Concurrency and Parallel Computing in JavaScript*

Stephan Herhut, Research Scientist, Intel Labs

StrangeLoop 2013

Concurrency

$\neq$

Parallel Computing

Trying A Definition

Concurrency: Performing multiple tasks at the same time, possibly interleaved.

One Man Band by Andrew Malone, from http://www.flickr.com/photos/andrewmalone/5163238038/

Trying A Definition (2)

Parallel Computing: Multiple entities working at the same time, on one or multiple tasks

1967 Buick Assembly Line by Alden Jewell, from http://www.flickr.com/photos/autohistorian/7599444736

Agenda

As a web developer, you are surrounded by concurrency

- Event-driven asynchronous programming
- Promises
- Web Workers

and today's hardware is increasingly parallel

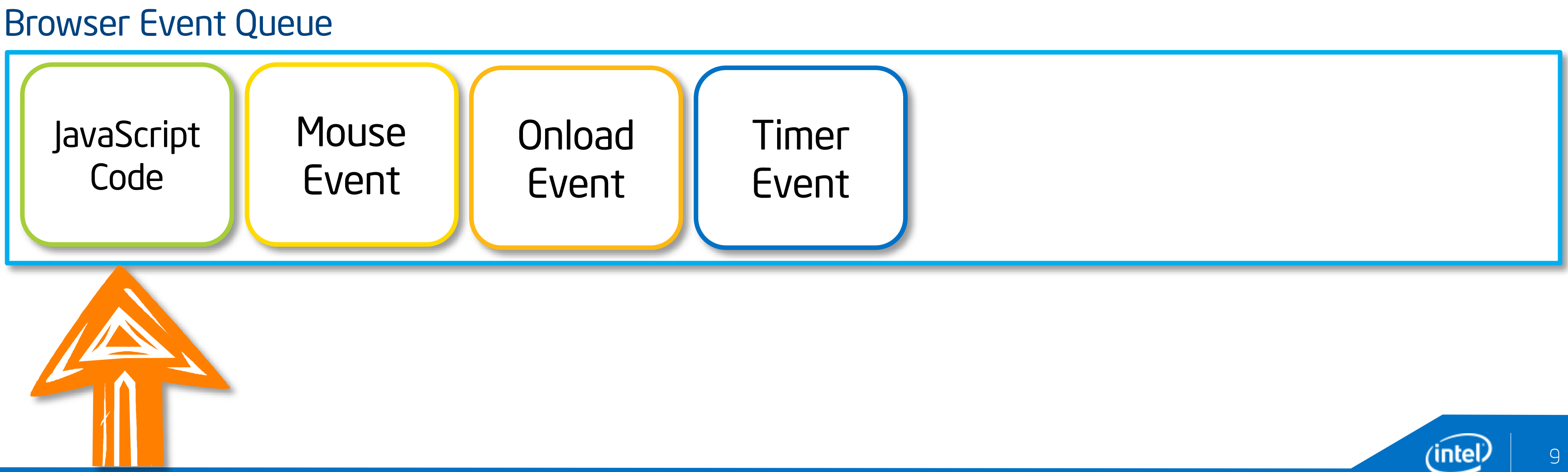
- SIMD units
- Multi-core processors
- Programmable GPUs

Concurrency

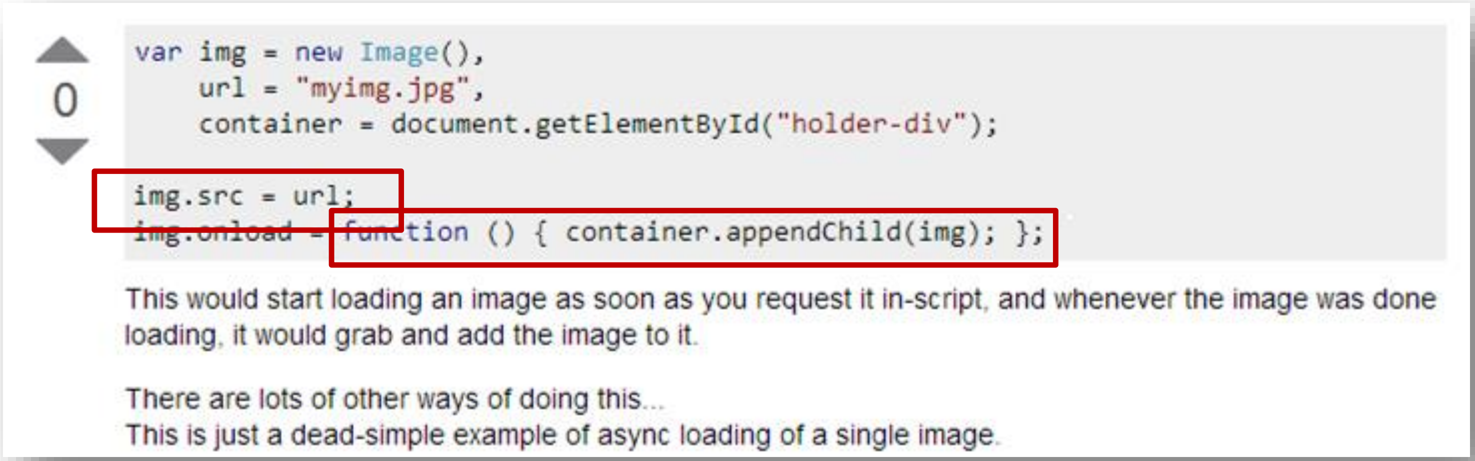
Asynchronous Programming, Web Workers and Promises

JavaScript* Execution Model

- Single event queue
- Events run to completion, no interleaving
- Events may create new events
- Ordering of events may be non-deterministic



