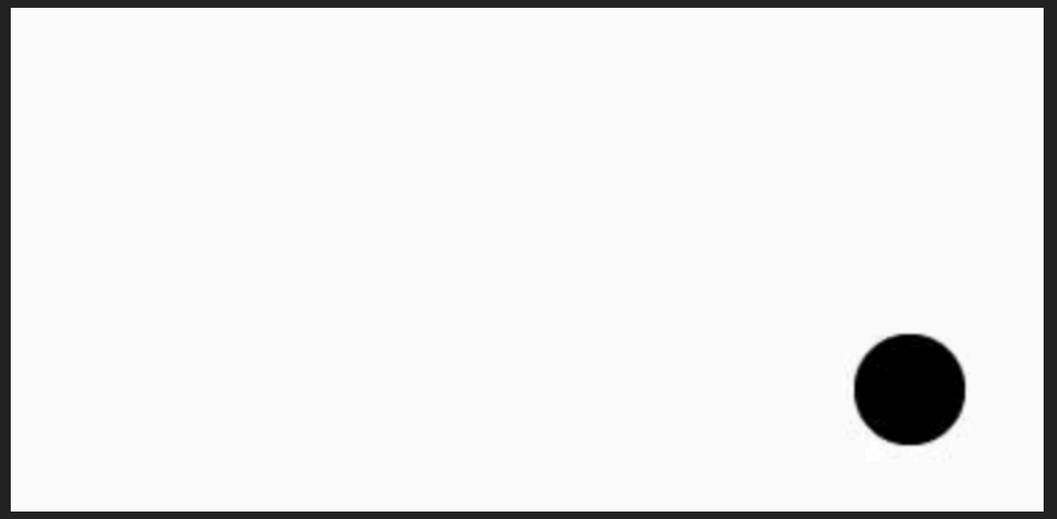
TEXT



电影

游戏



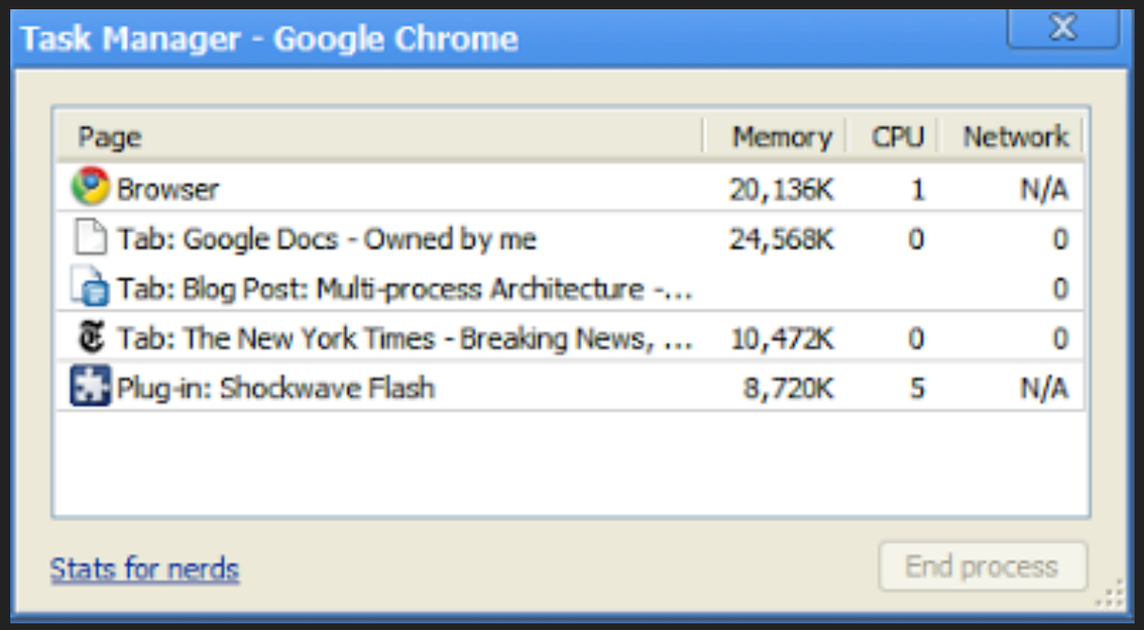


TEXT

网页动画本质

- 动画 = 计算 + 渲染
- 计算：CPU、GPU
- 渲染：layout、paint、composite

浏览器结构

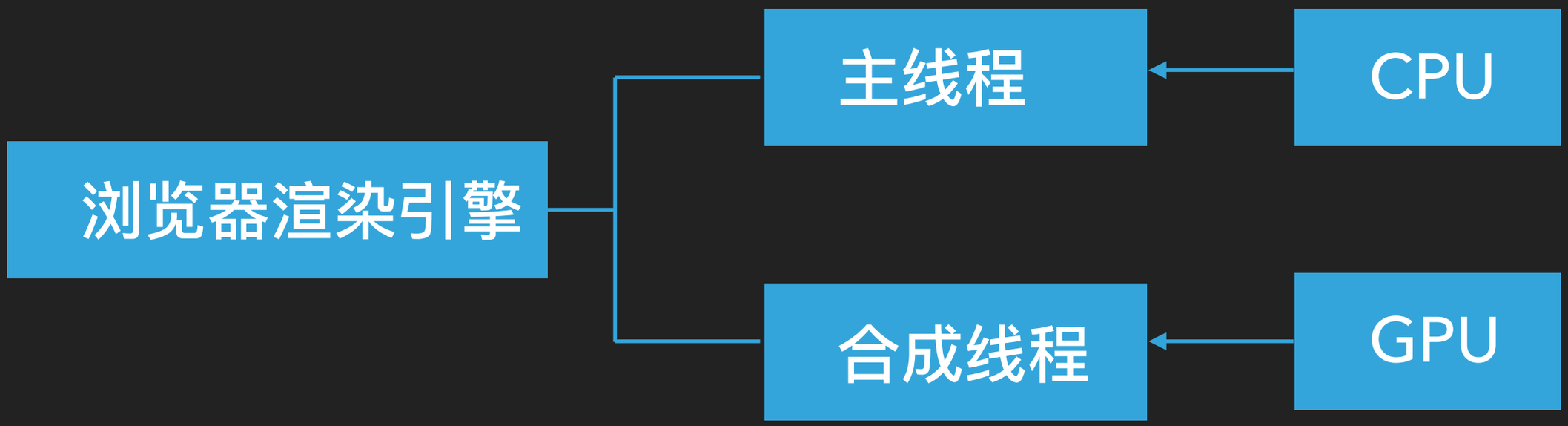


浏览器进程

插件进程

渲染器进程

浏览器渲染进程



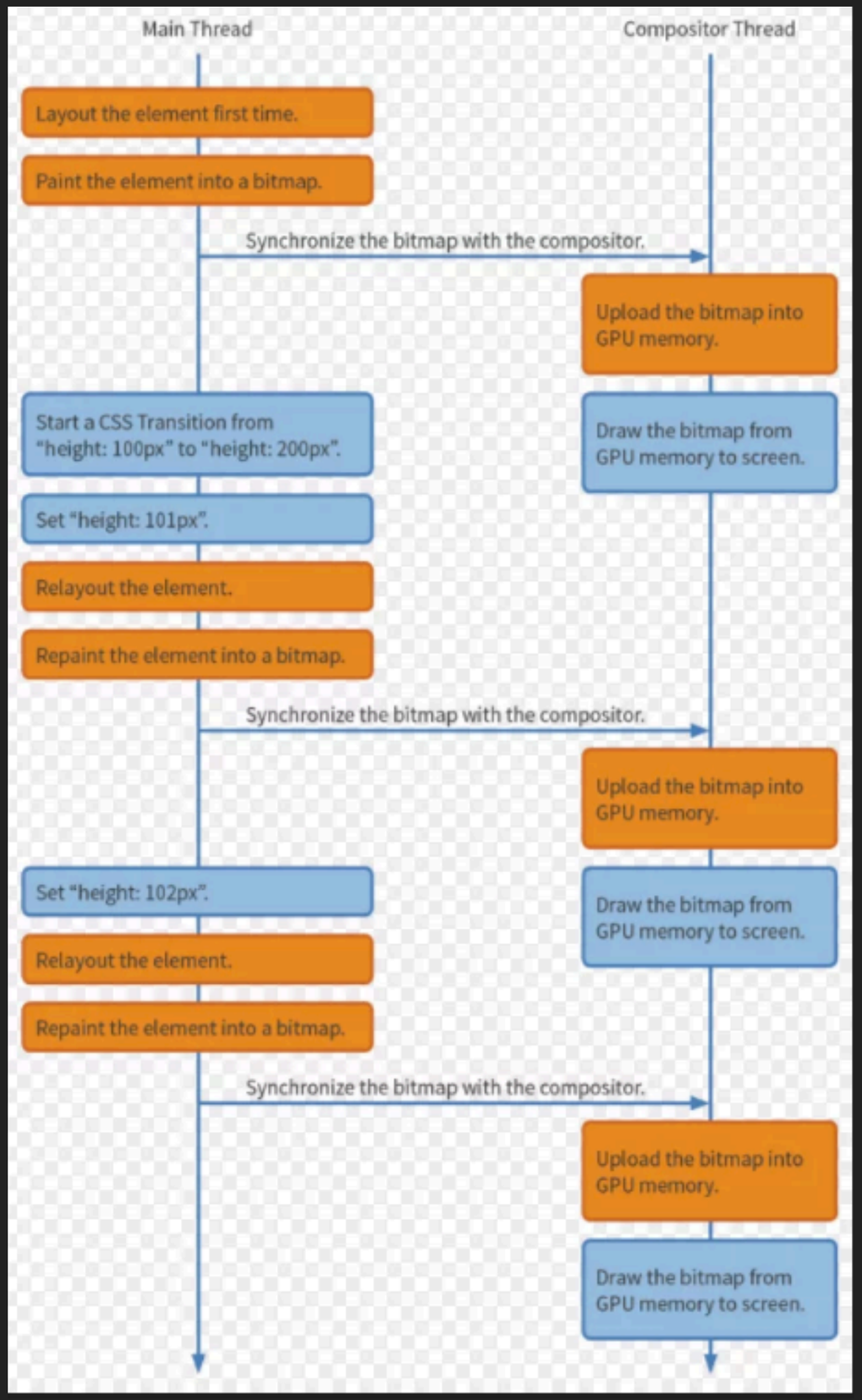
- 浏览器渲染引擎：Webkit、Gecko、Blink、EdgeHTML
- 主进程：运行js、css样式计算、layout、paint
- 合成线程：composite、render

常规动画实现原理

```
    <style>
        div {
            height: 100px;
            transition: height 1s linear;
            background-color: red;
        }

        div:hover {
            height: 200px;
        }
    </style>
</head>
<body>
    <div>我是一个测试的div</div>
</body>
```

demo



TEXT

实现网页动画的几种方式

- css3：性能很高，但是不够灵活，API有限
- canvas：依赖js，不会触发重排操作，旧IE不支持
- svg：擅长处理矢量图形，交互容易，操作dom，旧IE不支持
- js + dom：灵活，但复杂动画需要考虑性能
- gif：GIF兼容性最好，但是画质差，无交互
- flash：技术比较陈旧

