JavaOne™

JavaOne™

JavaFX Graphics Tips & Tricks

Richard Bair
Java Client Architect

MAKE THE FUTURE JAVA

ORACLE®

CAUTION !

WRITE CLEAN CODE, THEN PROFILE!

The content of this session represents the state-of-the-art as of JavaFX 2.2. JavaFX 8 already optimizes some of the issues demonstrated in this session.

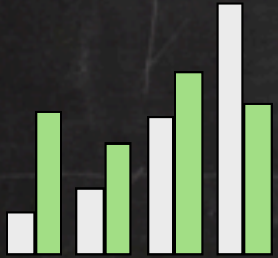
Syllabus

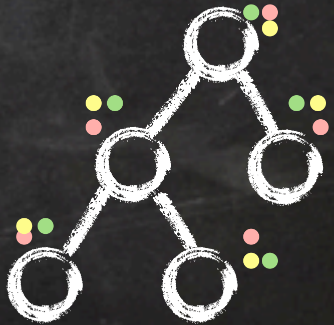
Performance #'s

Rules for good performance

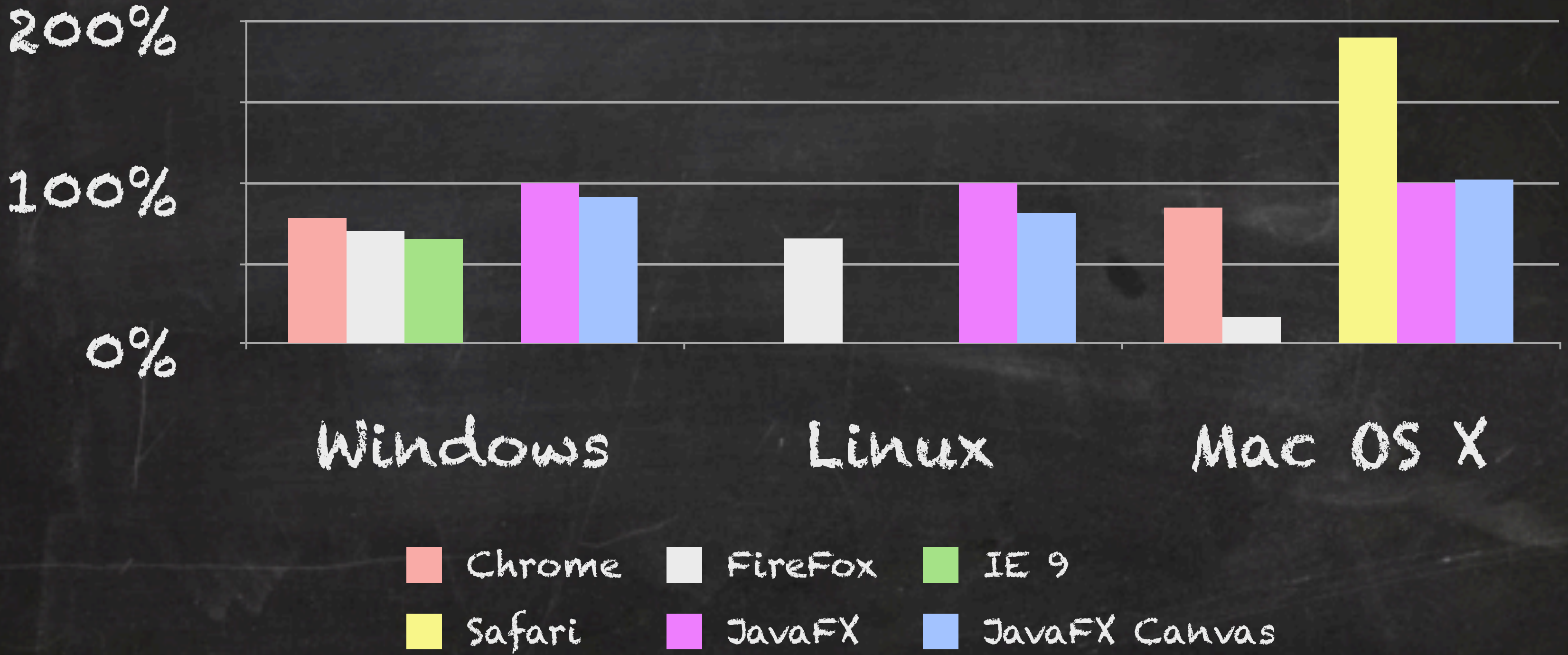
Tips n' Tricks

mem(N) < mem(M)
1K * N
Occlusion Culling
Dirty Area Managment

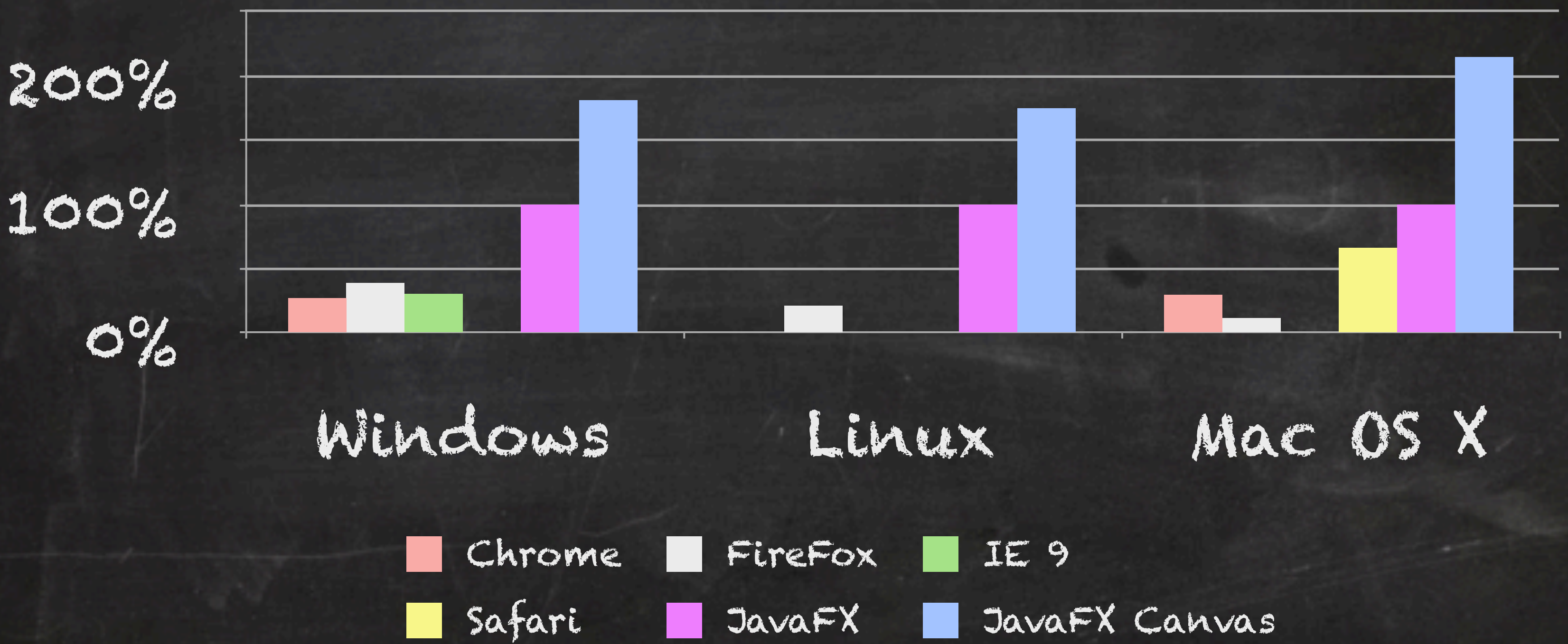




