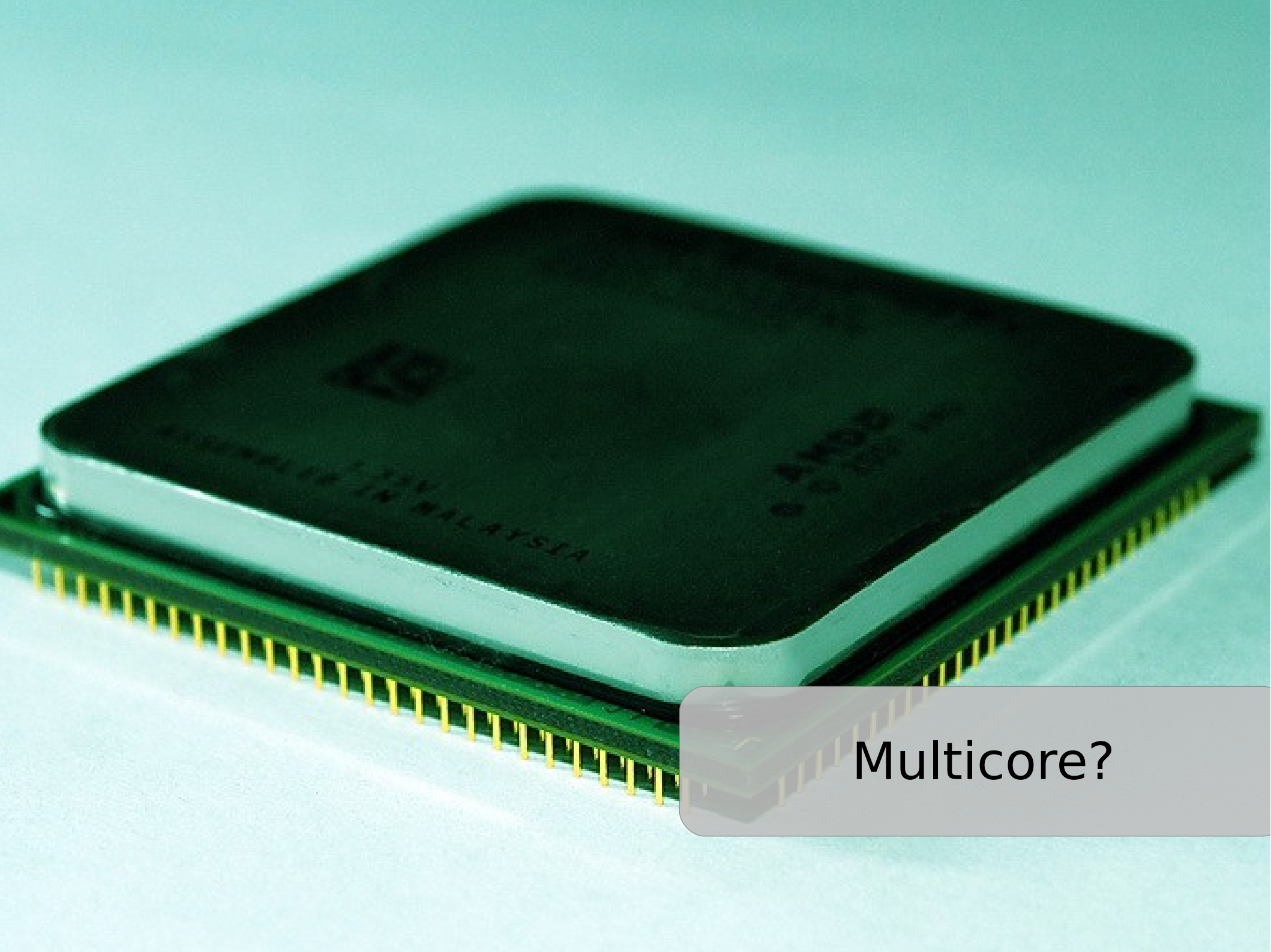

Multicore?


But isn't that *hard*?

Davidlohr Bueso
@davidlohr

Follow

A programmer had a problem. He thought to himself, "I know, I'll solve it with threads!". has Now problems. two he

Reply Retweet Favorited More

4,458 RETWEETS

1,127 FAVORITES

12:16 AM - 9 Jan 13

Disadvantages [edit]

Lock-based resource protection and thread/process synchronization have many disadvantages:

- They cause blocking, which means some threads/processes have to wait until a lock (or a whole set of locks) is released.
- Lock handling adds overhead for each access to a resource, even when the chances for collision are very rare. (However, any chance for such collisions is a race condition.)
- Locks can be vulnerable to failures and faults that are often very subtle and may be difficult to reproduce reliably. One example is the deadlock, where (at least) two threads each hold a lock that the other thread holds and will not give up until it has acquired the other lock.
- If one thread holding a lock dies, stalls/blocks or goes into any sort of infinite loop, other threads waiting for the lock may wait forever.
- Lock contention limits scalability and adds complexity.
- Balances between lock overhead and contention can be unique to given problem domains (applications) as well as sensitive to design, implementation, and even low-level system architectural changes. These balances may change over the life cycle of any given application/implementation and may entail tremendous changes to update (re-balance).
- Locks are only composable (e.g., managing multiple concurrent locks in order to atomically delete Item X from Table A and insert X into Table B) with relatively elaborate (overhead) software support and perfect adherence by applications programming to rigorous conventions.
- Priority inversion. High priority threads/processes cannot proceed, if a low priority thread/process is holding the common lock.
- Convoying. All other threads have to wait, if a thread holding a lock is descheduled due to a time-slice interrupt or page fault (*See lock convoy*)
- Hard to debug: Bugs associated with locks are time dependent. They are extremely hard to replicate.
- There must be sufficient resources - exclusively dedicated memory, real or virtual - available for the locking mechanisms to maintain their state information in response to a varying number of contemporaneous invocations, without which the mechanisms will fail, or "crash" bringing down everything depending on them and bringing down the operating region in which they reside. "Failure" is better than crashing, which means a proper locking mechanism ought to be able to return an "unable to obtain lock for <whatever> reason" status to the critical section in the application, which ought to be able to handle that situation gracefully. The logical design of an application requires these considerations from the very root of conception.