Demo:

spriteCanvas.html、spriteCss.html、spriteJs.html

TEXT

CSS-KEYWORDS

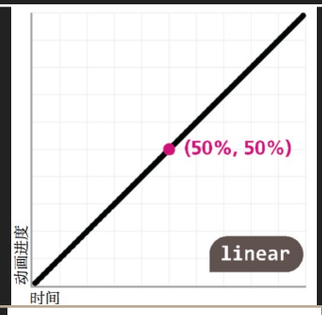
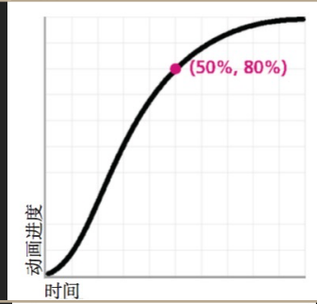
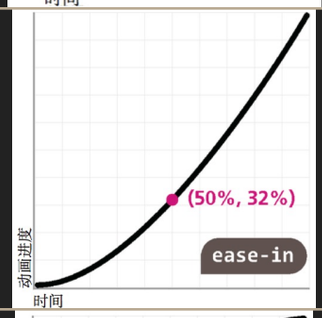
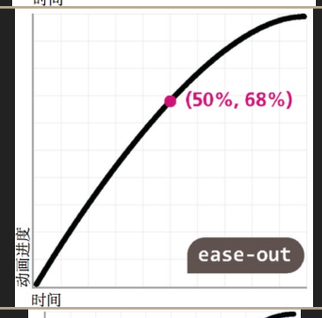
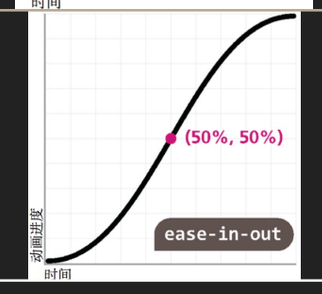
- transform:对元素进行变形
- transition:对元素某个属性或多个属性的变化进行控制，但只有两个关键帧：开始 结束
- animation:对元素某个属性或多个属性的变化，进行控制，可以设置多个关键帧(通过keyframes属性)

Demo

ANIMATION

- CSS3:animation-timing-function:cubic-bezier(x1,y1, x2,y2)

控制CSS3中的动画速度—ANIMATION-TIMING-FUNCTION

linear	cubic-bezier(0,0,1,1)	
ease	cubic-bezier(0.25,0.1,0.25,1)	
ease-in	cubic-bezier(0.42,0,1,1)	
ease-out	cubic-bezier(0,0,0.58,1)	
ease-in-out	cubic-bezier(0.42,0,0.58,1)	

算法原理-杨辉三角

- 1 $(x+y)^1=x+y$
- 1 1 $(x+y)^2=x^2+2xy+y^2$
- 1 2 1 $(x+y)^3 = x^3+3x^2y+3xy^2+y^3$
- 1 3 3 1 …
- 1 4 6 4 1

假设x=t,y=1-t,t∈(0,1)

$$(x+y)^3 = x^3+3x^2y+3xy^2+y^3$$

$$(t+(1-t))^3 = t^3+3t^2(1-t)+3t(1-t)^2+(1-t)^3$$

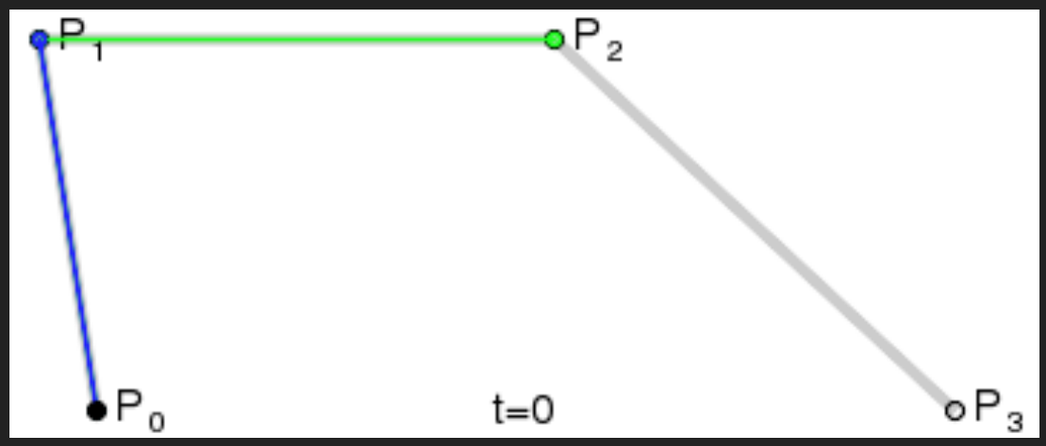
$$B(t)= P_0t^3+3P_1t^2(1-t)+3P_2t(1-t)^2+P_3(1-t)^3$$

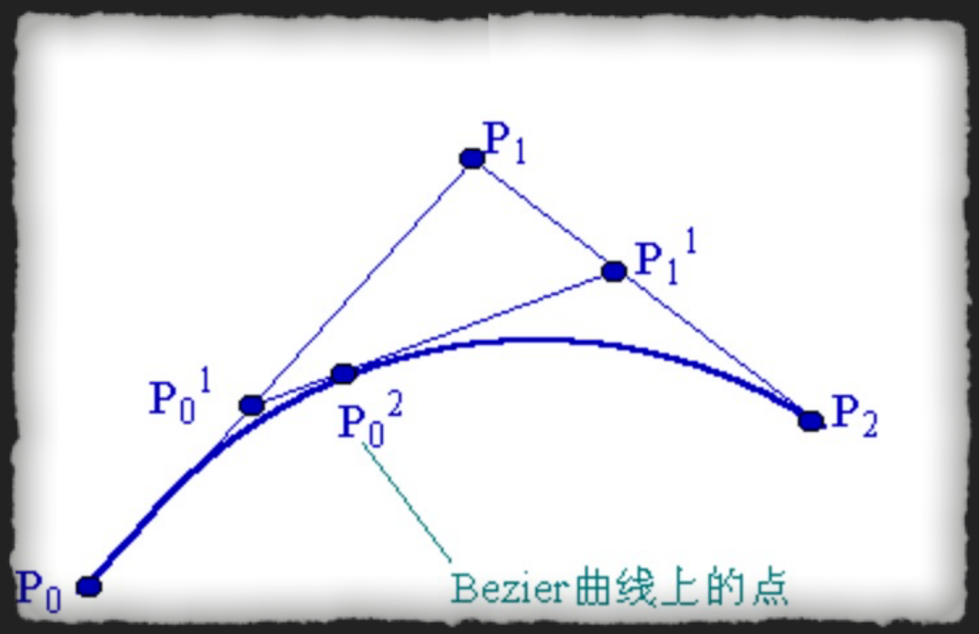
$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} \mathbf{P}_i (1-t)^{n-i} t^i = \binom{n}{0} \mathbf{P}_0 (1-t)^n t^0 + \binom{n}{1} \mathbf{P}_1 (1-t)^{n-1} t^1 + \cdots + \binom{n}{n-1} \mathbf{P}_{n-1} (1-t)^1 t^{n-1} + \binom{n}{n} \mathbf{P}_n (1-t)^0 t^n, \ t \in [0,1]。$$