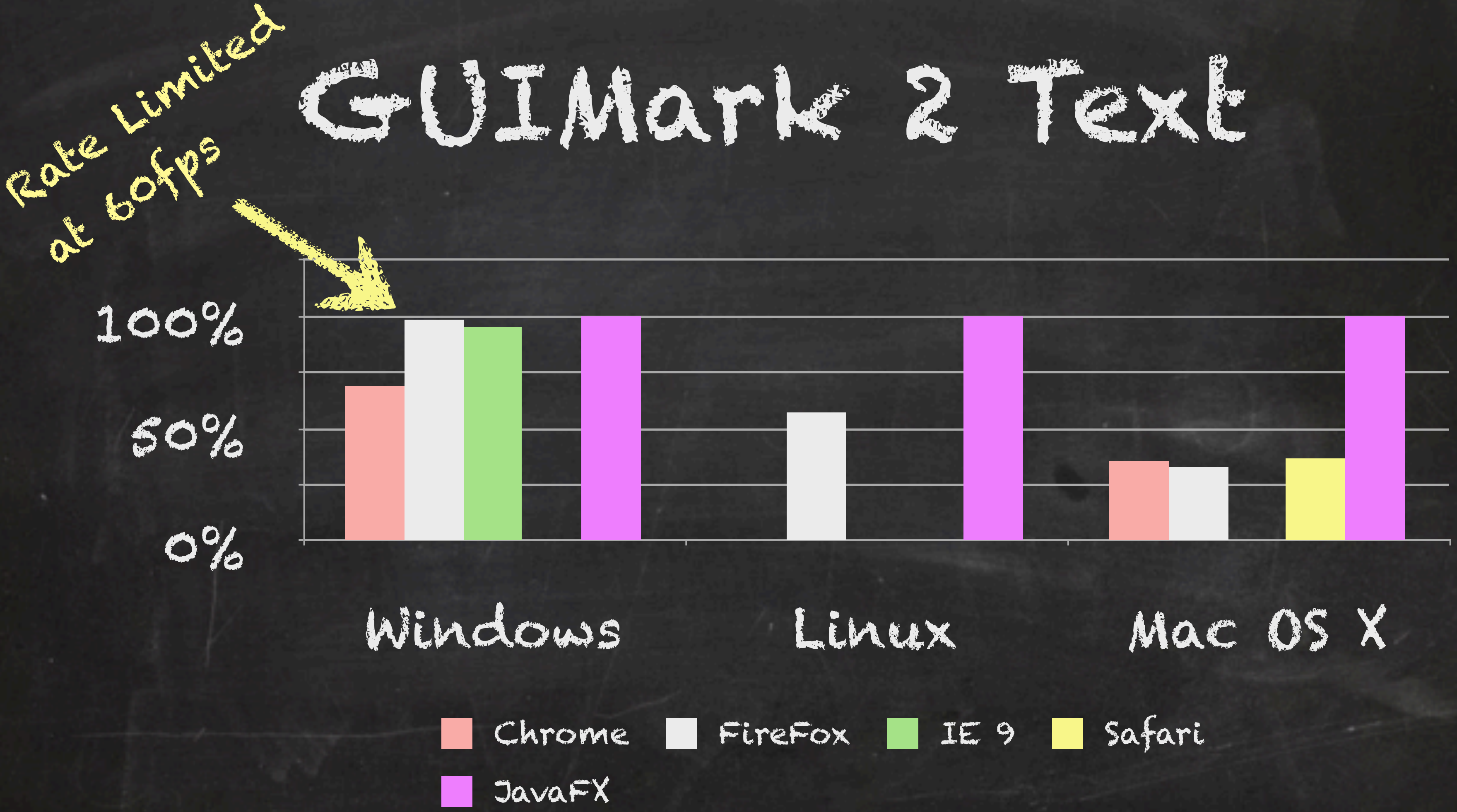
GUIMark 2 Vector



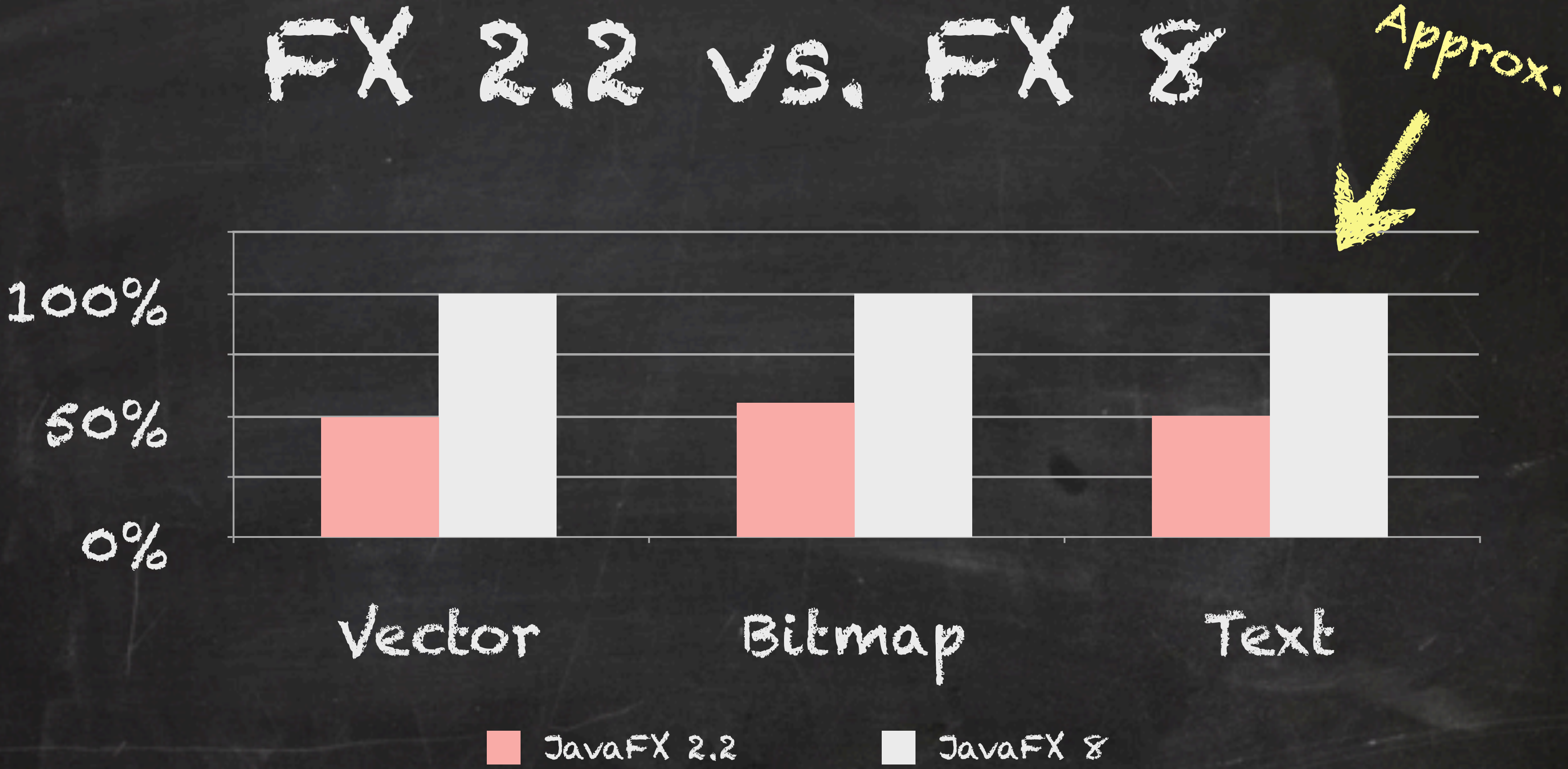
GUIMark 2 Bitmap



GUIMark 2 Text

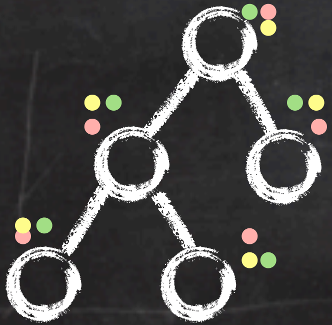


FX 2.2 vs. FX 8



Rule #1: Do Less Work

Use Fewer Nodes



CSS

Layout

Picking

Rendering

Dirty Regions

Smaller Systems require a much more intense round of performance tuning. But surprisingly, time is often spent where you least expected!

Execute Less Code

Every line counts

Extra method calls add up

- On some systems, excessive inlining is expensive
- Excessive method invocations are expensive
- So reduce unnecessary method calls!

Reduce Method Calls

```
@Override
protected double computePrefWidth(double height) {
    return getInsets().getLeft() + 200 +
            getInsets().getRight();
}
```

Reduce Method Calls

```
@Override
protected double computePrefWidth(double height) {
    final Insets insets = getInsets();
    return getInsets().getLeft() + 200 +
            getInsets().getRight();
}
```

Reduce Method Calls

```
@Override
protected double computePrefWidth(double height) {
    final Insets insets = getInsets();
    return insets.getLeft() + 200 + insets.getRight();
}
```

What limits you?

