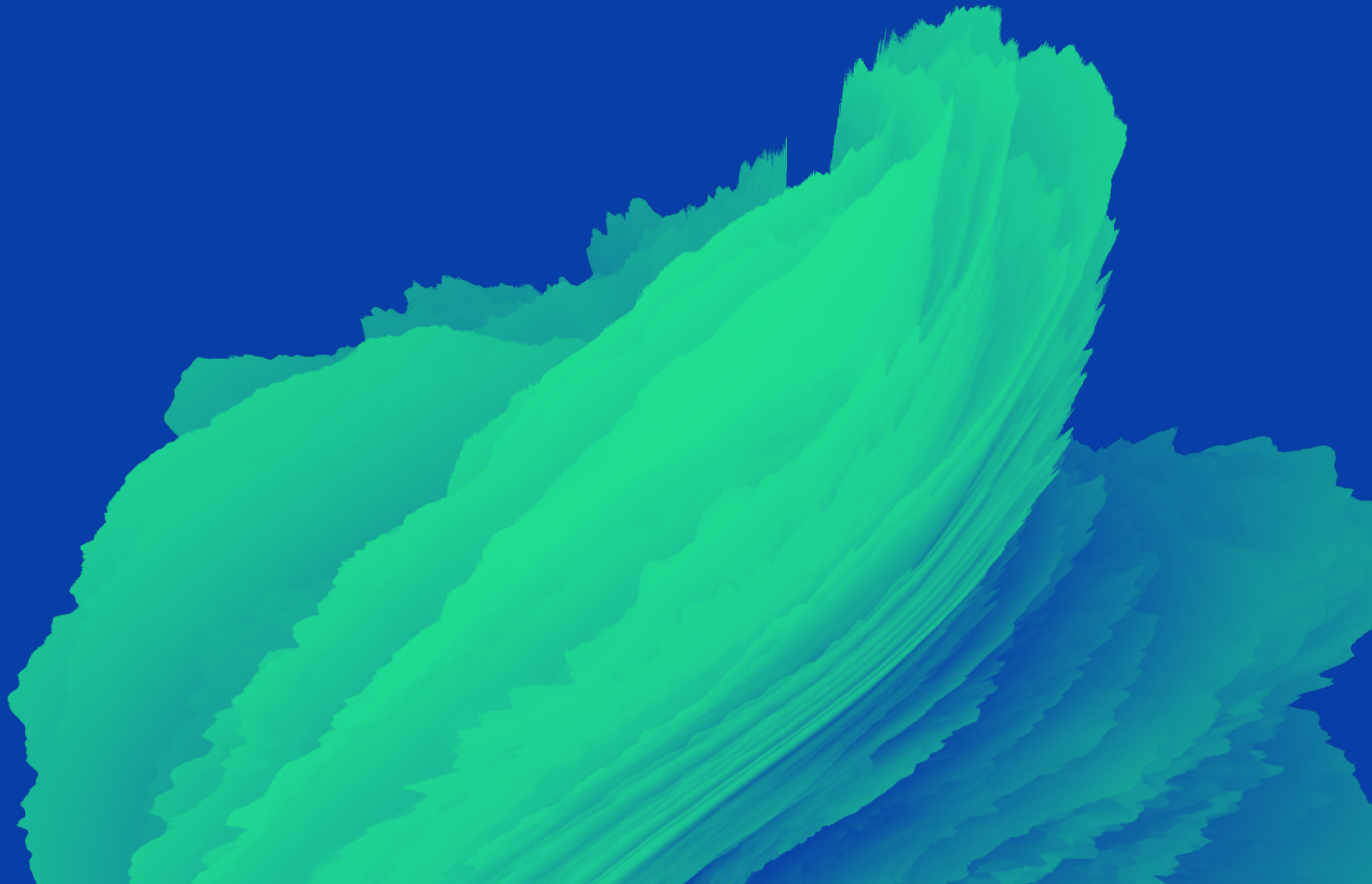
unity

Unite Europe 2017



Foreward, for Slideshare/Youtube

This deck contains slides presented during Unite Amsterdam 2017 and are intended as a supplement to that talk.

More slides are included in this deck than were presented live. The additional slides are included inline with the existing material, and primarily include code samples and additional data to supplement understanding the discussed topics.

In a handful of cases, additional material is added — this is mostly notable in the “Unity APIs You Cannot Trust” section, where several whole sections were cut for time. These have been left in as an added bonus for you, the dedicated reader.

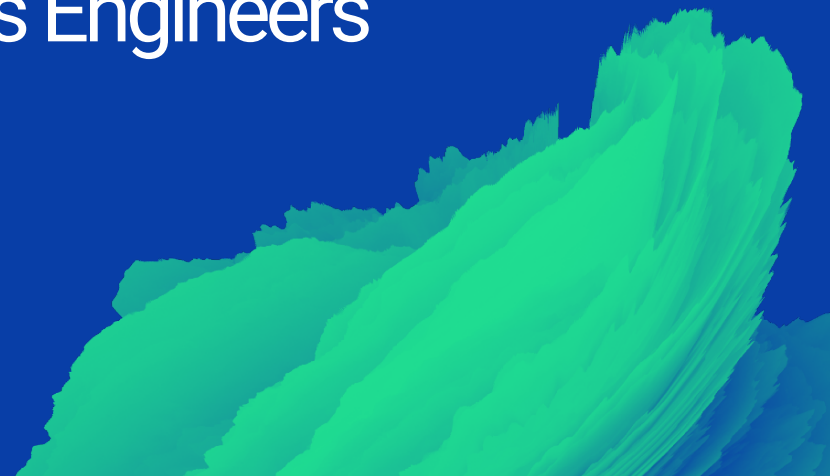
Enjoy!

Squeezing Unity

(Yet another performance talk)

Mark Harkness & Ian Dundore

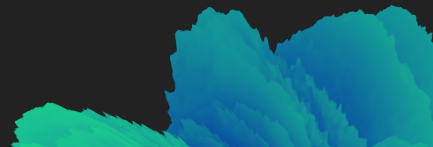
Developer Relations Engineers



What’re we talking about?

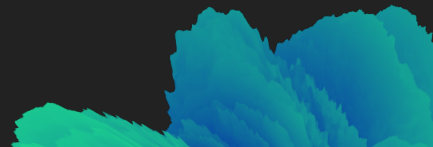
Today's Menu

- Scurrilous APIs
- Data structures
- Inlining
- ~~Extreme micro-optimization~~



Today's Menu

- Scurrilous APIs
- Data structures
- Inlining
- ~~Extreme micro-optimization~~
- Unity UI



Obligatory Reminder:
PROFILE. THINGS. FIRST.

Unity APIs You Cannot Trust

Particle Systems

- APIs recursively iterate over all child GameObjects.
  - Start, Stop, Pause, Clear, Simulate, etc.