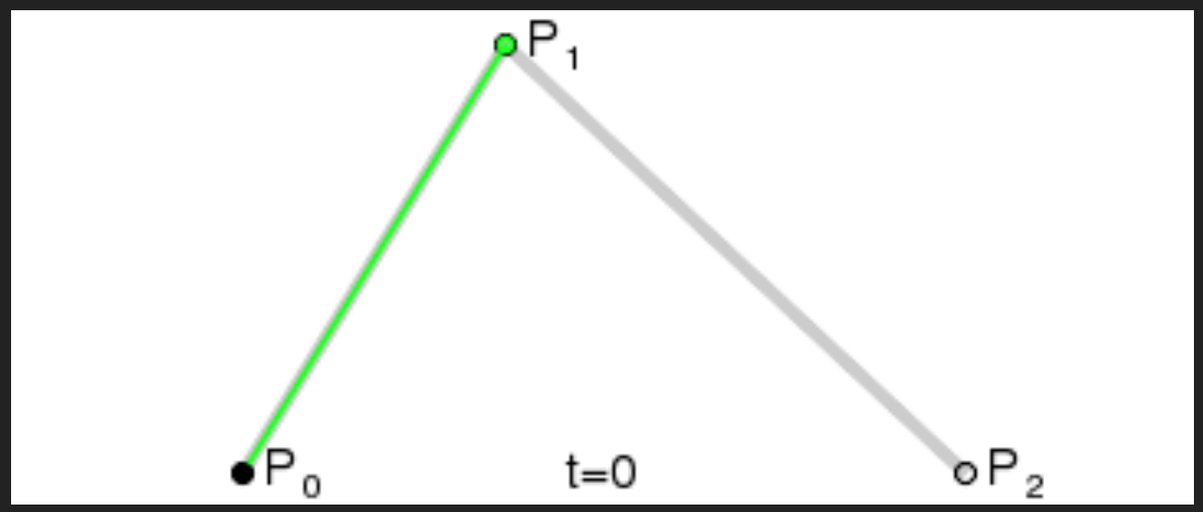
与t相关的速度曲线函数

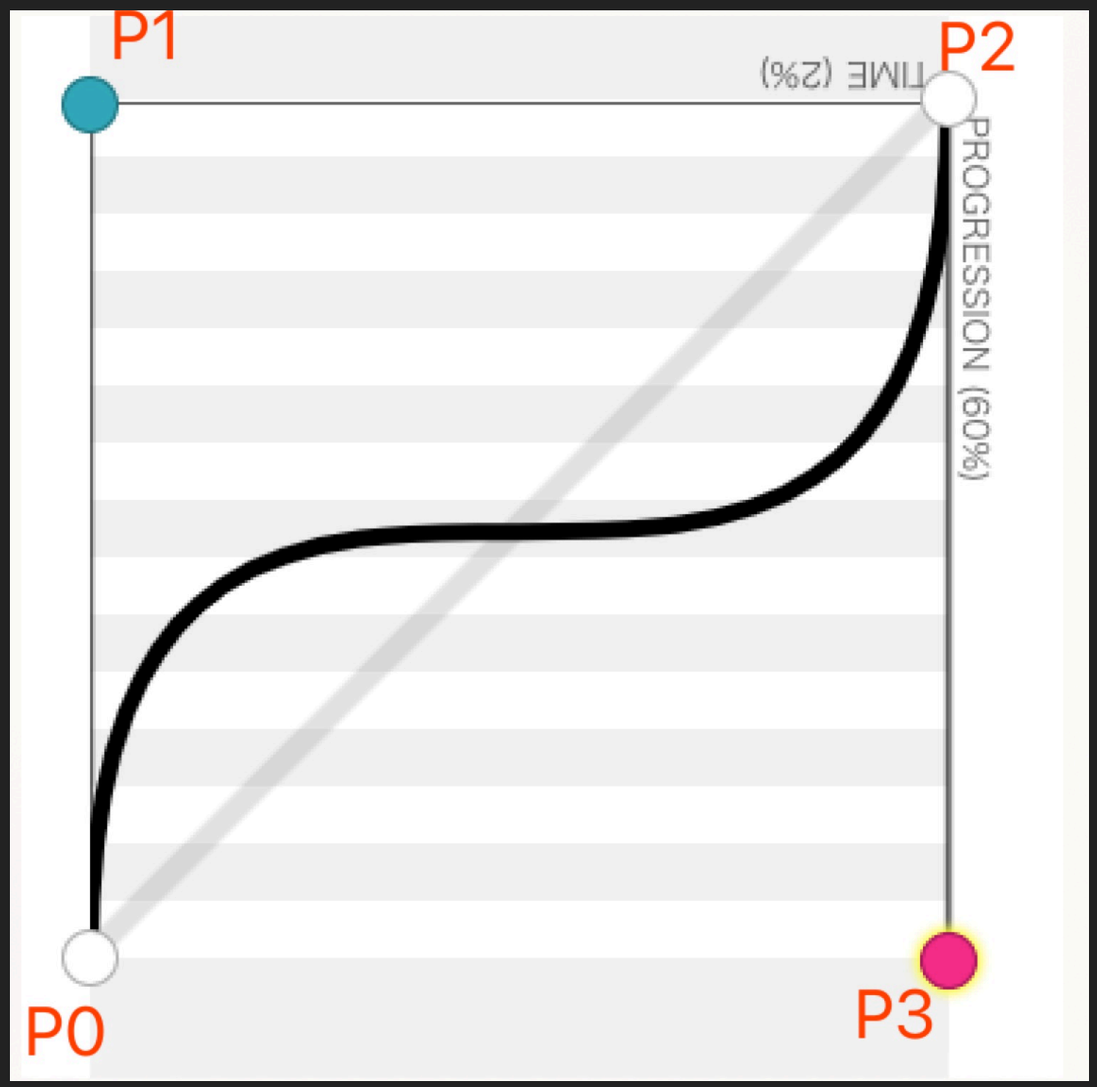
贝塞尔曲线

$$t = \frac{P_0P_0^1}{P_0^1P_1} = \frac{P_1P_1^1}{P_1^1P_2} = \frac{P_0^1P_0^2}{P_0^2P_1^1}$$