Fill rate (nearly 100% certainty)

Geometry rate (not likely)

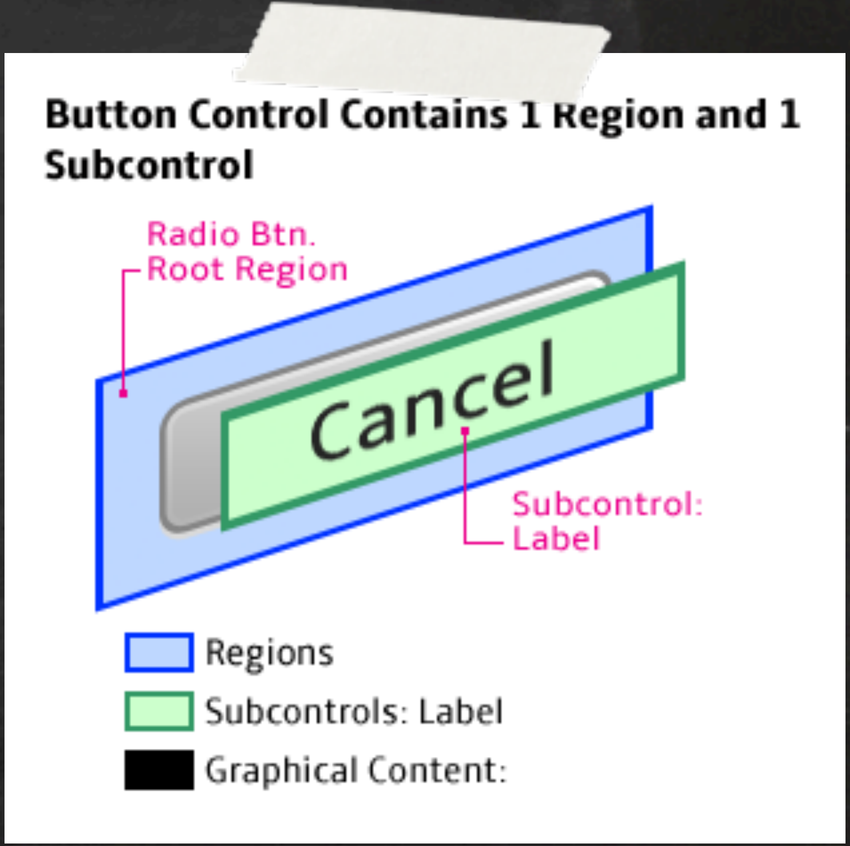
CSS Overhead (possible)

Layout computation time (maybe)

System I/O (good chance)

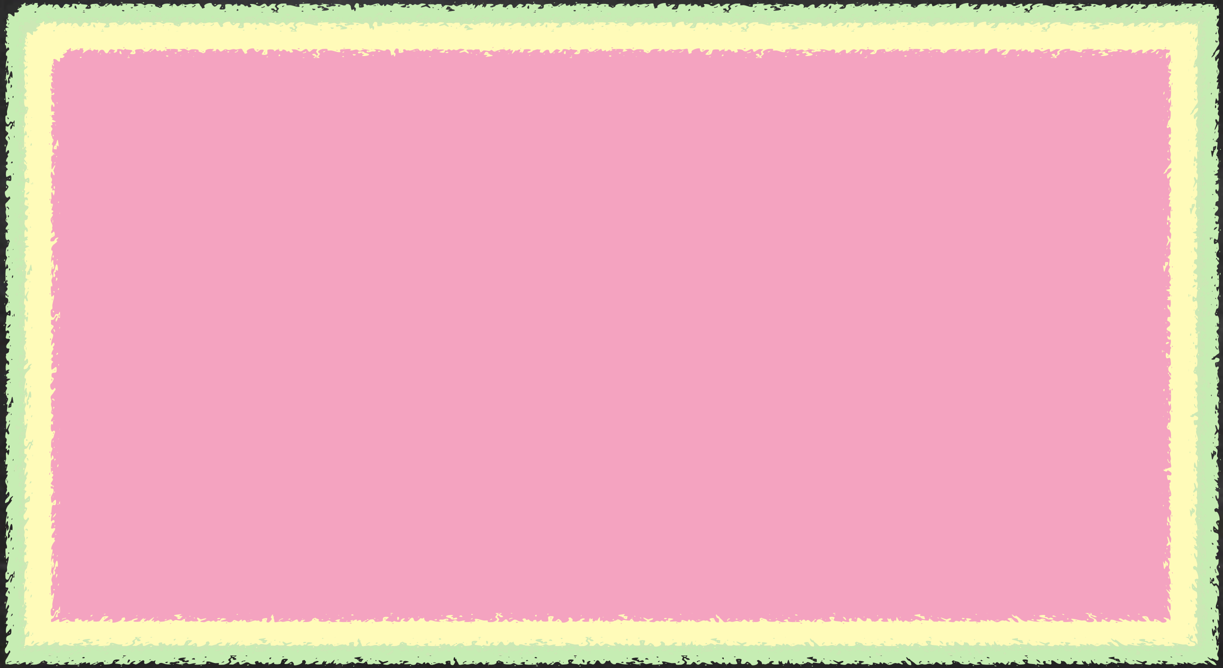
Fill Rate

Buttons Are Drawn with multiple layers



Fill Rate

> 90% visible pixels are drawn multiple times!