Example: Asynchronous Image Loading

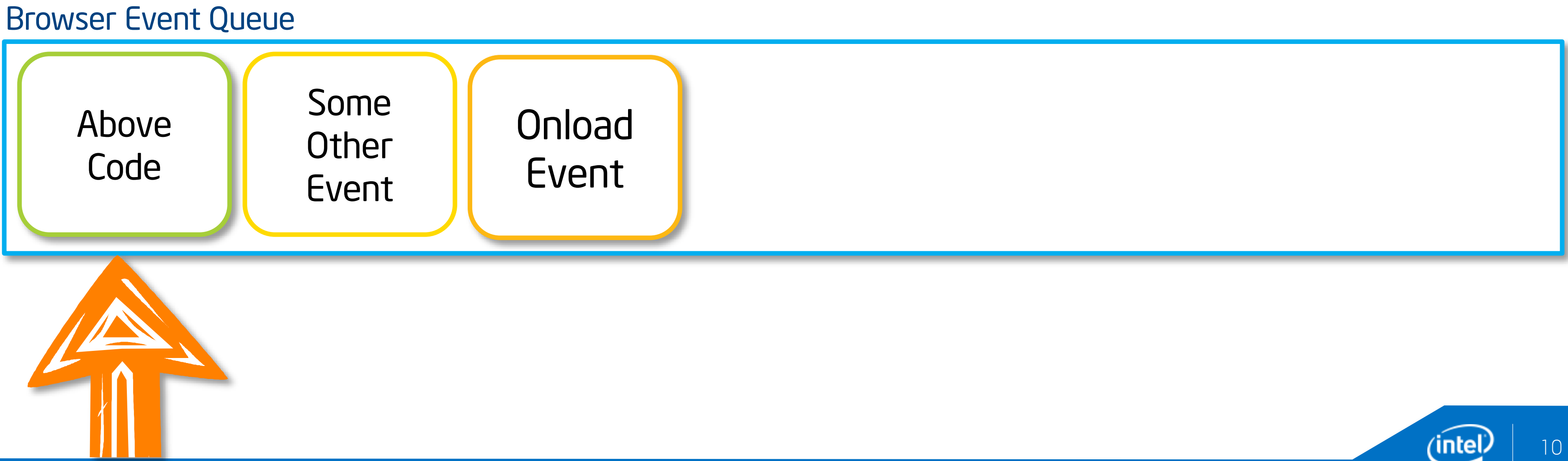


```
var img = new Image(),
    url = "myimg.jpg",
    container = document.getElementById("holder-div");

img.src = url;
img.onload = function () { container.appendChild(img); };
```

This would start loading an image as soon as you request it in-script, and whenever the image was done loading, it would grab and add the image to it.

There are lots of other ways of doing this...
This is just a dead-simple example of async loading of a single image.

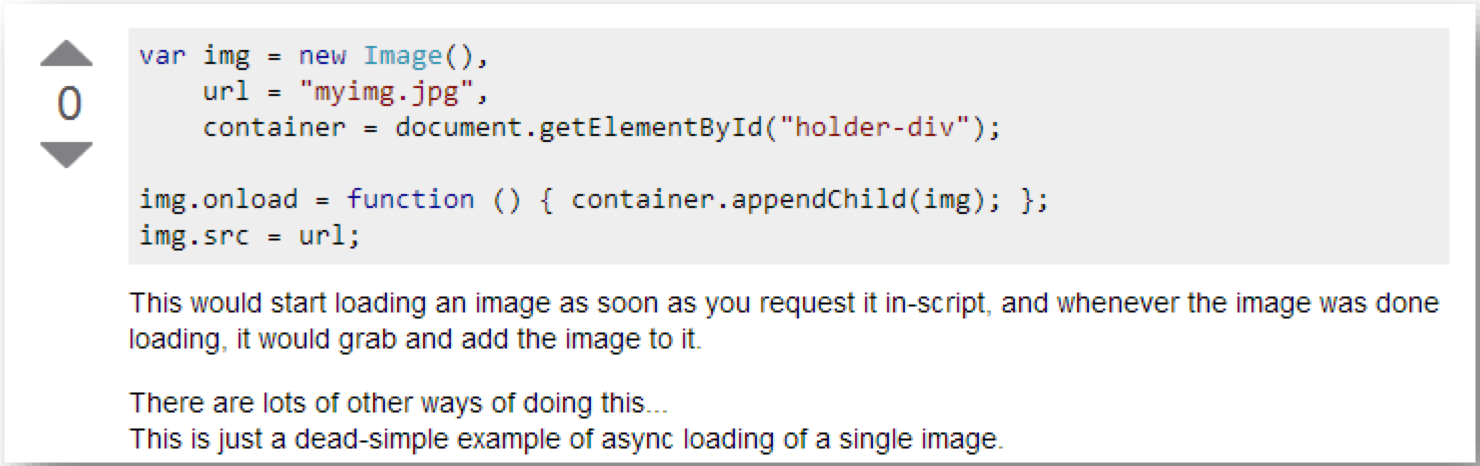
http://stackoverflow.com/questions/15999760/load-image-asynchronous



Fixed: Asynchronous Image Loading

Always make sure your callback is defined before an event can fire

- Although JavaScript* events always run to completion before the next event is scheduled, background tasks may run interleaved or even in parallel.



0

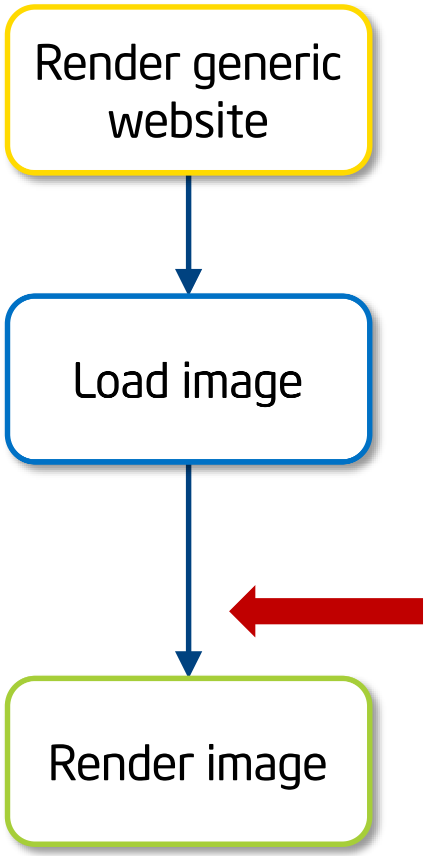
```
var img = new Image(),
    url = "myimg.jpg",
    container = document.getElementById("holder-div");

img.onload = function () { container.appendChild(img); };
img.src = url;
```

This would start loading an image as soon as you request it in-script, and whenever the image was done loading, it would grab and add the image to it.