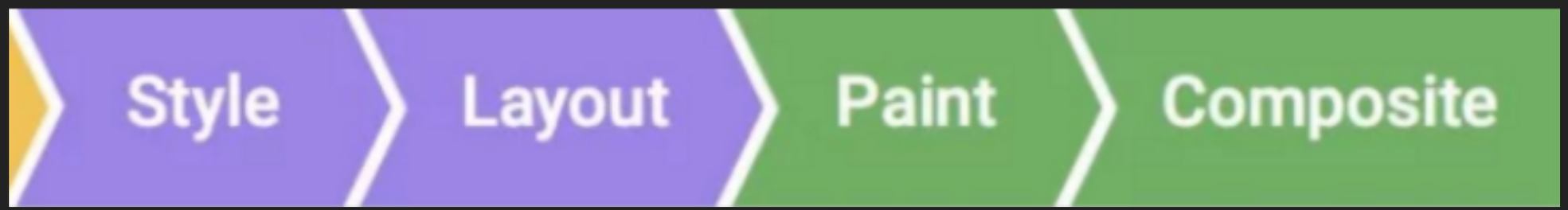




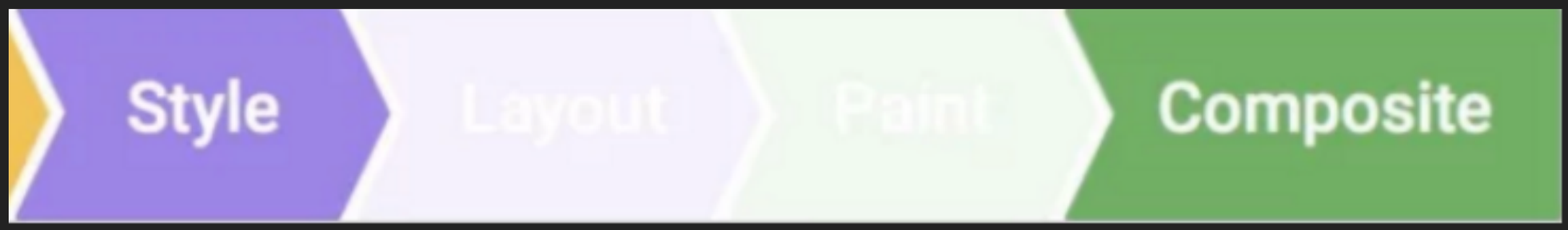


CSS3渲染原理

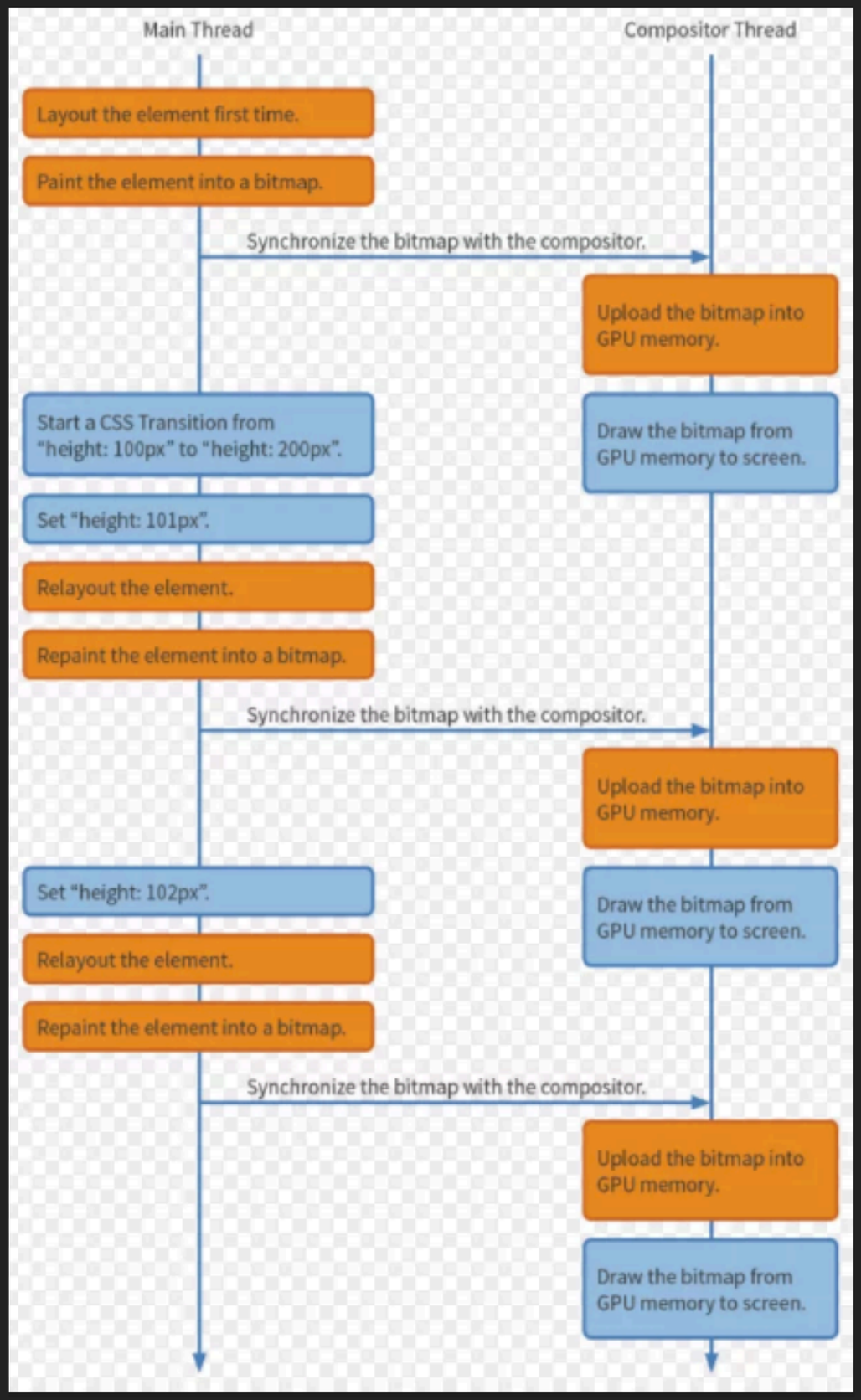
before

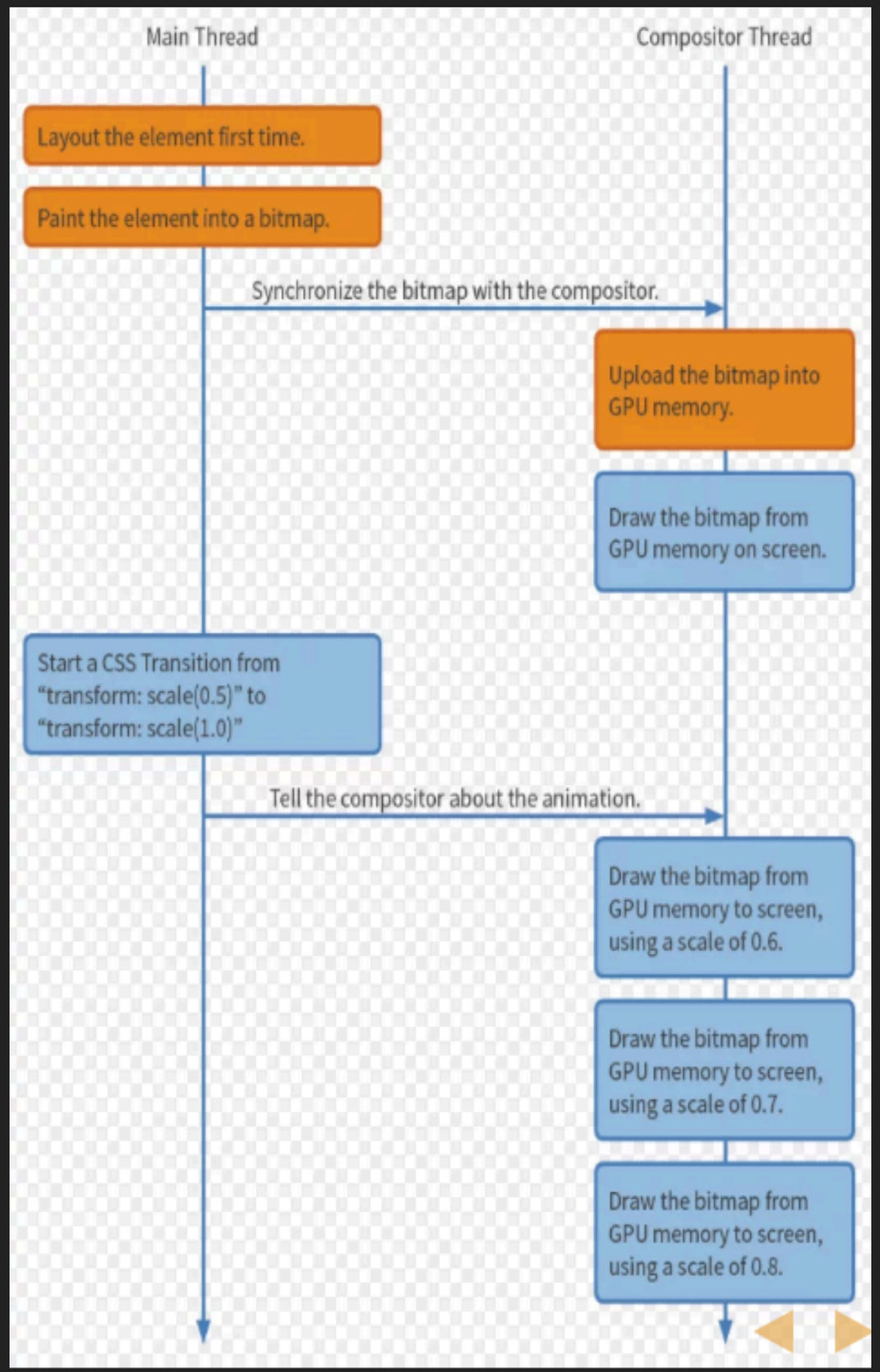


after



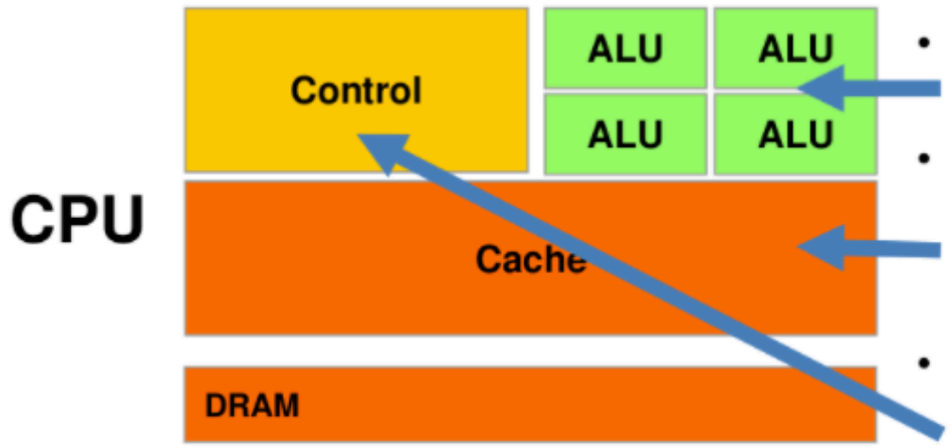
demo


Main Thread
Compositor Thread
Layout the element first time.
Paint the element into a bitmap.
Synchronize the bitmap with the compositor.
Upload the bitmap into GPU memory.
Start a CSS Transition from "height: 100px" to "height: 200px".
Draw the bitmap from GPU memory to screen.
Set "height: 101px".
Relayout the element.
Repaint the element into a bitmap.
Synchronize the bitmap with the compositor.
Upload the bitmap into GPU memory.
Set "height: 102px".
Draw the bitmap from GPU memory to screen.
Relayout the element.
Repaint the element into a bitmap.
Synchronize the bitmap with the compositor.
Upload the bitmap into GPU memory.
Draw the bitmap from GPU memory to screen.