- *IsAlive(),* with the default argument, is recursive!

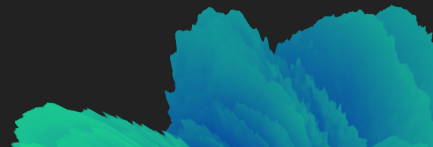
Particle Systems (2)

- Recursive calls execute on *every* child Transform.
- This happens even if none of the children have ParticleSystem Components!

- Assuming a GameObject with 100 children:
  - ParticleSystem.Stop() = 100 calls to GetComponent

Particle Systems

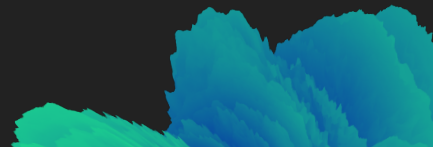
- Solution:
  - **Make sure to pass *false* to *withChildren* param**
  - Cache a list of ParticleSystems in the hierarchy.
  - Iterate over this list manually when calling APIs



Particle Systems: More Sad News

- All calls to *Stop* and *Simulate* will cause GC allocs
  - Versions: 5.4, 5.5, 5.6, 2017.1
- The C# wrapper code was written to use a closure.

- Fixed in 2017.2.



Particle Systems: Workaround

- ParticleSystem APIs are backed by internal helper methods.
  - *Start* --> Internal_Start(ParticleSystem)
  - Stop —> Internal_Stop(ParticleSystem)
  - Simulate —> Internal_Simulate(…)
- Can be addressed via Reflection…

Particle Systems: Function Signatures

```
private static bool Internal_Stop(ParticleSystem self,
            ParticleSystemStopBehavior stopBehavior)

private static bool Internal_Simulate(ParticleSystem self,
            float t, bool restart, bool fixedTimeStep)
```

Arguments have the same meanings as public API.

Classic: Array-returning APIs

Beware APIs that return arrays

- Tenet: When a Unity API returns an array, it is always a new copy of the array.
  - This is for safety reasons.
  - *Mesh.vertices*, *Input.touches*
- Maxim: Minimize calls to APIs that return arrays.

Bad: Allocates many times

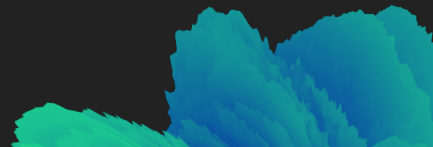
```
for(int i = 0; i < Input.touches.Length; ++i)
{
   doSomethingWithTouch(Input.touches[i]);
}
```

Better: Allocates once

```
var touchArray = Input.touches;
for(int i = 0; i < touchArray.Length; ++i) {
   doSomethingWithTouch(touchArray[i]);
}
```

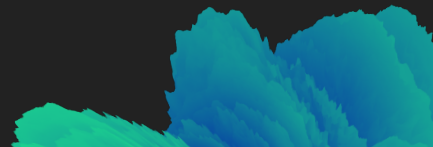
Best: Doesn’t allocate

```
for(int i = 0; i < Input.touchCount; ++i) {
   doSomethingWithTouch(Input.GetTouch(i));
}
```

A non-exhaustive list

- Input.touches
  - Use Input.GetTouch!
- Physics.RaycastAll & other “All” *casts
  - Non-allocating Physics APIs have existed since 5.3!
- GetComponents, GetComponentsInChildren
  - Versions which accept pre-allocated List<T> objects exist!



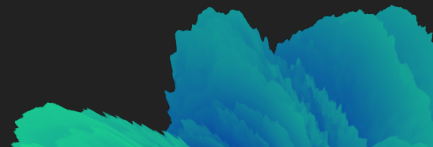
Property Name Hashing

Materials?

- When addressing properties on materials & shaders, use integer property IDs instead of string property names.
  - GetFloat, SetFloat, GetTexture, SetTexture, etc.
- If you pass string properly names, they’re hashed.

Don’t do this

```
Material myMaterial = GetMyMaterialSomehow();
myMaterial.SetFloat(“_SomeProperty”, 100f);
myMaterial.SetTexture(“_SomeTexture”, someTex);
```

```
// During initialization…
int propId_SomeFloat;
Int propId_SomeTexture;
void Awake() {
   propId_SomeFloat = Shader.PropertyToID(“_SomeFloat”);
   propId_SomeTexture = Shader.PropertyToID(“_SomeTexture”);
}

// … Later
Material myMaterial = GetMyMaterialSomehow();
myMaterial.SetFloat(propId_SomeFloat, 100f);
myMaterial.SetTexture(propId_SomeTexture, someTex);
```

Materials: The Catch

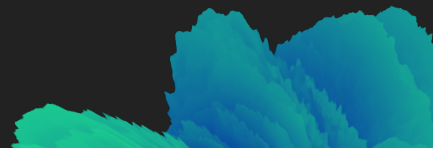
- Material.color
- Material.mainTexture
- Material.mainTextureOffset
- Material.mainTextureScale

- These properties use the string-argument methods.

Materials: Workaround

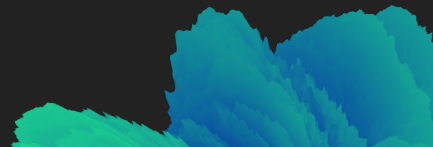
- Use the standard methods instead
- Get & cache the property ID yourself

- Material.color
  - Material.GetColor, Material.SetColor
  - Property name: “_Color”



Materials: Workaround

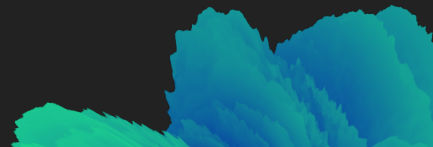
- Property name: “_MainTex”
- Material.mainTexture
  - Material.GetTexture, Material.SetTexture
- Material.mainTextureOffset
  - Material.GetTextureOffset, Material.SetTextureOffset
- Material.mainTextureScale
  - Material.GetTextureScale, Material.SetTextureScale



My favorite one

- *Camera.main*
  - Calls Object.FindObjectWithTag(“MainCamera”)
  - Every single time you access it.

- Remember this one! It’ll be on the homework.



Data Structures

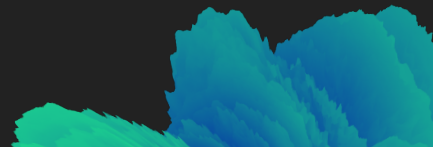
Know how your data structures work!

- Mostly iterating? Use arrays or Lists.
- Mostly adding/removing? Dictionary or Hashset.
- Mostly indexing by key? Dictionary.

- Common problem: iterating over dictionaries & hash sets.
  - These are hash tables: high iteration overhead.

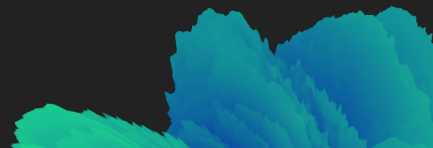
What happens when concerns collide?