There are lots of other ways of doing this...
This is just a dead-simple example of async loading of a single image.

http://stackoverflow.com/questions/15999760/load-image-asynchronous

Asynchronous Image Loading Flow

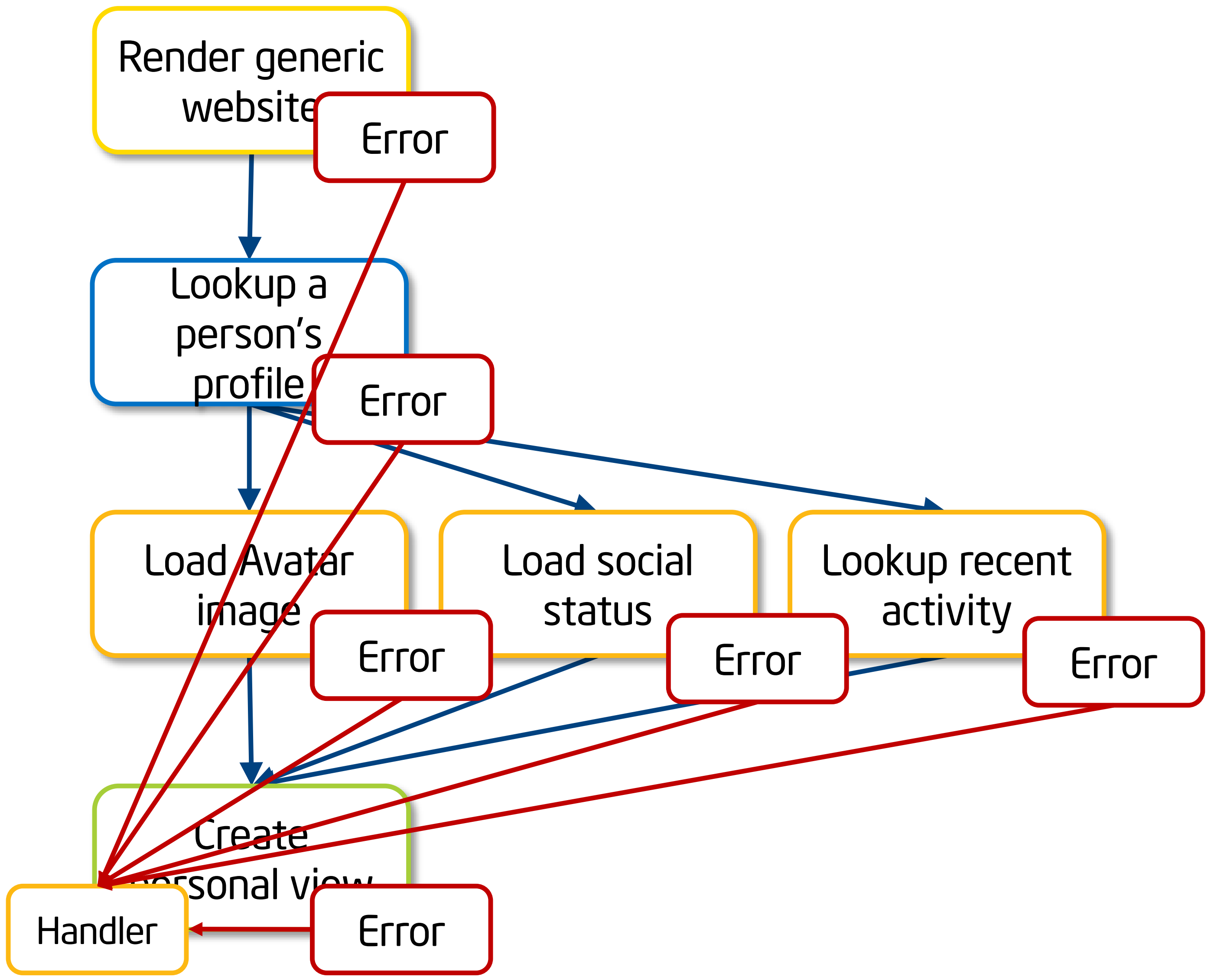


```
var img = new Image(),
    url = "myimg.jpg",
    container = document.getElementById("holder-div");

img.onload = function () { container.appendChild(img); };
img.src = url;
```

dependency is encoded using an *onload* callback

```
function () { container.appendChild(img); }
```