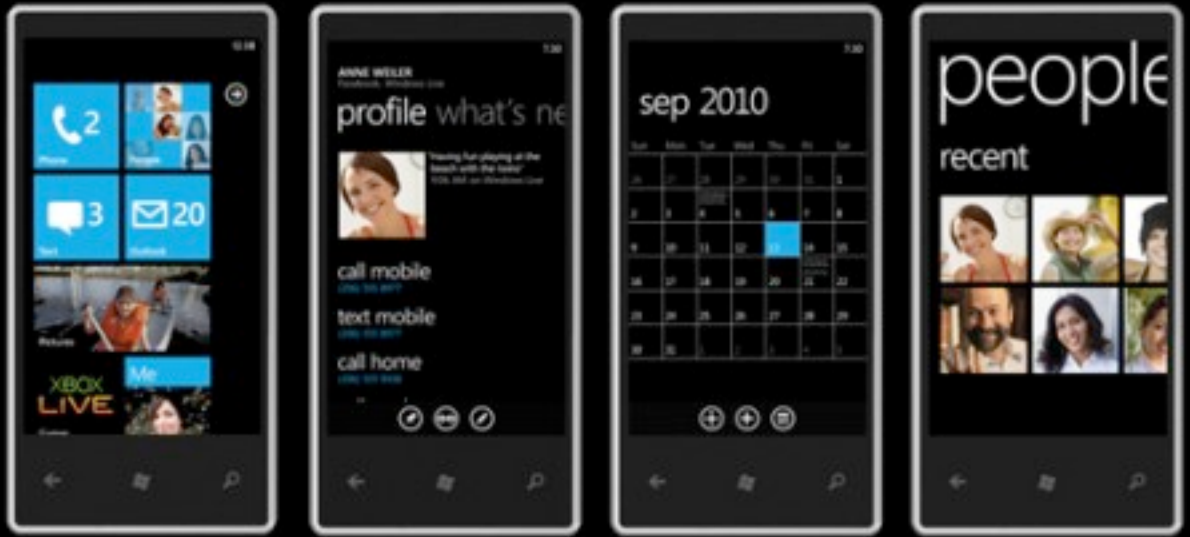
Improving Fill Rate

- Only draw what has changed
- Dirty Regions!
  - Scene Graph does this automatically!
- Limit use of (some) effects
- Limit use of non-rectangular non-axis aligned clips
- Reduce Overdraw

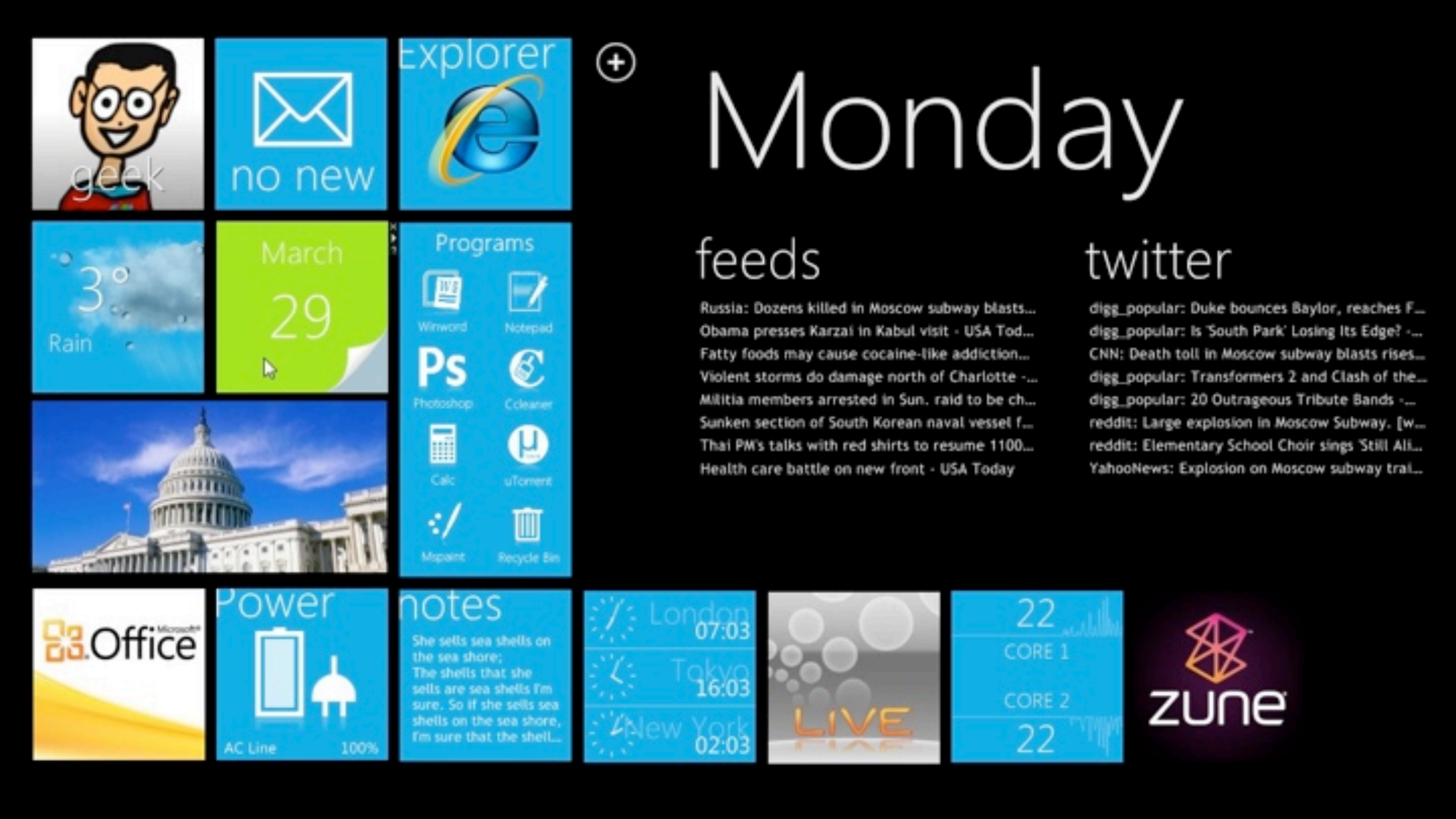
Reducing Overdraw

- Use Image Skinning
- Automatic Region Texture Cache (FX 8)
- Background Fills consolidated
- Simplify the Style (Metro, Android)
- Reduce # of overlapping Nodes
- ~~Reduce # of Nodes~~
  - Will have NO EFFECT!

Metro



Windows Phone 7 Series | MIX10



She sells sea shells on the sea shore;
The shells that she sells are sea shells I'm sure. So if she sells sea shells on the sea shore, I'm sure that the shell...