- Common case: an Update Manager.
  - Need low-overhead iteration.
  - Need constant-time insertion.
  - Need constant-time duplicate checks.



Examine the requirements.

- Need low-overhead iteration.
  - Array or List.
- Need constant-time insertion.
  - List, Dictionary, HashSet
- Need constant-time duplicate checks.
  - Dictionary, HashSet



No single data structure works?

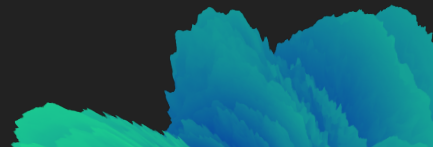
- … so use two.
- Maintain a List for iteration.
- Before changing the List, perform checks on a HashSet
  - If removal is a concern, consider a linked list or intrusively-linked list implementation.
- Downside: Higher memory cost.

Dictionary Quick Tip!

UnityEngine.Object-keyed Dictionaries

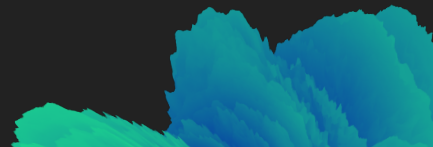
- Default comparer uses *Object.Equals*
- Calls *UnityEngine.Object.CompareBaseObjects*
- Calls C#’s Object.ReferenceEquals method.

- Small amount of overhead involved.

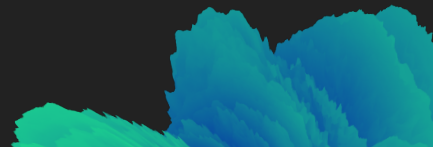


InstanceID-keyed Dictionaries!

- All UnityEngine.Objects have an Instance ID
  - Unique!
  - Never changes over lifetime of application!
  - Just an integer!
- *myObj.GetInstanceID()*



```
public class ThingCaller : MonoBehaviour {

    private ThingToCall[] objs;
    private int[] ids;

    private Dictionary<ThingToCall, int> fooCumulativeObj;
    private Dictionary<int, int> fooCumulativeId;
```

```
void Start () {
   objs = new ThingToCall[NUM_OBJS];
   ids = new int[NUM_OBJS];
   fooCumulativeObj = new Dictionary<ThingToCall, int>();
   fooCumulativeId = new Dictionary<int, int>();

   var template = new GameObject();
   template.AddComponent<ThingToCall>();
   for (int i = 0; i < NUM_OBJS; ++i) {
      var newGo = Object.Instantiate(template);
      objs[i] = newGo.GetComponent<ThingToCall>();
      ids[i] = objs[i].GetInstanceID();

      fooCumulativeObj.Add(objs[i], 0);
      fooCumulativeId.Add(ids[i], 0);
   }
}
```

```
void Update () {
   ShuffleThings();
   for (int i = 0; i < NUM_OBJS; ++i) {
      var baseObj = objs[i];

      int val = fooCumulativeObj[baseObj];
      val += objs[i].SomeMethod();
      fooCumulativeObj[baseObj] = val;
   }
}
```

```
void Update () {
   ShuffleThings();
   for (int i = 0; i < NUM_OBJS; ++i) {
      var baseObj = objs[i];
      var baseId = baseObj.GetInstanceID();

      int val = fooCumulativeId[baseId];
      val += objs[i].SomeMethod();
      fooCumulativeId[baseId] = val;
   }
}
```

```
void Update () {
   ShuffleThings();
   for (int i = 0; i < NUM_OBJS; ++i) {
      var baseObj = objs[i];
      var baseId = ids[i];

      int val = fooCumulativeId[baseId];
      val += objs[i].SomeMethod();
      fooCumulativeId[baseId] = val;
   }
}
```

How will it perform?

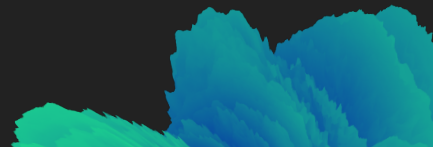
Cached IDs are best, duh.

(msec)	OSX, Mono	iPad 2, IL2CPP
Object key	0.15	1.33
Int key (uncached)	0.19	0.91
Int key (cached)	0.06	0.54

Why?

- Calling *GetInstanceID* invokes unsafe code.
  - Mono's JIT compiler optimizes the loop poorly.
  - Less of a problem under IL2CPP.

- C#'s Integer-keyed Dictionary is highly optimized.



When to use this

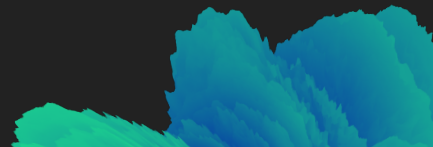
- Do *not* do this for all Object-keyed Dictionaries!

- Reserve for key data structures
  - Dictionaries that are indexed hundreds or thousands of times per frame.

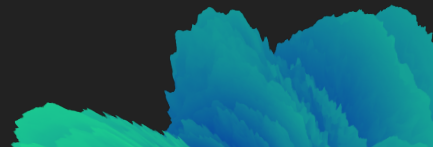
Recursion

```
public class TreeSearcher : MonoBehaviour {
    private const int NUM_NUMBERS = 1001;
    private int[] numbers;

    private BSTree tree;

     void Start () {
        tree = new BSTree();
        numbers = new int[NUM_NUMBERS];
        for (int i = 0; i < NUM_NUMBERS; ++i) {
            numbers[i] = i;
            tree.Add(i);
        }
        tree.Balance();
    }
```

```
void Update() {
   int numsFound = 0;
   ShuffleNumbers();
   for(int i = 0; i < NUM_NUMBERS; ++i) {
      BSTreeNode n = tree.FindRecursive(numbers[i]);
      if(n != null) {
         numsFound += 1;
      }
   }
   // … log out numsFound to prevent optimization
}
```

```
void Update() {
   int numsFound = 0;
   ShuffleNumbers();
   for(int i = 0; i < NUM_NUMBERS; ++i) {
      BSTreeNode n = tree.FindStack(numbers[i]);
      if(n != null) {
         numsFound += 1;
      }
   }
   // … log out numsFound to prevent optimization
}
```

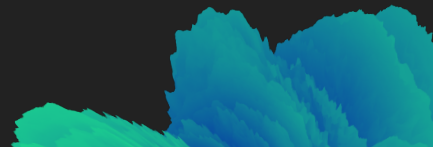



Searching a thousand-node BST x 1000

(msec)	OSX, Mono	iPad 2, IL2CPP	Editor
Recursive	2.9	6.9	13.1
Stack	1.9	6.5	13.1

Interpreting these results.

- On JIT platforms, recursion overhead is high.
- On cross-compiled platforms, the difference is smaller but still exists.
- Editor suppresses JIT optimizations and emits additional code to support debugging.