A More Complex Scenario: Personalization



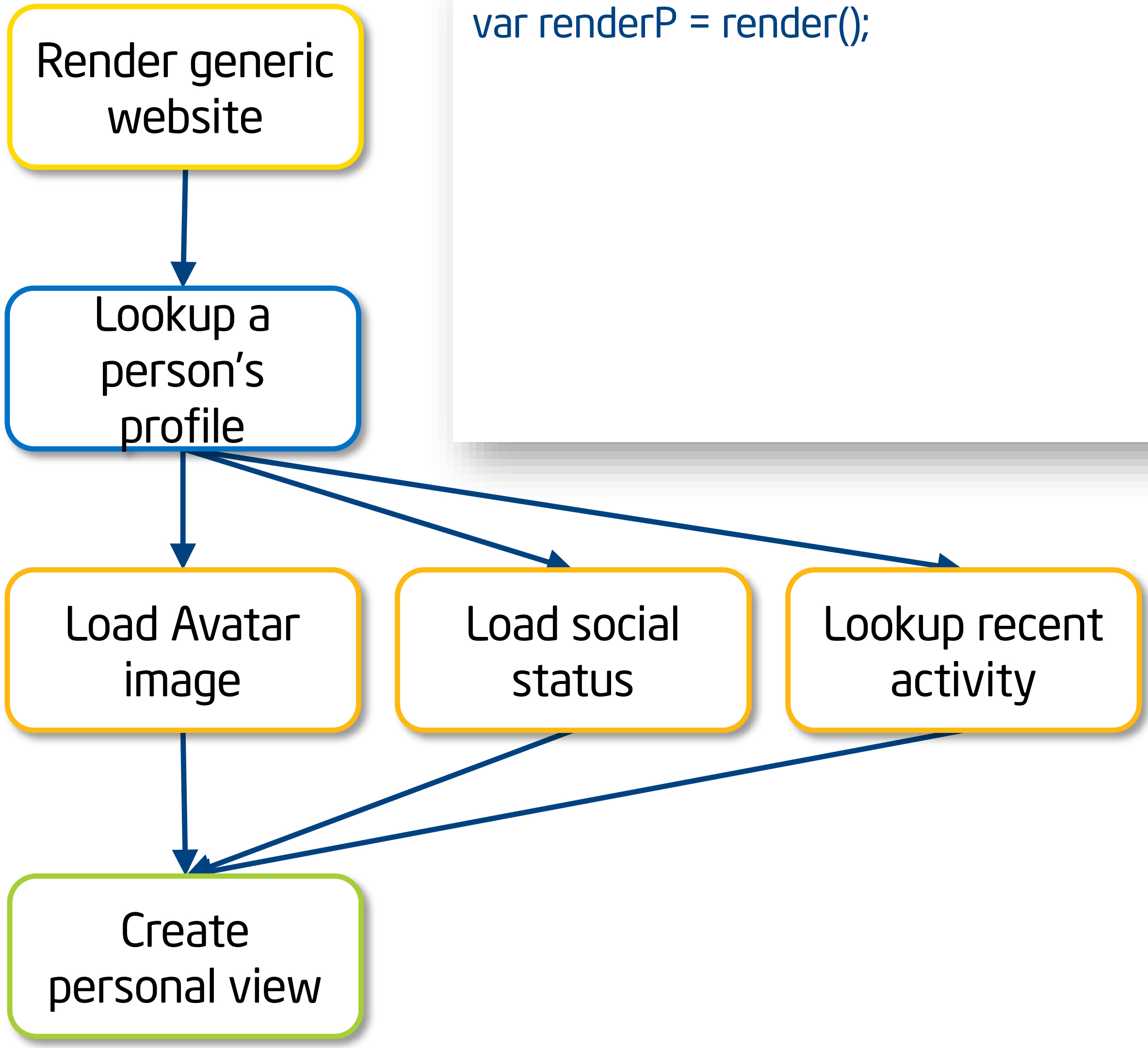
Enter: Promises

Promises enable a direct encoding of an asynchronous flow in data

- Promise objects encode state of asynchronous operation
- Different methods to encode dependencies like sequence, concurrency and joins
- Errors propagate along the flow until caught

Luxury Gold Ring Designs by epSos.de, from http://www.flickr.com/photos/epsos/8904048841/