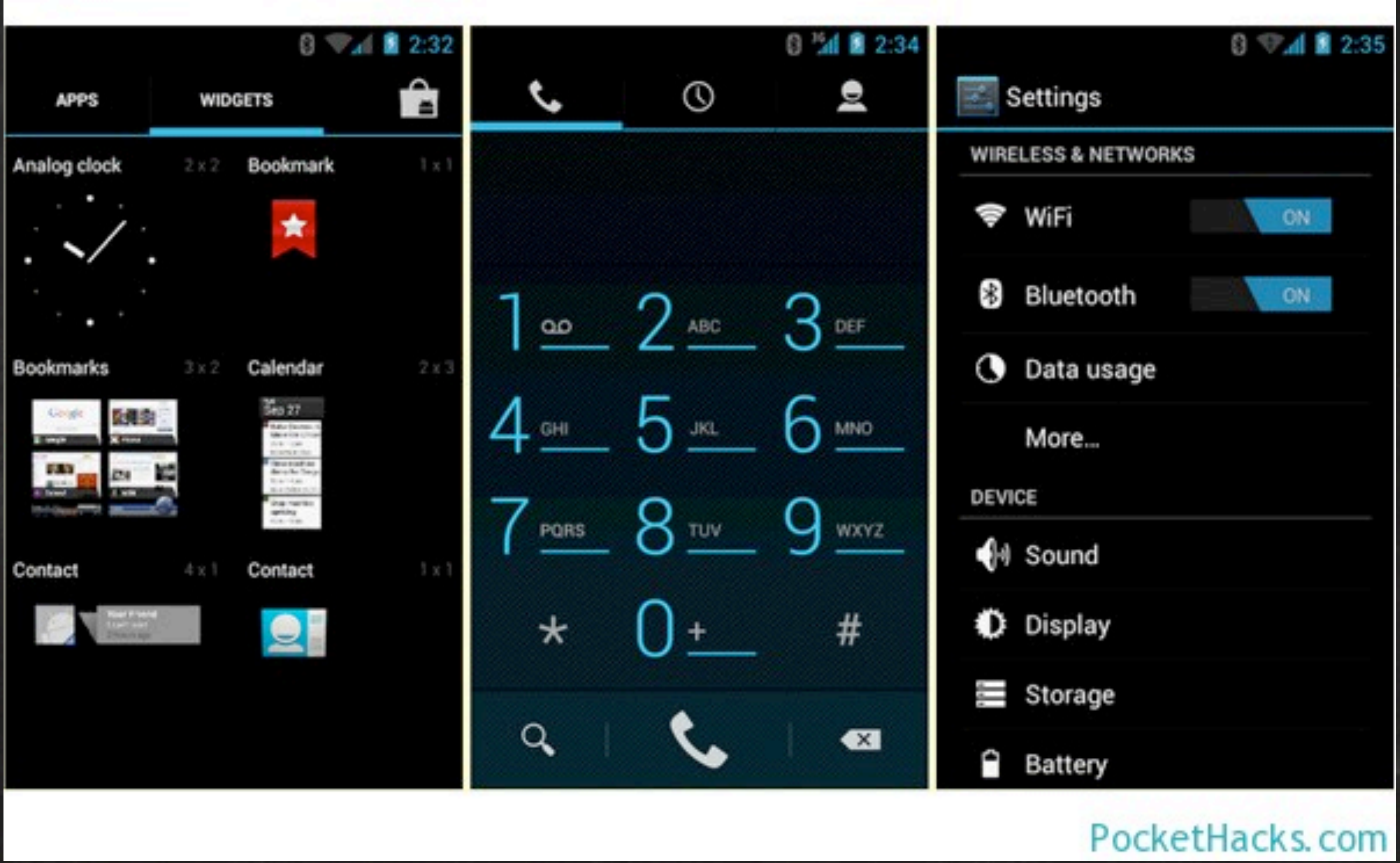
Monday

feeds

Russia: Dozens killed in Moscow subway blasts...
Obama presses Karzai in Kabul visit - USA Tod...
Fatty foods may cause cocaine-like addiction...
Violent storms do damage north of Charlotte -...
Militia members arrested in Sun. raid to be ch...
Sunken section of South Korean naval vessel f...
Thai PM's talks with red shirts to resume 1100...
Health care battle on new front - USA Today

twitter

digg_popular: Duke bounces Baylor, reaches F...
digg_popular: Is 'South Park' Losing Its Edge? -...
CNN: Death toll in Moscow subway blasts rises...
digg_popular: Transformers 2 and Clash of the...
digg_popular: 20 Outrageous Tribute Bands -...
reddit: Large explosion in Moscow Subway. [w...
reddit: Elementary School Choir sings 'Still Ali...
YahooNews: Explosion on Moscow subway trai...