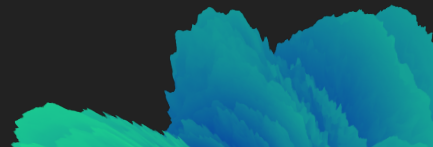
Inlining

Inlining?

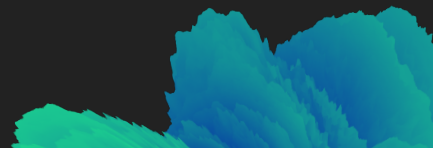
- Hypothetically the simplest way to eliminate method-call overhead.
- *Hypothetically.*

- In reality, the Unity C# compiler is very averse to inlining.

```
public class Inliner : MonoBehaviour {

    public const int NUM_RUNS = 100;
    public string searchWord;
    public TextAsset originalText;

    private string[] lines;

    void OnEnable() {
        lines = (originalText != null)
           ? originalText.text.Split('\n')
           : new string[0];
    }
```

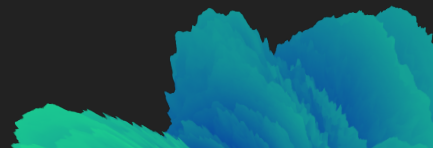



```
private int TestInlining() {
    int numLines = lines.Length;
    int count = 0;
    for (int i = 0; i < NUM_RUNS; ++i)
    {
        count = 0;
        for (int lineIdx = 0; lineIdx < numLines; ++lineIdx)
        {
            count +=
               FindNumberOfInstancesOfWord(
                  lines[lineIdx],
                  searchWord);
        }
    }
}
```

```
private int FindNumberOfInstancesOfWord(string text,
                                        string word) {
    int idx, count, textLen, wordLen;
    count = idx = 0;
    textLen = text.Length;
    wordLen = word.Length;
    while(idx >= 0 && idx < textLen) {
        idx = text.IndexOf(word, idx,
                           System.StringComparison.Ordinal);
        if (idx < 0)
            break;
        idx += wordLen;
        count += 1;
    }
    return count;
}
```


```
[MethodImplAttribute(MethodImplOptions.NoInlining)]
private int FindNumberOfInstancesOfWord(string text,
                                        string word) {

    // … goes over line with String.IndexOf
    // … counts # of times word appears
    // … and returns the result
}
```

New in the 4.6 runtime!

```
[MethodImplAttribute(MethodImplOptions.AggressiveInlining)]
private int FindNumberOfInstancesOfWord(string text,
                                        string word) {

    // … goes over line with String.IndexOf
    // … counts # of times word appears
    // … and returns the result
}
```

```
private int TestInliningManual() {
    int numLines = lines.Length;
    int count = 0;
    for (int i = 0; i < INLINE_LIMIT; ++i) {
        count = 0;
        for (int lineIdx = 0; lineIdx < numLines; ++lineIdx) {
            int idx = 0;
            int textLen = lines[lineIdx].Length;
            int wordLen = searchWord.Length;
            while (idx >= 0 && idx < textLen) {
               idx = lines[lineIdx].IndexOf(searchWord, idx,
                                            System.StringComparison.Ordinal);
               if (idx < 0)
                  break;
               idx += wordLen;
               count += 1;
            }
        }
    }
    return count;
}
```

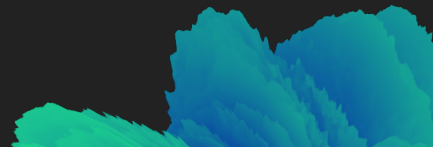
Copy-pasted function body

Inlining!

(msec)	OSX, Mono	iPad 2, IL2CPP
3.5, No inlining	5.2	26.0
3.5, Manual	4.6	23.8
4.6, Aggressive	5.6	26.5
4.6, Manual	5.4	24.5

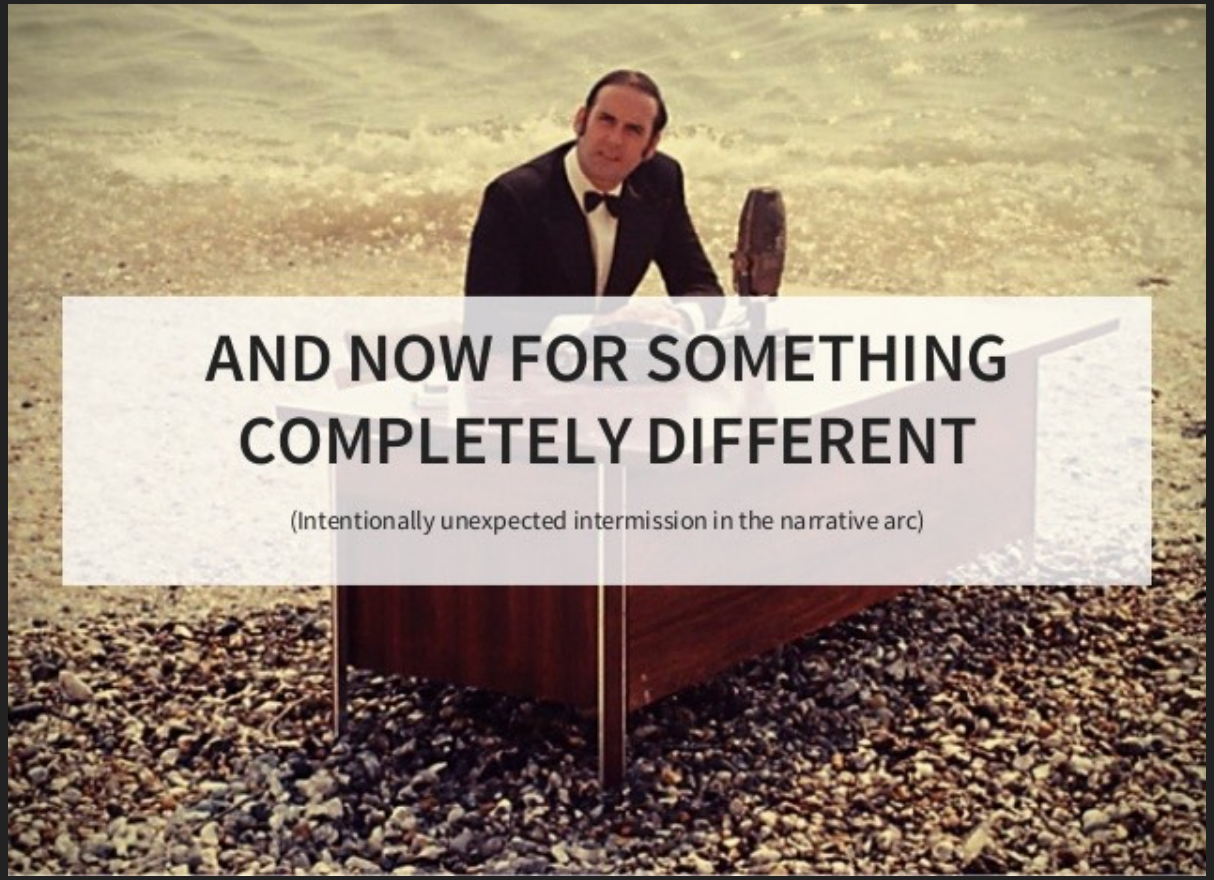
Results?