Encoding Flow with Promises



```
var renderP = render();
```

Internal State of a Promise

pending

- The operation has not yet completed

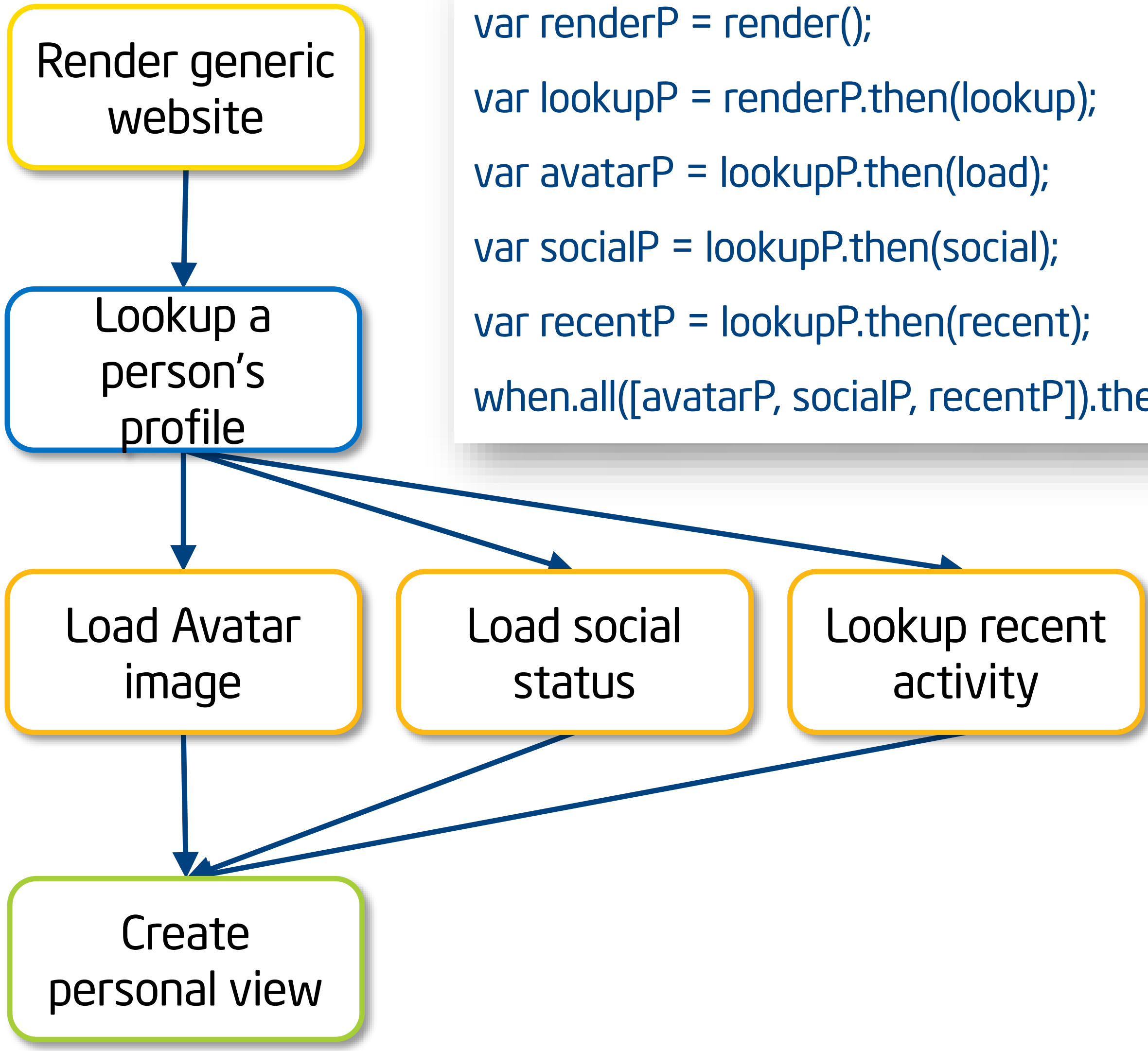
fulfilled

- The operation completed and has produced a valid result

rejected

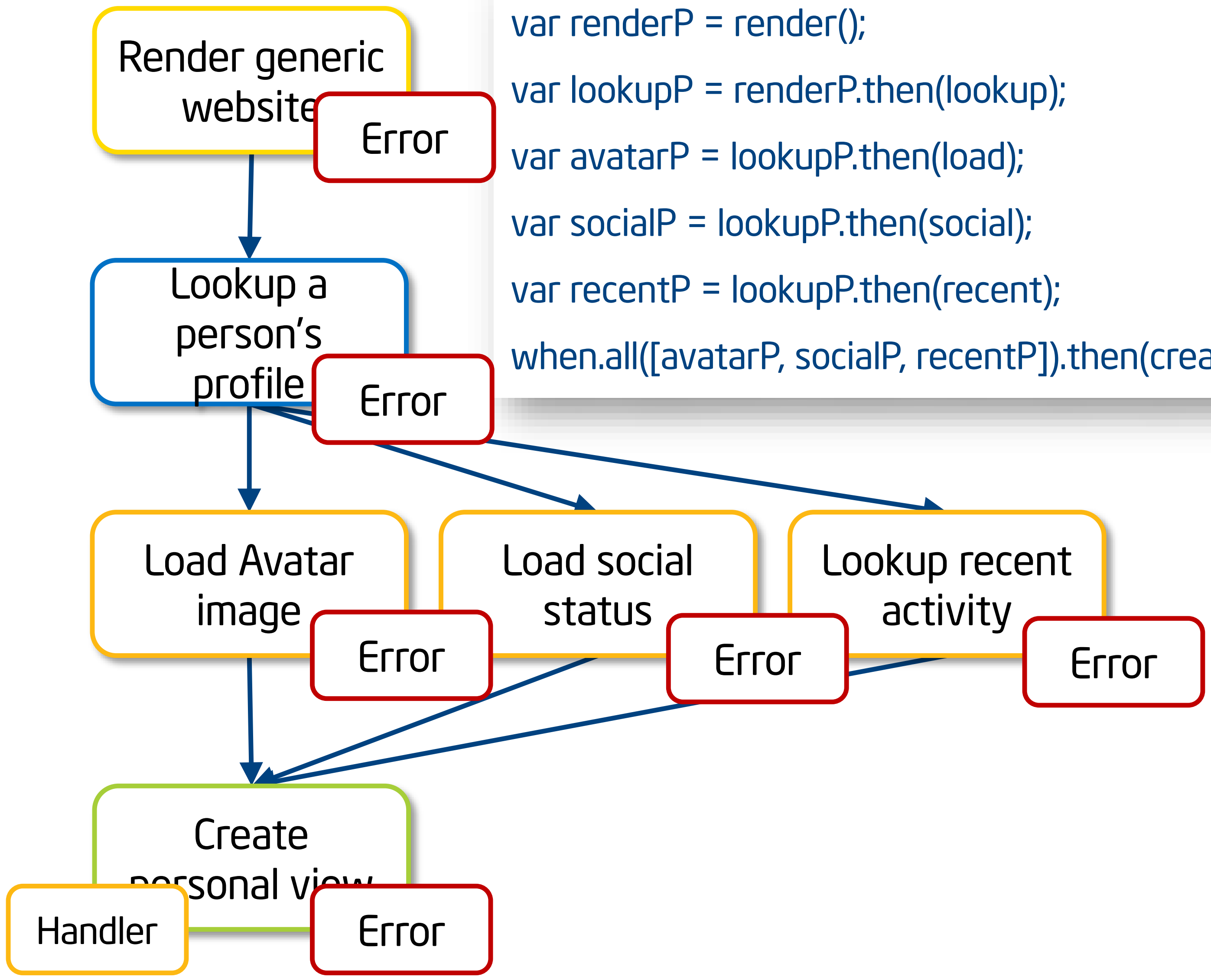
- The operation has failed or one of its dependencies have failed

Encoding Flow with Promises



```
var renderP = render();
var lookupP = renderP.then(lookup);
var avatarP = lookupP.then(load);
var socialP = lookupP.then(social);
var recentP = lookupP.then(recent);
when.all([avatarP, socialP, recentP]).then(create);
```

Handling Errors with Promises



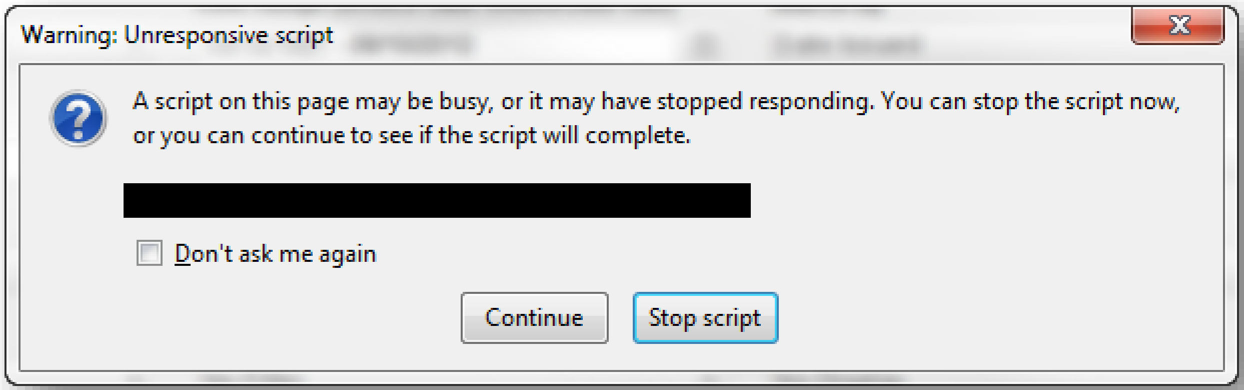
```
var renderP = render();
var lookupP = renderP.then(lookup);
var avatarP = lookupP.then(load);
var socialP = lookupP.then(social);
var recentP = lookupP.then(recent);
when.all([avatarP, socialP, recentP]).then(create, handler);
```

More On Promises

Many uses, many specifications

- https://github.com/promises-aplus
- https://github.com/cujojs/when
- http://api.jquery.com/category/deferred-object/
- …

Long Running Scripts



Warning: Unresponsive script

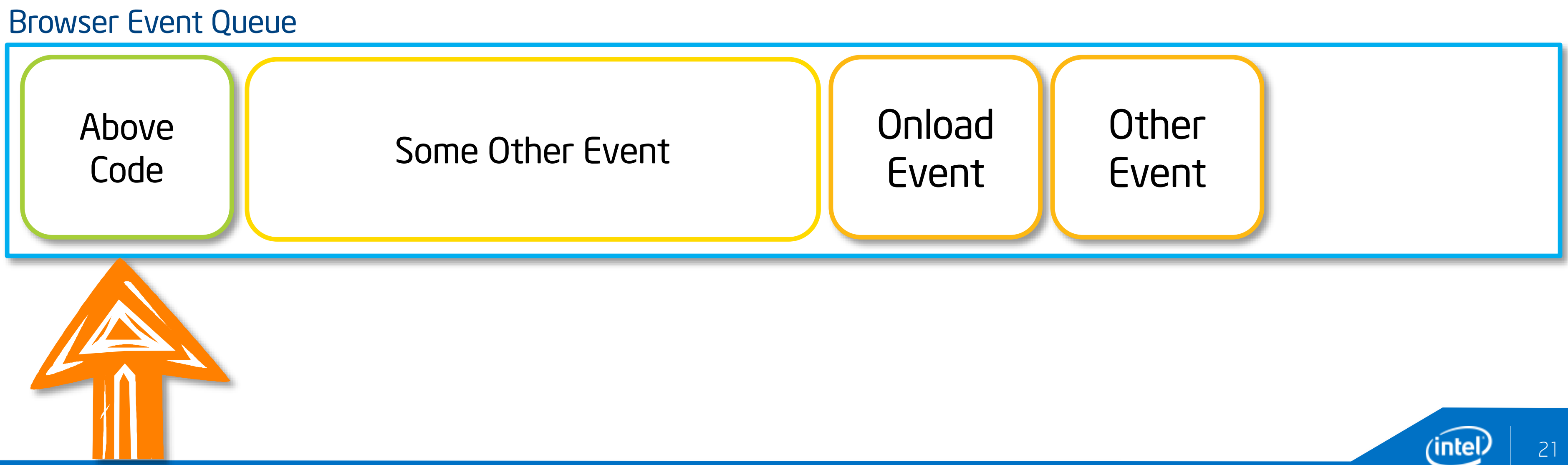
A script on this page may be busy, or it may have stopped responding. You can stop the script now, or you can continue to see if the script will complete.