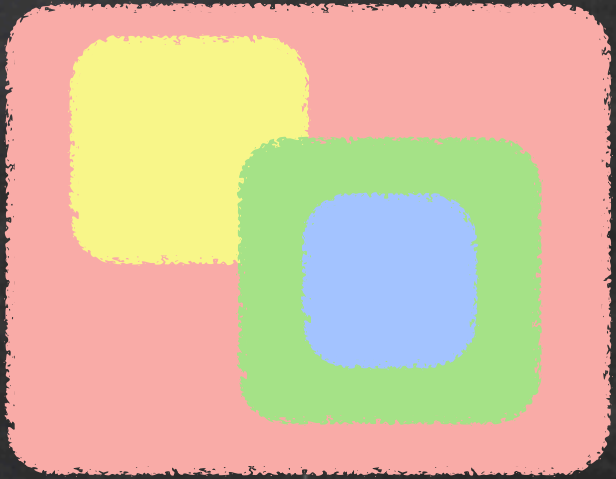


WiFi ON

Bluetooth ON

Occlusion Culling

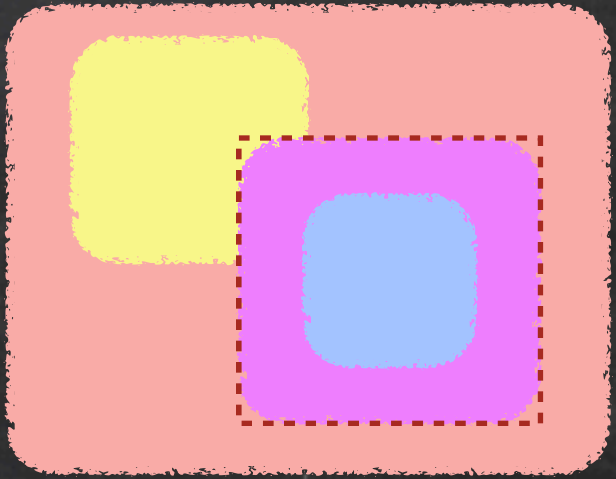
Suppose green becomes purple



Occlusion Culling

Suppose green becomes purple

We have to draw the following dirty area

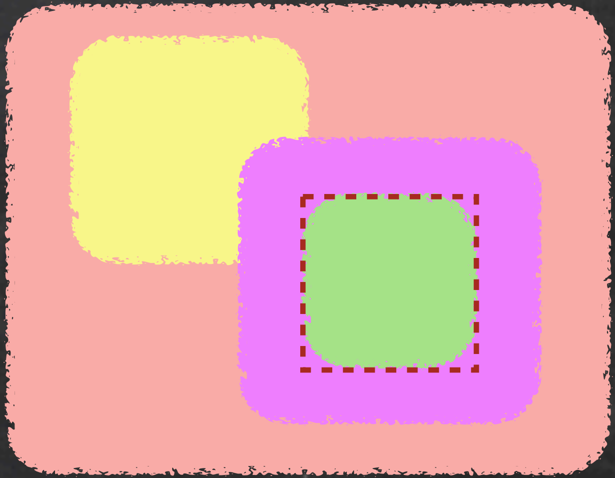


This requires drawing all of red, yellow, purple, and blue inside the dirty area

Occlusion Culling

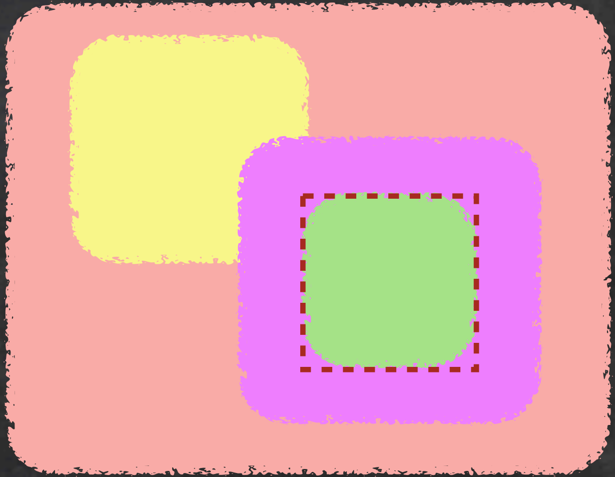
Suppose blue becomes green

Now the dirty region is thus



We only have to draw the purple and green because there are no visible red or yellow parts

Occlusion Culling



By not drawing (culling) things that won't be visible, we reduce overdraw and increase rendering performance

CSS Costs

- Parsing a stylesheet
- Whenever the id / style class changes, the node and potentially all child nodes must be updated
- Pseudo-class state changes are typically very fast -- unless you have children who's state depends on a parent's selector!

CSS Horror Show

```
.parent:hover .child { ... }
```

Horrible! If the parent's hover changes we visit each child and recompute the style!

CSS Horror Show

```
.parent .child { ... }
```

Yikes! When we encounter a node with the .child style class, we must walk up the entire scene graph looking for a .parent!

CSS Horror Show

```
node.setStyle("-fx-background-color:blue;")
```

The "style" property is very convenient, but don't over-do it. We have to fire up the CSS parser whenever we encounter a style, and the internal processing is heavier.

CSS Honor Show

```
.parent > .child { ... }
```

Alright! When matching .child, we only have to check the immediate ancestor to see if it has .parent

CSS Honor Show

```
.child:hover { ... }
```

Handling pseudo-classes for child matches is dead easy and super fast!

Tip: Avoid Structure Changes

Changing the scene graph requires re-applying CSS.

Requires "structural integrity checks"

Use toFront / toBack (we've optimized this)

Tip: Use FXCollections

Shoot for minimal notification overhead

- setAll vs. clear & addAll
- avoid multiple add calls

FXCollections.sort()

- sends "permutation" change events
- "permutations" are handled by separate fast paths

Tip: Virtualization

ListView is blistering fast

- Reuses Nodes, keeps memory usage, CSS changes, layout changes, invalidations, and everything else to the minimum!

Reuse ListView for all your virtualization needs!

Tip: Manual Layout

Extend Region

- (almost) Always implement computePrefWidth, computePrefHeight
- implement layoutChildren()

Custom layout can cut corners over the built in layout containers

Layout

JavaFX Asks:

- "How wide / tall would you like to be?"
- "How wide / tall is the biggest you would allow?"
- "What is your smallest sensible width / height?"
- "Is your width dependent on your height, or vice versa?"
- "What is your baseline?"
- "What should I consider your 'natural' position, width, and height?"
- "Can you be resized?"

Layout

These questions are all asked for each node during layout.

JavaFX asks a lot of questions.

Major Tip!

Content Bias

contentBias = HORIZONTAL | VERTICAL

HORIZONTAL = height depends on width

VERTICAL = width depends on height

null = width and height are independent

Content Bias

(contentBias = null) is by far the fastest

- All computed pref / min width / height are cached

Content Bias

(contentBias = HORIZONTAL) is common for text with a wrapping width (where height depends on width)

Content Bias

contentBias != null isn't actually well supported in the built-in layouts. Its a bug :-(

Doh!