- Manual inlining is useful for small methods on extremely hot paths.
  - Maintenance nightmare. Use sparingly!
  - *AggressiveInlining* mostly eliminates the difference in 4.6, and is much easier to maintain.
- *AggressiveInlining* is not yet implemented for IL2CPP.
  - On the roadmap though.



Other places where this is relevant...

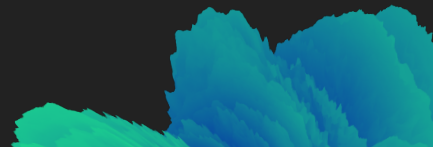
- Trivial properties generate methods
  - Yes, these are method calls!
  - `public int MyProperty { get; set; }`
- Convert to public fields in hot code-paths.
  - `public int MyProperty;`



UnityUI

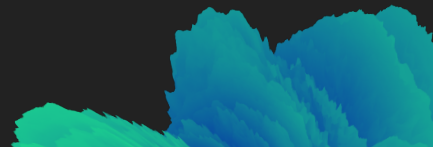
The Absolute Basics

- Canvases generate meshes & draw them
  - Drawing happens once per frame
  - Generation happens when “something changes”
- “Something changes” = “1 or more UI elements change”
  - Change one UI element, dirty the whole canvas.
  - Yes, *one*.



What's a rebuild? Theoretically...

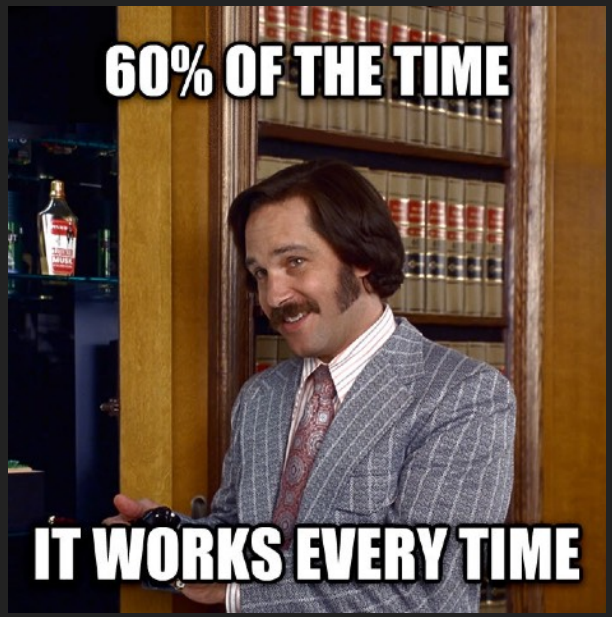
- Three steps:
  - Recalculate layouts of auto-layouted elements
  - Regenerate meshes for all *enabled* elements
    - *Yes, meshes are still created when alpha=0!*
  - Regenerate materials to help batch meshes



What's a rebuild? In Reality...

- Usually, all systems are dirtied instead of individually.

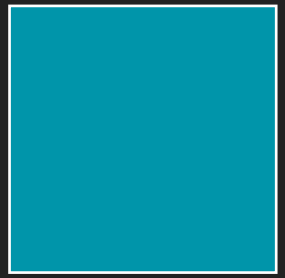
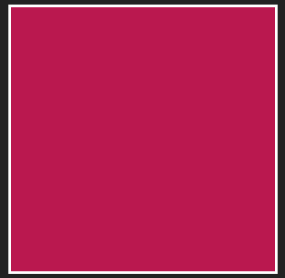
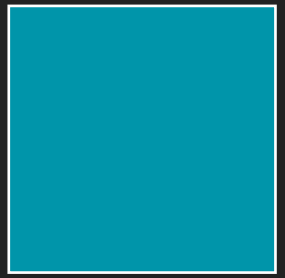
- Notable exceptions:
  - *Color* property
  - *fill** properties on Image



After all that...

- Canvas takes meshes, divides them into batches.
  - Sorts the meshes by depth (hierarchy order)
  - *Yes, this involves a sort() operation!*
- All UI is transparent. No matter what.
  - Overdraw is a problem!

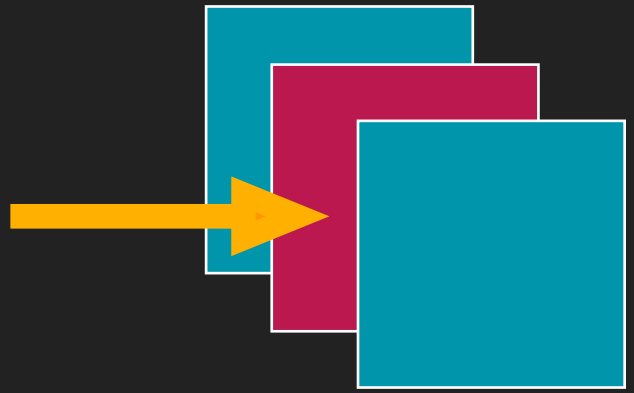
Trivially batchable.

Colors represent different materials.

Not batchable!

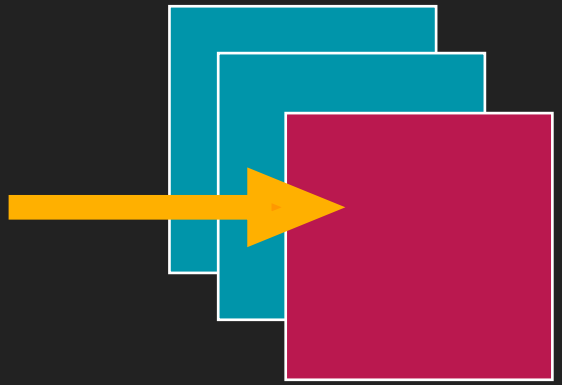
Red quad is interposed



Colors represent different materials.

This is batchable!

Red quad is overlaid

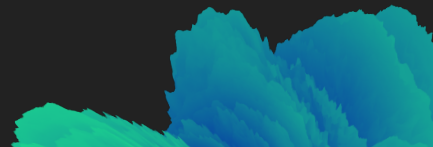


Colors represent different materials.

The Vicious Cycle

- Sort means performance drops faster-than-linear as number of drawable elements rises.

- Higher number of elements means a higher chance any given element will change.



Slice up your canvases!

- Add more canvases.
- Each canvas owns its own set of UI elements.
  - Elements on different canvases will not batch.