[redacted]

How to Handle Long Running Events

Event based model enables concurrency, yet

- Only one event executes at a time
- No even is ever interrupted (run to completion)



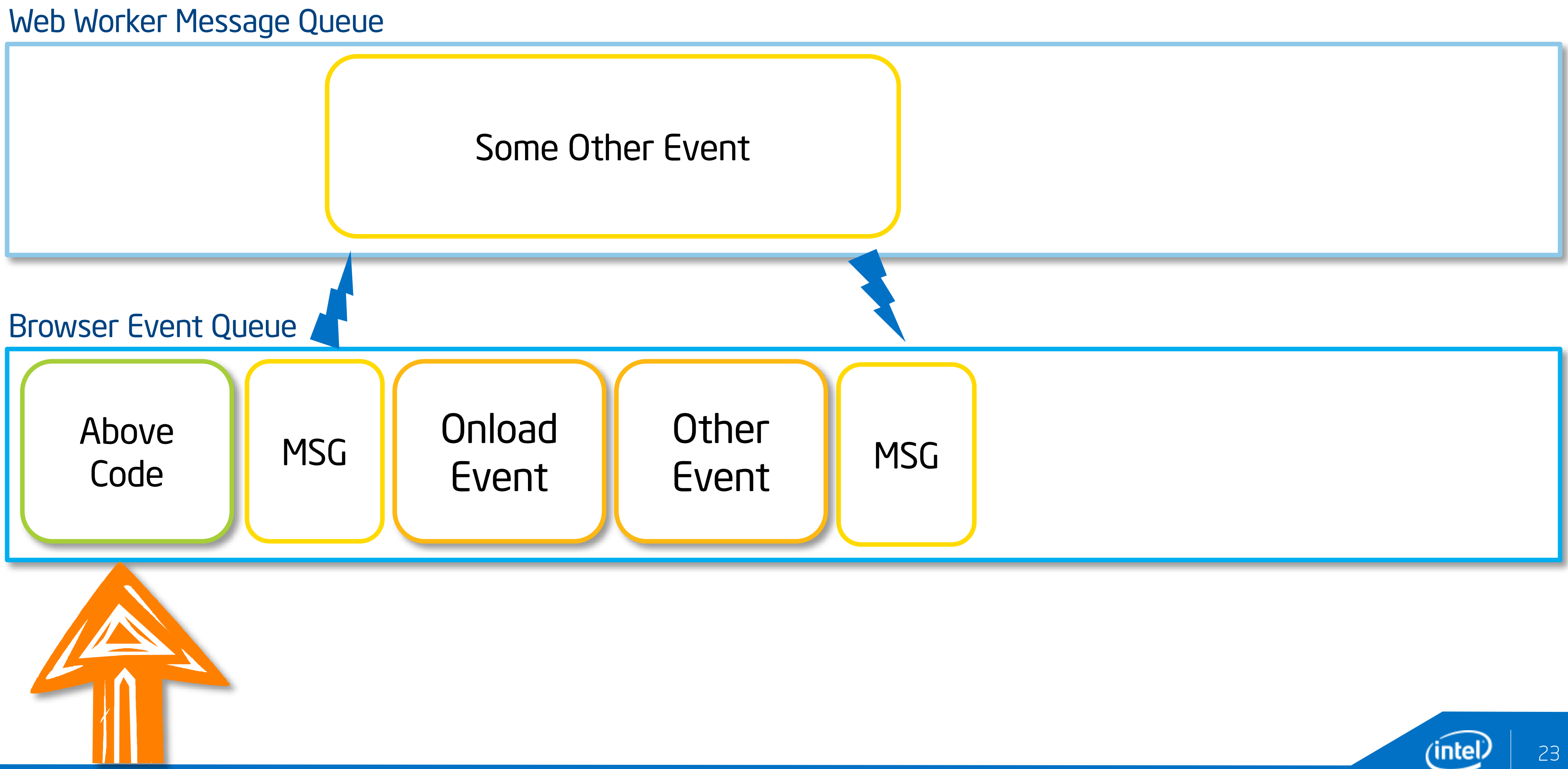
Enter: Web Workers

Spawn a second instance of the JavaScript* engine

- No shared state, communication via messages (actors model)
- Has its own event (message) queue

Spider and Web by Dwight Sipler, from http://www.flickr.com/photos/photofarmer/3948508109/