Rule #2: Know Your Device

NVidia GForce GTX 690

# Cores = 3072

Fill Rate = 234 Billion / Sec

Mem Bandwidth = 384 Gbps

Max Power = 300W

Min Sys Power = 650W

NVidia GForce 310

# Cores = 16

Mem Bandwidth = 8 Gbps

Max Power = 30.5W

Min Sys Power = 300W

PowerVR SGX 543MP3

# Cores = 3

JavaFX gives you a single development platform and a single set of APIs, but which APIs you can and can't use is going to depend on the inherent performance characteristics of the device.

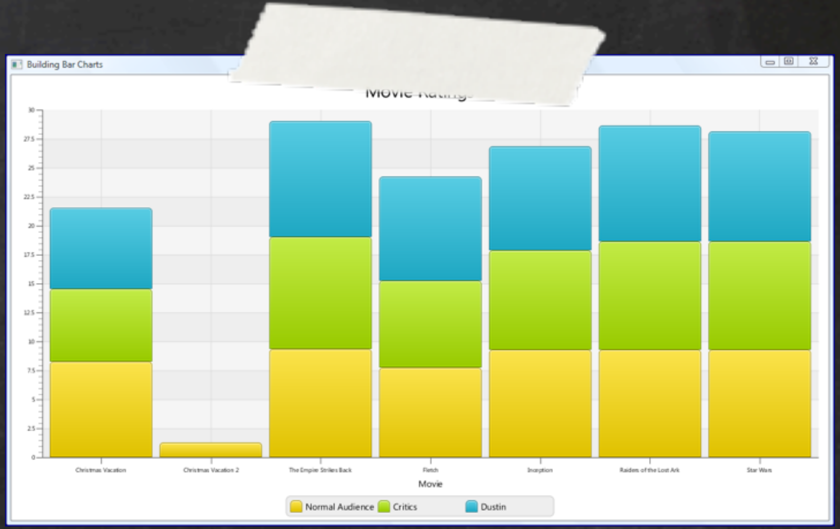
Rule of Thumb:

20K-100K Nodes on Desktop
500-1000 Nodes on Embedded
100-200 Nodes on Small Embedded (320x200)

It really just depends on your hardware

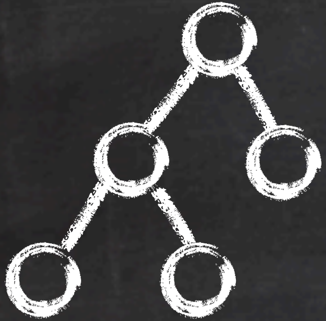
Tip: Cache

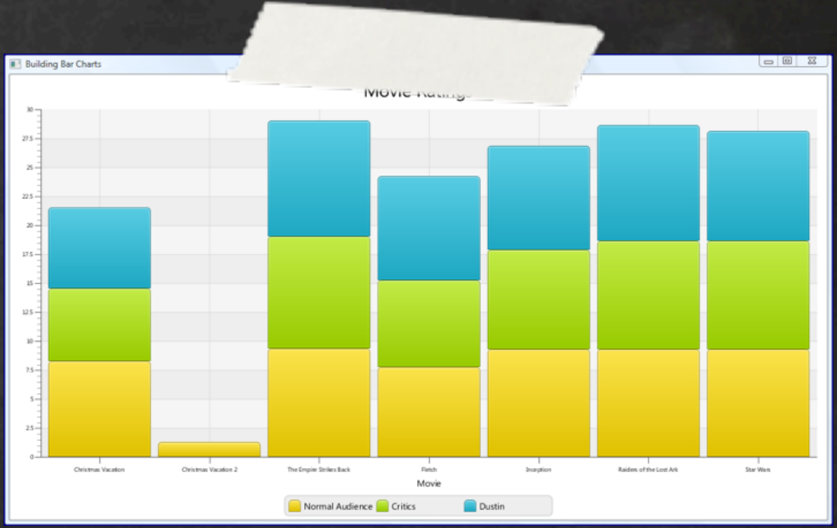
Lots 'o work to draw



If nothing's changing, by George, cache it!

Draw once to image





Draw a bazillion times to screen

Backfires if the node is changing a lot!

Tip: Cache Hint

Set CacheHint to SPEED when rotating, scaling for better performance!

Tip: -Djavafx.pulseLogger=true

```
PULSE: 1 [250ms:989ms]
T12 (8ms): CSS Pass
T12 (2ms): Layout Pass
T12 (151ms): Waiting for previous rendering
T12 (2ms): Copy state to render graph
T10 (24ms): Dirty Opts Computed
T10 : 2 different dirty regions to render
T10 (54ms): Painted
T10 (4ms): Painted
Counters:
        Nodes rendered: 70
        Nodes visited during render: 143
        Parent#layout() on clean Node: 2
        Parent#layout() on dirty Node: 122
```

Tip: -Djavafx.pulseLogger=true

Pulse Count & Duration & Time since last pulse

Various events (two threads)

Various counters

```
PULSE: 1 [250ms:989ms]
T12 (8ms): CSS Pass
T12 (2ms): Layout Pass
T12 (151ms): Waiting for previous rendering
T12 (2ms): Copy state to render graph
T10 (24ms): Dirty Opts Computed
T10 : 2 different dirty regions to render
T10 (54ms): Painted
T10 (4ms): Painted
Counters:
        Nodes rendered: 70
        Nodes visited during render: 143
        Parent#layout() on clean Node: 2
        Parent#layout() on dirty Node: 122
```

CAUTION !

WRITE CLEAN CODE, THEN PROFILE!

The preceding were general guidelines and principles to guide in performance tuning. Don't overdo it or you will have an unmaintainable mess.

JavaOne™