- Main tool for constraining size of UI batches.

Nesting Canvases

- Canvases can be created within other canvases.

- Child canvases inherit rendering settings.
  - Maintain own geometry.
  - Perform own batching.

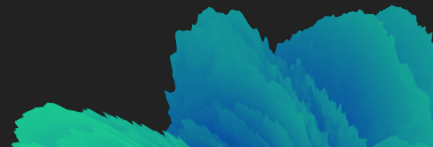
General Guidelines

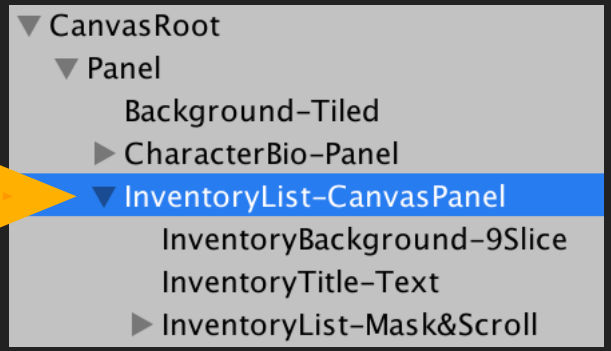
- Subdivide big canvases
  - Group elements that are updated simultaneously
- Separate dynamic elements from static ones
  - Backgrounds, static labels, etc.
  - Spinners, throbbers, blinking carets, etc.

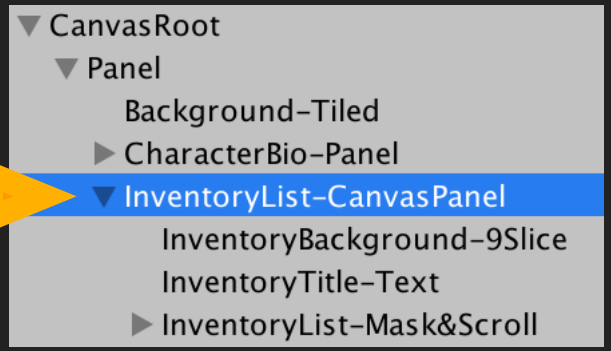
Scroll Rect,
1000 Items

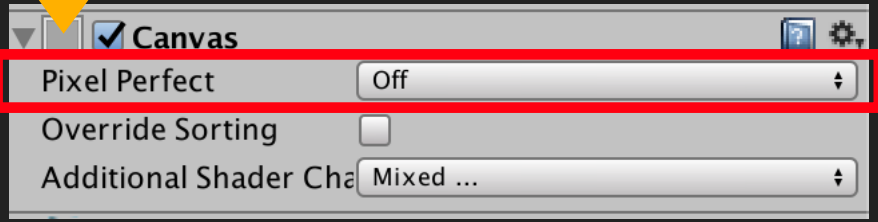
25ms just to scroll a list!

Overview	Total	Self	Calls	GC Alloc	Time ms
▶ PreLateUpdate.ScriptRunBehaviourLateUpdate	80.9%	0.0%	1	24 B	21.30
▶ PostLateUpdate.PlayerUpdateCanvases	14.7%	0.0%	1	102 B	3.87
▶ Update.ScriptRunBehaviourUpdate	1.0%	0.0%	1	0 B	0.27









unity
Unite
Europe
2017

Down to “only” 5ms

Overview	Total	Self	Calls	GC Alloc	Time ms
▶ PostLateUpdate.PlayerUpdateCanvases	46.1%	0.0%	1	51 B	2.94
▶ PreLateUpdate.ScriptRunBehaviourLateUpdate	32.8%	0.0%	1	24 B	2.09
▶ Update.ScriptRunBehaviourUpdate	6.4%	0.0%	1	0 B	0.41

Where from here?

- Pool child UI elements in Scroll Rect
  - Minimizes cost of rebuilds

- Modify Scroll Rect code
  - Stop scrolling when velocity gets very small
    - Right now it will move items 0.0001 pixels…

GraphicRaycaster

What's it do?

- Translates screen/touch input into Events.
- Sends events to interested UI elements.

- Need one on every Canvas that requires input.
  - Including subcanvases.

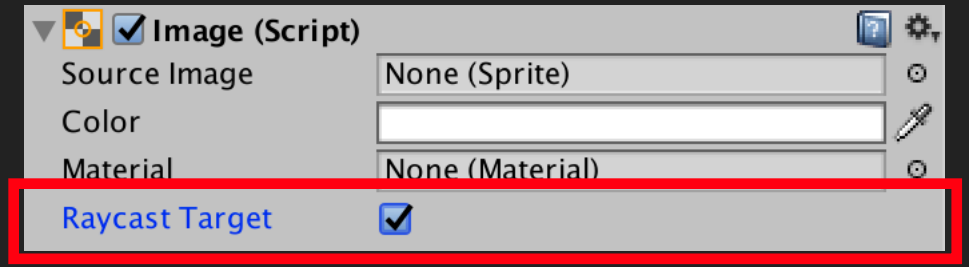
Not really a “raycaster”...

- With default settings, only tests UI graphics.

- Performs point-rectangle intersection check for each RectTransform marked interactive.
  - Yes, just a simple loop.

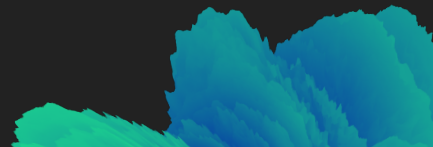
First tip!

- Turn off “Raycast Target” where possible.
- Directly Reduces number of per-frame checks.

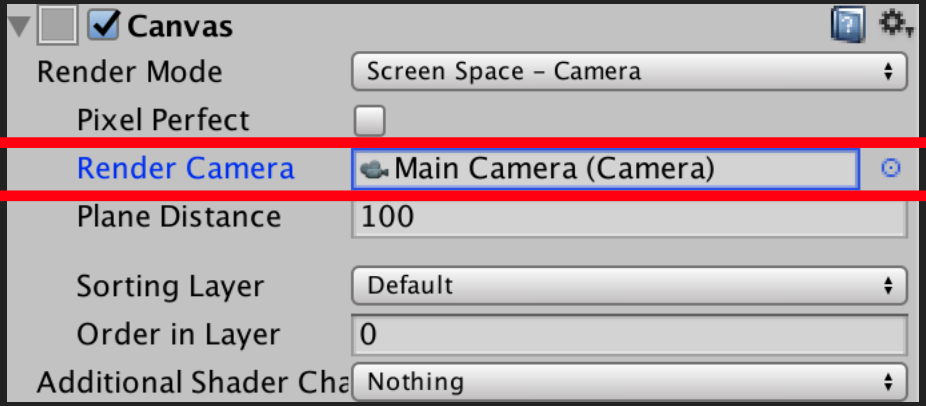


2D/3D Geometry?