Concurrency Model with Web Worker



Parallel Computing

SIMD Units, Multi-Core CPUs and GPUs

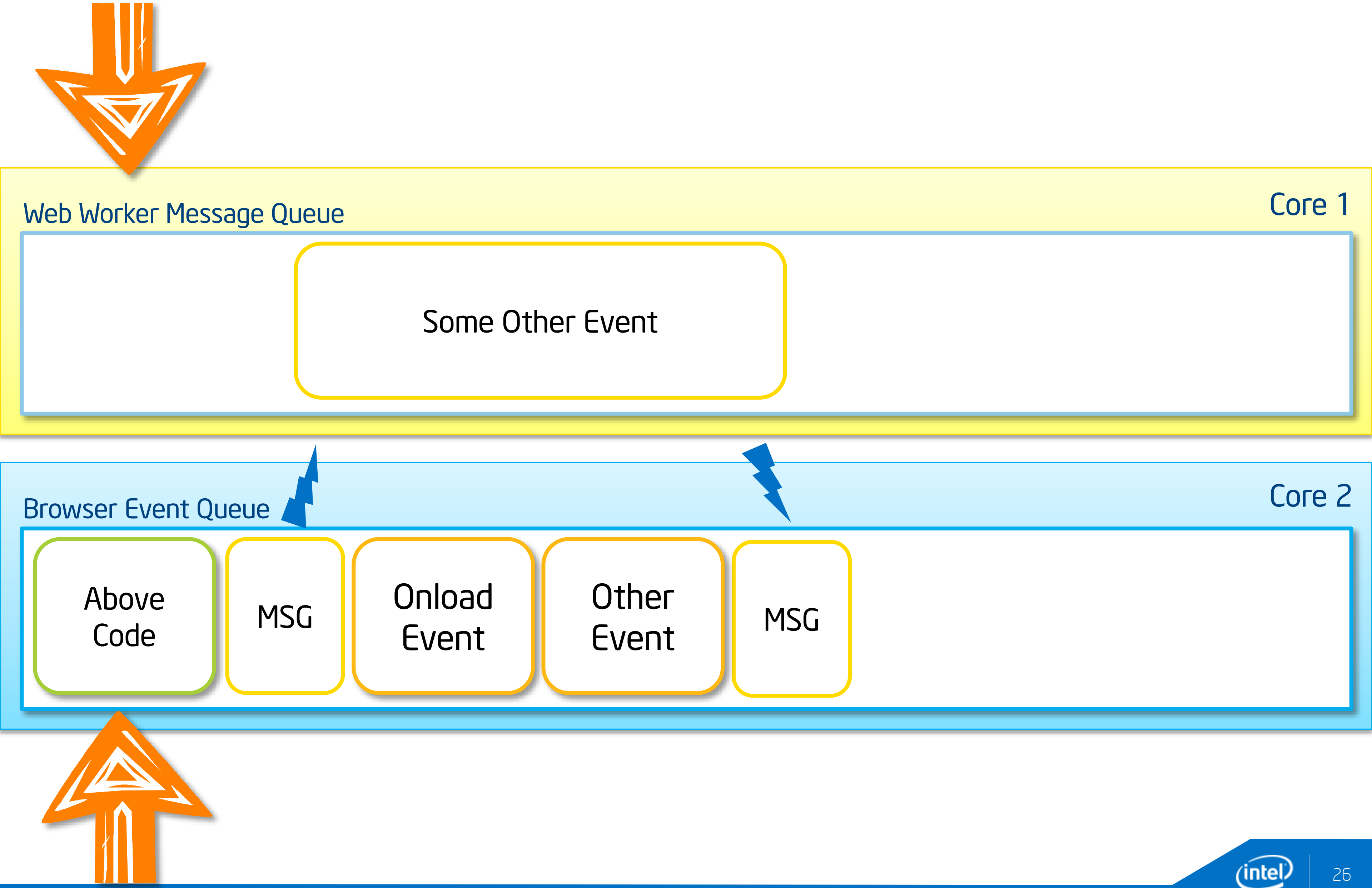
Modern Parallel Hardware

All form factors (server, laptop, tablet, phone) have dedicated parallel hardware

- Multi-core CPUs
- SIMD Extensions
- Programmable GPUs

For the best experience, web applications have to tap into those resources.

Task Parallelism with Web Workers



SIMD Programming

Single Instruction Multiple Data

- Typically operate on 4 or 8 element vectors
- Perform the same operation on each element
- Conditional control flow has to be encoded by merging values

Train Tracks by lobo236, from http://www.flickr.com/photos/lobo235/65337807/

SIMD Examples

add	[1,2,3,4]	[2,3,4,5]		[3.5.7.9]
and	[1,2,3,4]	[2,3,4,5]		[0,2,0,4]
lessThan	[1,2,3,4]	[4,3,2,1]		[T,T,F,F]
select	[T,F,T,F]	[1,2,3,4]	[2,3,4,5]	[1,3,3,5]
shuffle	[3,2,1,0]	[1,2,3,4]		[4,3,2,1]

Adding SIMD to JavaScript*

Proposal by John McCutchan

- Add special vector-value types: float32x4 and int32x4
  - SIMD style accessors x, y, z, w
- Also arrays of these: Float32x4Array
- Use static SIMD object to provide a set of intrinsic operations
  - add, sub, mul, div, …
  - and, or, xor, …
  - lessThan, lessThanOrEqual, equal, notEqual, greaterThanOrEqual, greaterThan
  - select, shuffle
  - …

SIMD Example in JavaScript*

```
// assume a and b are instances
// of Float32Array

var x = new Float32Array(a.length)
for (var i = 0; i < x.length; ++i) {
    x[i] = a[i] + b[i];
}
```

```
// assume a and b are instances
// of Float32x4Array

var x = new Float32x4Array(a.length)
for (var i = 0; i < x.length; ++i) {
    x[i] = SIMD.add(a[i], b[i]);
}
```

SIMD Example in JavaScript* (2)

```
// assume a and b are instances
// of Float32Array

var x = new Float32Array(a.length)
for (var i = 0; i < x.length; ++i) {
    if (a[i] > 10)
        x[i] = b[i] + b[i];
    else
        x[i] = a[i] + b[i];
}
```

```
// assume a and b are instances
// of Float32x4Array

var x = new Float32x4Array(a.length)
for (var i = 0; i < x.length; ++i) {
    var x_t = SIMD.add(b[i], b[i]);
    var x_e = SIMD.add(a[i], b[i]);
    var p = SIMD.greaterThan(a[i], 10);
    x[i] = SIMD.select(p, x_t, x_e)
}
```

More On SIMD in JavaScript*

Proposal document at ECMA

- http://wiki.ecmascript.org/doku.php?id=strawman:simd_number

Implementation effort from Intel and Mozilla*

- https://bugzilla.mozilla.org/show_bug.cgi?id=894105

Parallel JavaScript* (formerly River Trail)

High-level data-parallel programming API

- Operates on n-dimensional arrays
- Perform the same operation on each element (mostly)
- Designed for multi-core CPUs and programmable GPUs

Springtime at New River Trail by vastateparkstaff, from http://www.flickr.com/photos/vastateparksstaff/8770597938/

Adding Parallelism to JavaScript*

Proposal by Intel Labs and Mozilla*

- Extends standard JavaScript array objects and upcoming typed objects (formerly binary data)
- Provides new parallel methods
  - buildPar (factory)
  - mapPar
  - reducePar
  - scanPar
  - scatterPar
  - filterPar

Example: Increment

```
var a = [1,2,3,4,5,6];

a.mapPar(function (v) { return v+1; });

a.map(function (v) { return v+1; });
```

A New Concept: Temporal Immutability

Temporal Immutability: The global heap is immutable **during** parallel execution