Main Thread
Compositor Thread
Layout the element first time.
Paint the element into a bitmap.
Synchronize the bitmap with the compositor.
Upload the bitmap into GPU memory.
Draw the bitmap from GPU memory on screen.
Start a CSS Transition from "transform: scale(0.5)" to "transform: scale(1.0)"
Tell the compositor about the animation.
Draw the bitmap from GPU memory to screen, using a scale of 0.6.
Draw the bitmap from GPU memory to screen, using a scale of 0.7.
Draw the bitmap from GPU memory to screen, using a scale of 0.8.

通用计算

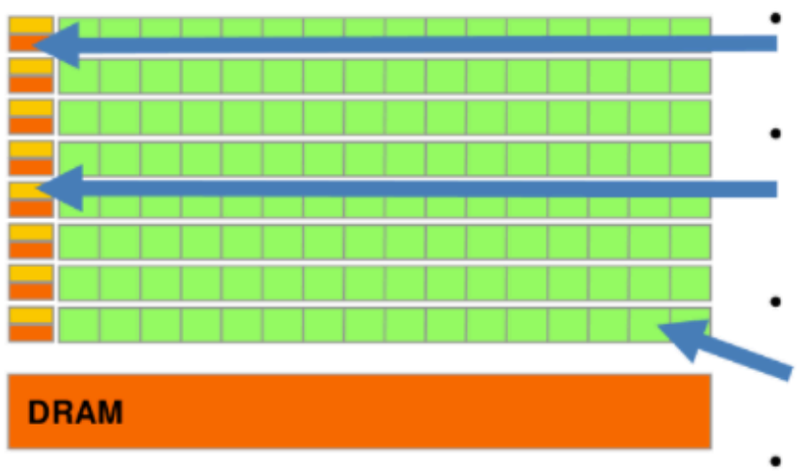
CPUs: Latency Oriented Desigr



- Powerful ALU
  - Reduced operation latency
- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency

专用计算

GPUs: Throughput Oriented Design

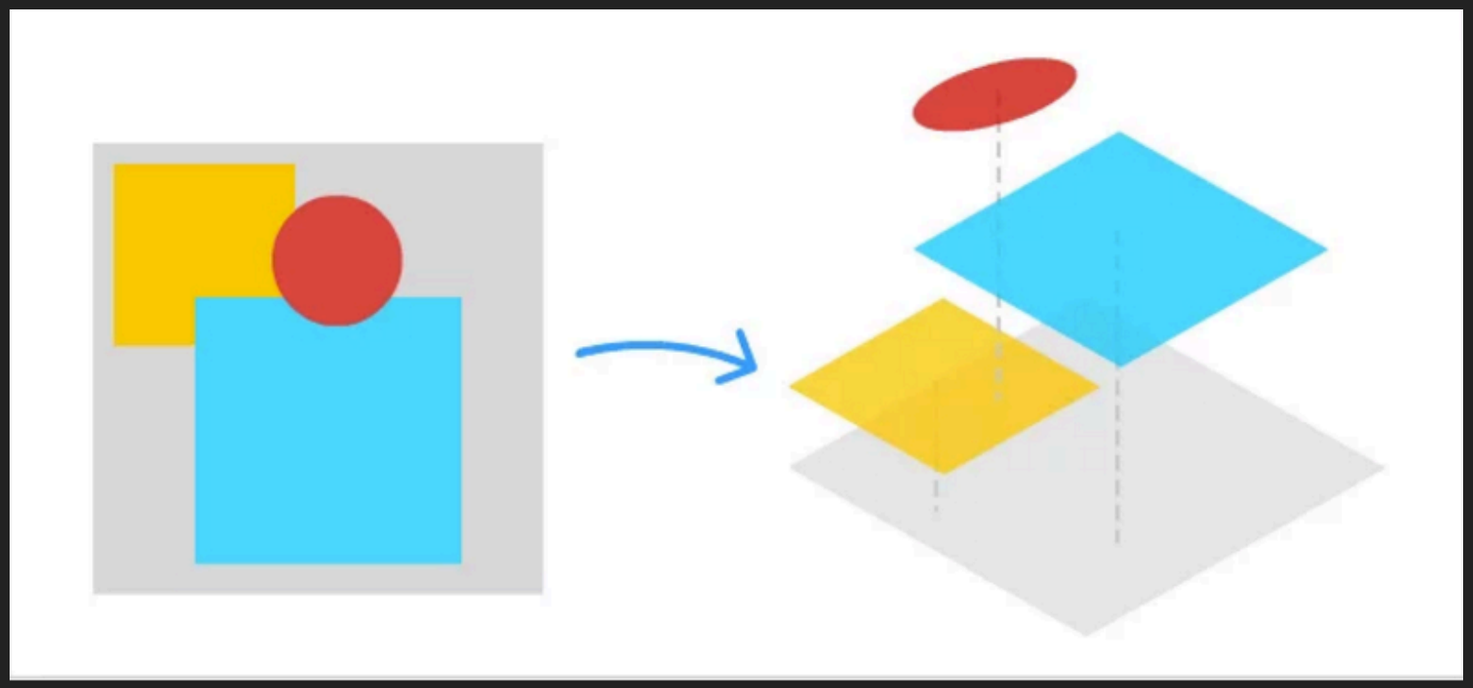


- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies

CANVAS 特点

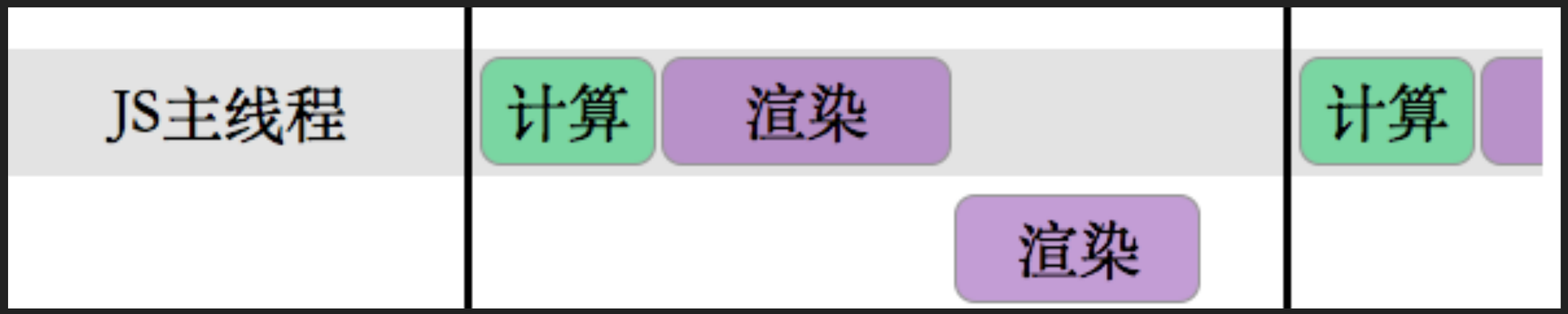
- 处理动画灵活，但是画布性能差
- context是状态机，调取API都会引起重新渲染
- 兼容性问题导致的层层封装（编译适配，字体）
- 采用CPU + GPU共同计算

CANVAS 绘制方式



- 每层精灵运动频率不同
- 方便画布动作清除

离屏CANVAS



- 减少渲染相关API调用次数
- 尽量调用渲染开销低的API

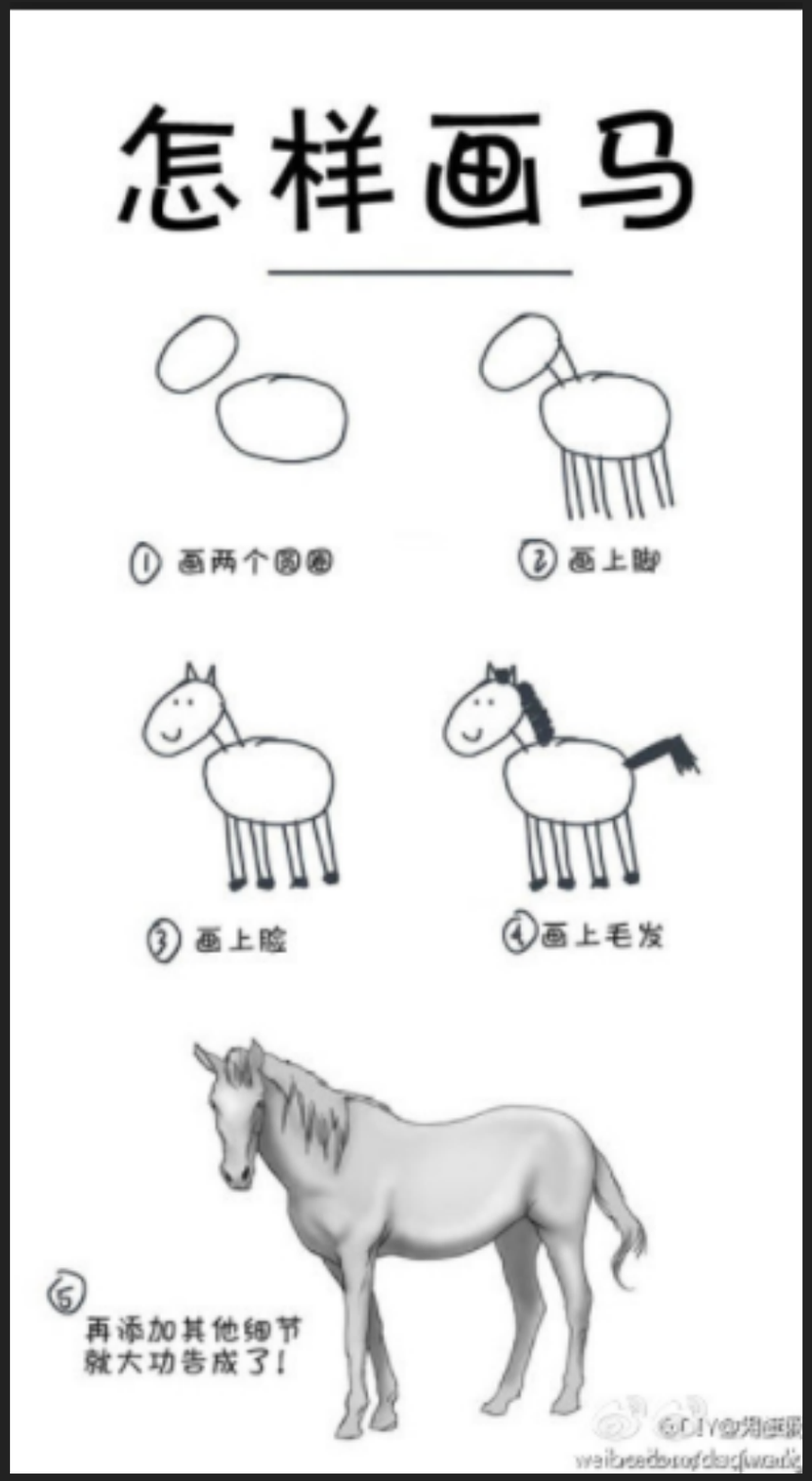
局部绘制

- cancelRequestAnimation
- animation-play-state
- pageVisibility

避免阻塞

- 代码层级的优化---任务拆分
- web worker

总结



分享只是提供一种性能优化的思路，更多的方式依赖于实践操作。

QA & 谢谢