- “Worldspace” or “Screen Space - Camera” canvas
- Blocking mask can select whether to cast against 2D/3D geometry
- Will then perform a 2D/3D raycast outwards from the Canvas’ event camera
  - Limited to camera’s visible frustum



… “event camera”?



What if that isn’t set?

```
public override Camera eventCamera
{
    get
    {
        if (canvas.renderMode ==
RenderMode.ScreenSpaceOverlay
            || (canvas.renderMode ==
RenderMode.ScreenSpaceCamera && canvas.worldCamera == null))
            return null;

        return canvas.worldCamera != null ?
canvas.worldCamera : Camera.main;
    }
}
```


unity
Unite
Europe
2017

eventCamera is accessed 7-10 times per frame for Raycasting.

Per World-Space Canvas.

Camera.main
=
FindObjectWithTag

10 Canvases + X tagged objects

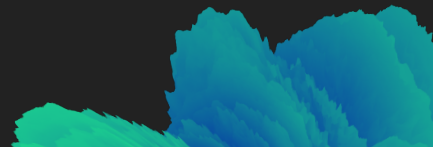
X

	Unassigned	Assigned
10	0.07	0.06
100	0.13	0.07
1000	0.85	0.07

Timings are in milliseconds

What can you do about this?

- Avoid use of *Camera.main*
  - Cache references to Cameras.
  - Create a system to track the main camera.
- Matters mostly in per-frame code



What about in Unity UI?

- “World Space” canvas?
  - Always assign a World Camera

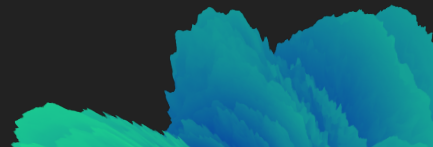
- Minimize number of Graphic Raycasters
  - Do not add them to non-interactive UIs!

Layouts!

Setting dirty flags should be cheap, right?

- Layout works on a “dirty flag” system.
  - When an element changes in a way that would invalidate the surrounding layout, it must notify the layout system.

- Layout system = Layout Groups



Wait but...

- Layout Groups are components too!
  - Vertical Layout Group, Grid Layout Group...
  - Scroll Rect
- They are always Components on parent GameObjects of Layout Elements.

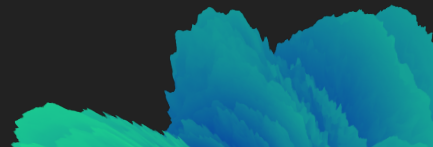
How do Layout Elements know which Layout Group to dirty?

Walking the code...

- https://bitbucket.org/Unity-Technologies/ui/

- Graphic.cs :: SetLayoutDirty
  - Base class for all drawable UI elements
    - UI Text, UI Image, etc.

```
public virtual void SetLayoutDirty()
{
    if (!IsActive())
        return;

    LayoutRebuilder.MarkLayoutForRebuild(rectTransform);

    if (m_OnDirtyLayoutCallback != null)
        m_OnDirtyLayoutCallback();
}
```

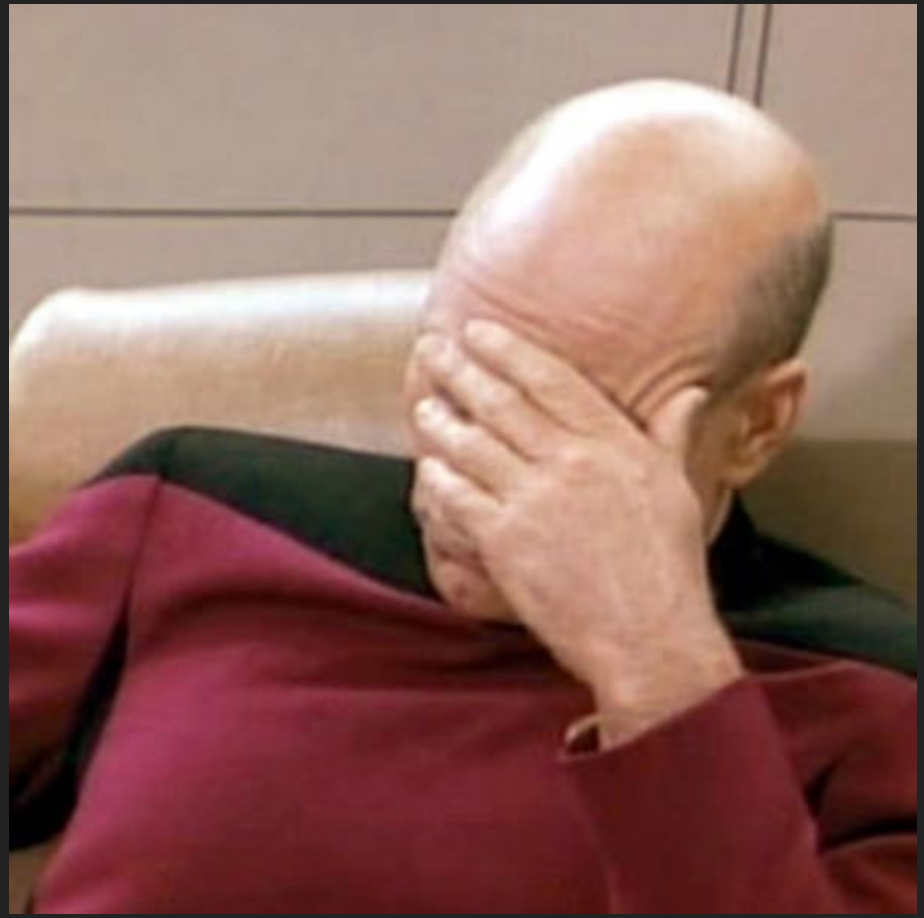
```
public static void MarkLayoutForRebuild(RectTransform rect)
{
    // ...
    var comps = ListPool<Component>.Get();
    RectTransform layoutRoot = rect;
    while (true)
    {
        var parent = layoutRoot.parent as RectTransform;
        if (!ValidLayoutGroup(parent, comps))
            break;
        layoutRoot = parent;
    }
    // ...
}
```


Immediately, we see something...

- MarkLayoutForRebuild searches for the Layout Group closest to the root of the hierarchy.
  - Stops if it doesn't find a Layout Group.
- Number of checks rises linearly with number of consecutively nested layout groups.