Example: Increment (2)

```
var a = [1,2,3,4,5,6];

a.mapPar (function (v,i) { ++a[i]; });

a.map(function (v,i) { ++a[i]; });
```

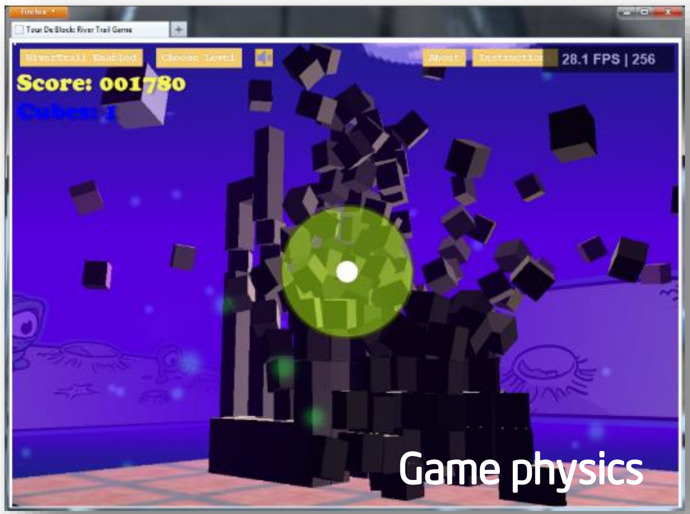
Example: Sum

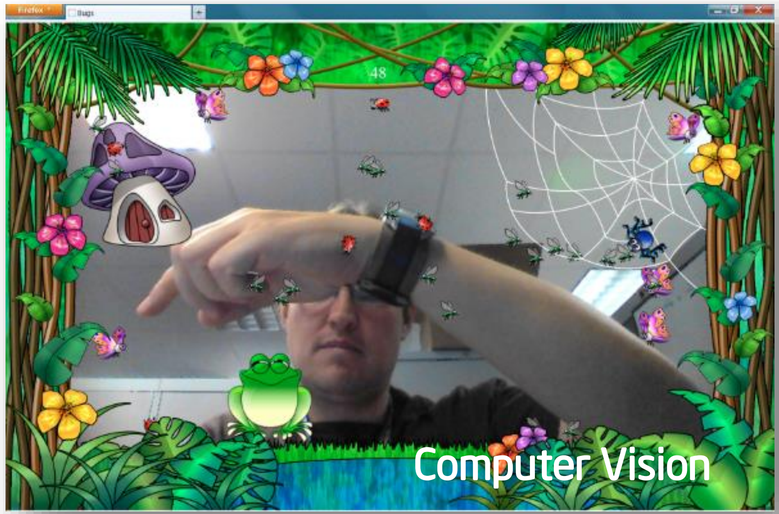
```
var a = [1,2,3,4,5,6];

a.reducePar(function (a,b) { return a+b; });

a.reduce(
    function (prev, curr) { return prev+curr; });
```

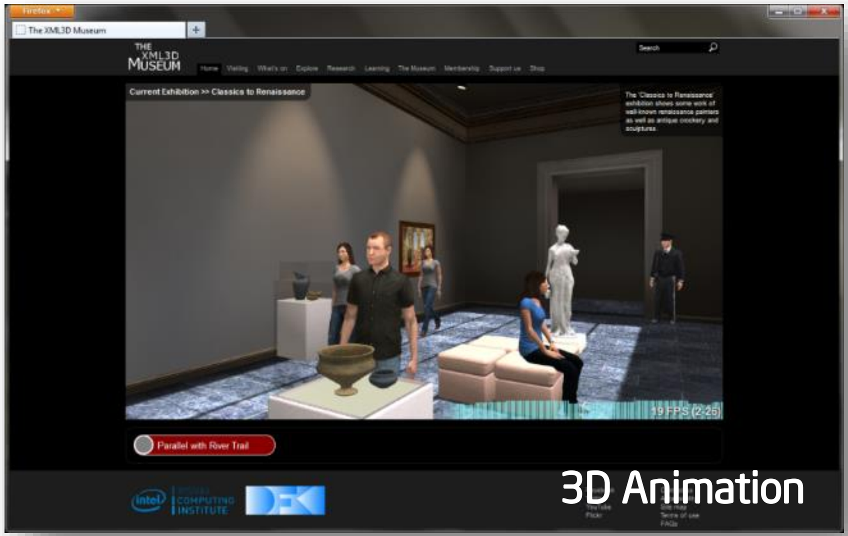
Parallel JavaScript* + Typed Objects: Grayscale

```
var Pixel = new ArrayType( uint8, 4);
var Image = new ArrayType( ArrayType( Pixel, 480), 320);

var I = getImage(); // returns value of type Image

I.mapPar(2, function (v) {
    var lum = v[0] * 0.21 + v[1]*0.72 + v[2] * 0.07;
    return new Pixel([lum, lum, lum, v[3]]);
}
```

Some Sample Applications






The XML3D Museum
THE XML3D MUSEUM
Home Visiting What's on Explore Research Learning The Museum Membership Support us Shop
Current Exhibition >> Classics to Renaissance
19 FPS (2.25)
Parallel with River Trail
3D Animation

The 'Classics to Renaissance' exhibition shows some work of well-known renaissance painters as well as antique crockery and sculptures.

More On Parallel JavaScript*

Firefox* implementation in progress, available in the latest Nightly

- http://nightly.mozilla.org

ECMA proposal available for comments

- http://wiki.ecmascript.org/doku.php?id=strawman:data_parallelism
- http://wiki.ecmascript.org/doku.php?id=harmony:typed_objects

Conclusion

Concurrency is a given in web development

- Be aware of event semantics
- Use Promises to encode complex asynchronous flows
- Use Web Workers for long running tasks

Parallel Computing offers additional speedup

- Web Workers can enable task parallelism
- SIMD extensions are on their way for small scale parallel computing
- Parallel JavaScript* for larger scale parallel computing

Image Credits

- *One Man Band* by Andrew Malone, from http://www.flickr.com/photos/andrewmalone/5163238038/
- *1967 Buick Assembly Line* by Alden Jewell, from http://www.flickr.com/photos/autohistorian/7599444736
- *Luxury Gold Ring Designs* by epSos.de, from http://www.flickr.com/photos/epsos/8904048841/
- *Spider and Web* by Dwight Sipler, from http://www.flickr.com/photos/photofarmer/3948508109/
- *Train Tracks* by lobo236, from http://www.flickr.com/photos/lobo235/65337807/
- *Springtime at New River Trail* by vastateparkstaff, from http://www.flickr.com/photos/vastateparksstaff/8770597938/