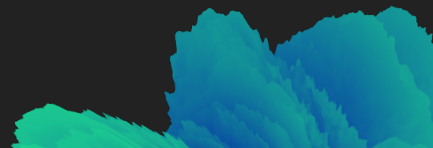
What’s ValidLayoutGroup do?

```
private static bool ValidLayoutGroup
    (RectTransform parent, List<Component> comps)
{
    if (parent == null)
        return false;

    parent.GetComponents(typeof(ILayoutGroup), comps);
    StripDisabledBehavioursFromList(comps);
    var validCount = comps.Count > 0;
    return validCount;
}
```

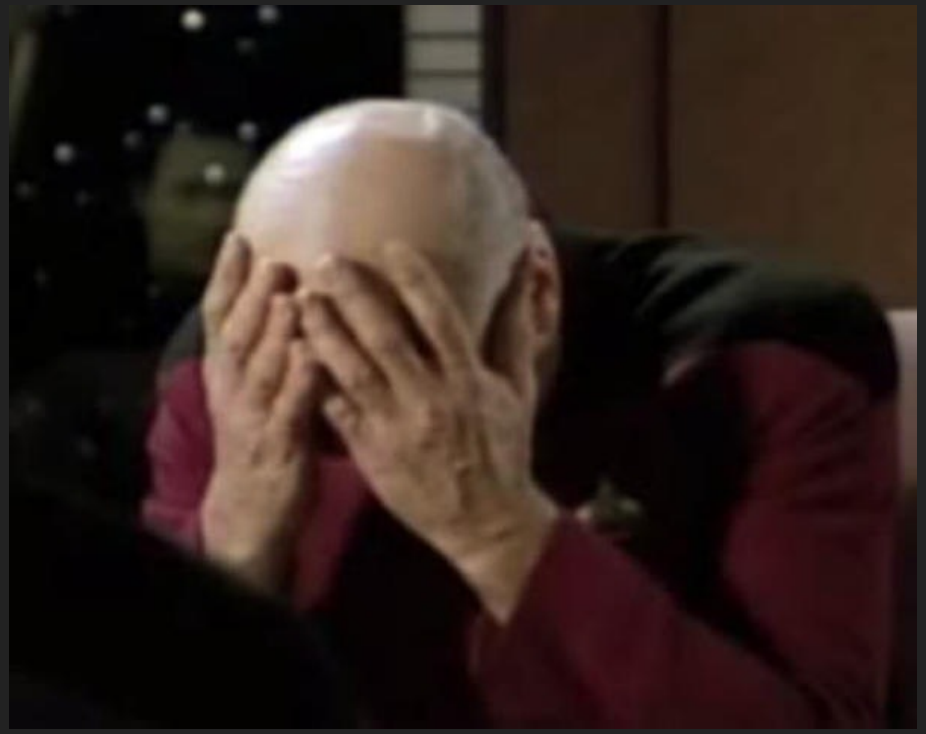


unity
Unite
Europe
2017

```
static void StripDisabledBehavioursFromList
    (List<Component> components)
{
    components.RemoveAll(e =>
        e is Behaviour &&
        !((Behaviour)e).isActiveAndEnabled);
}
```

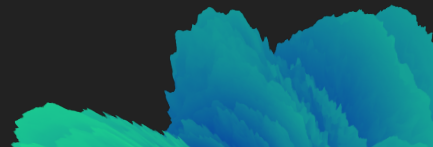
LINQ + List Modification in the hot path...



Takeaways

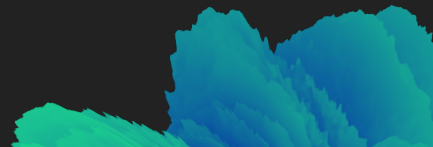
- Every UI element that tries to dirty its layout will perform at least 1 GetComponents call.
  - *Each time* it marks its layout dirty.

- Each layout group multiplies this cost.
  - Nested layout groups are particularly bad.



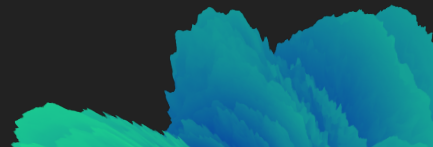
Things that mark layouts dirty...

- OnEnable & OnDisable
- Reparenting
  - Twice! Once for old parent, once for new parent
- OnDidApplyAnimationProperties
- OnRectTransformDimensionsChanged



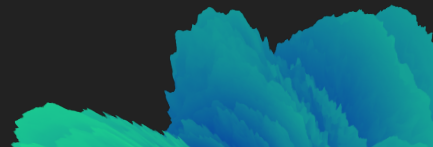
OnRectTransformDimensionsChanged?

- Change Canvas Scaling Factor, Camera
  - Sent to all elements on Canvas
- Change Transform anchors, position
  - Sent to all child RectTransforms
    - All the way to the bottom of the hierarchy!



What else?

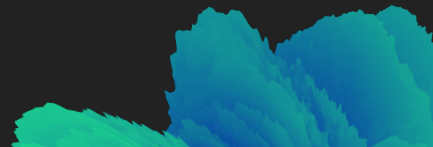
- Images
  - Changing *sprite* or *overrideSprite*
- Text
  - Changing most properties
  - Particularly: *text*, *alignment*, *fontSize*



Yeah, EVERYTHING dirties Layout!

Enough. Let's have some solutions.

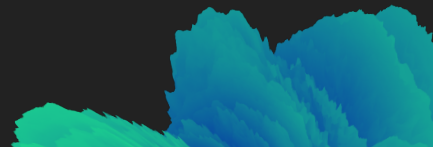
- Avoid Layout Groups wherever possible.
  - Use anchors for proportional layouts.
- On hot UIs with dynamic number of elements, consider writing own code to calculate layouts.
  - Update this on-demand instead of with every single change.



Pooling UI objects?

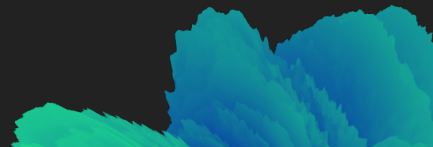
- Maximize number of dirty calls that will no-op.

- Disable *first*, then reparent into pool.
- Removing from pool?
  - Reparent *first*, then update data, *then* enable.



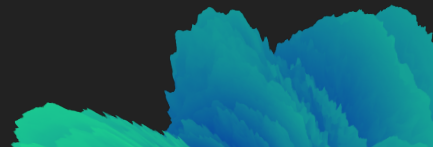
Hiding a canvas?

- Consider disabling the *Canvas* component.
  - Stops drawing the Canvas
  - Does not trigger expensive OnDisable/ OnEnable callbacks on UI hierarchy.
- Be careful to disable child components that run expensive per-frame code!



In extreme cases...

- Build your own UnityEngine.UI.dll from source.
- Eliminate LayoutRebuilder.
- This is a lot of work.



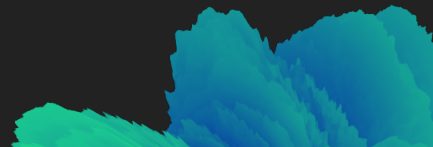
Animators on your UI?

DON’T

Beware of Animators on UI!

- Animators will dirty their elements *every frame*.
  - Even if value in animation doesn't change!
  - Animators have no no-op checks.

- Animators are OK on things that *always* change
- … or use your own code or tweening system



unity
Unite
Europe
2017

Thank you!