Some people[*who?*] use a concurrency control strategy that doesn't have some or all of these problems. For example, some people[*who?*] use a funnel or serializing tokens, which makes their software immune to the biggest problem—deadlocks. Other people[*who?*] avoid locks entirely—using non-blocking synchronization methods, like lock-free programming techniques and transactional memory. However, many of the above disadvantages have analogues with these alternative synchronization methods.

Any such "concurrency control strategy" would require actual lock mechanisms implemented at a more fundamental level of the operating software (the analogues mentioned above), which may only relieve the application level from the details of implementation. The "problems" remain, but are dealt with beneath the application. In fact, proper locking ultimately depends upon the CPU hardware itself providing a method of atomic instruction stream synchronization. For example, the addition or deletion of an item into a pipeline requires that all contemporaneous operations needing to add or delete other items in the pipe be suspended during the manipulation of the memory content required to add or delete the specific item. The design of an application is better when it recognizes the burdens it places upon an operating system and is capable of graciously recognizing the reporting of impossible demands.

JavaScript Web Workers

Tobias Pfeiffer
@PragTob
pragtob.wordpress.com

JavaScript Web Workers

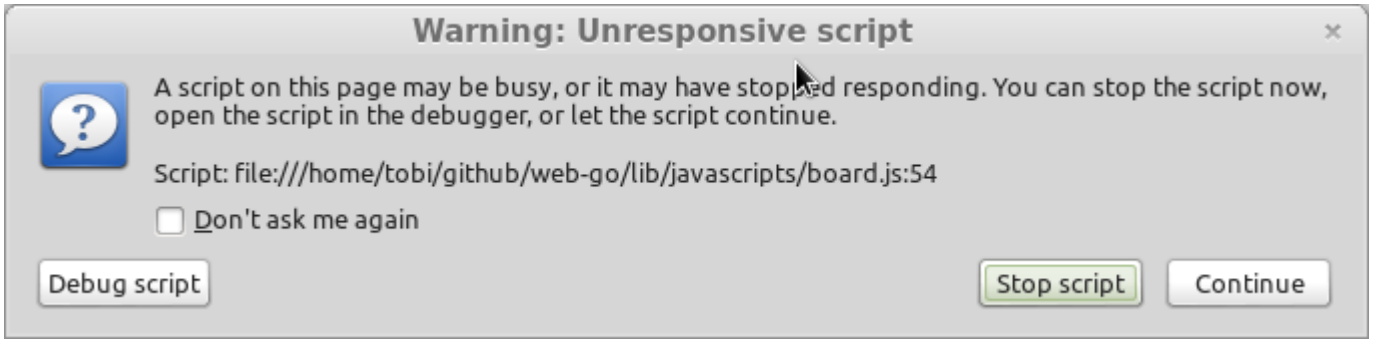
Tobias Pfeiffer*
@PragTob
pragtob.wordpress.com

*with contributions from Tobias Metzke

HTML

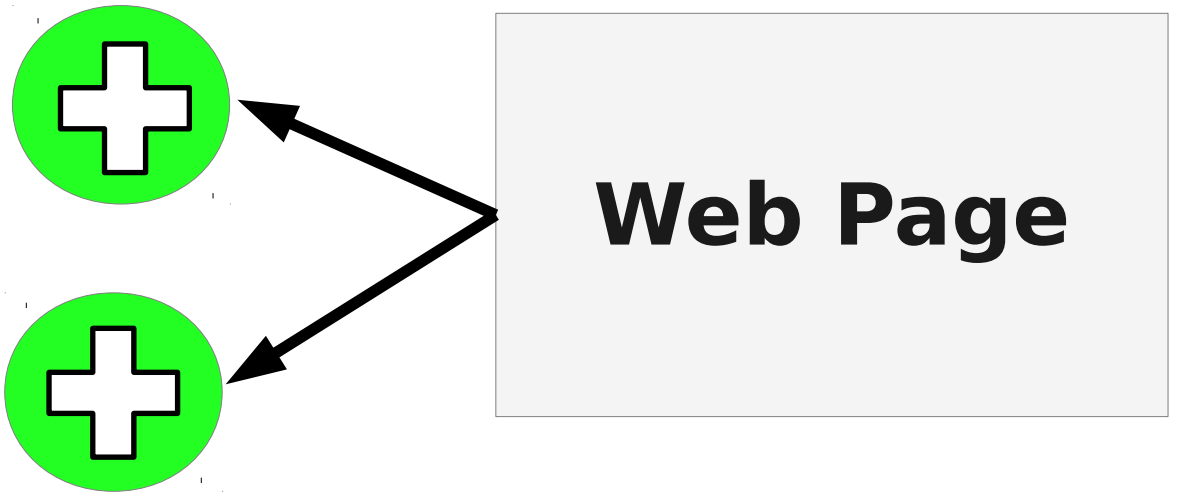

Multi-threaded

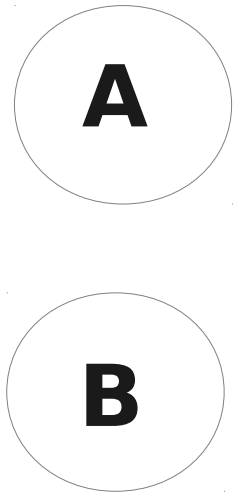

Share nothing

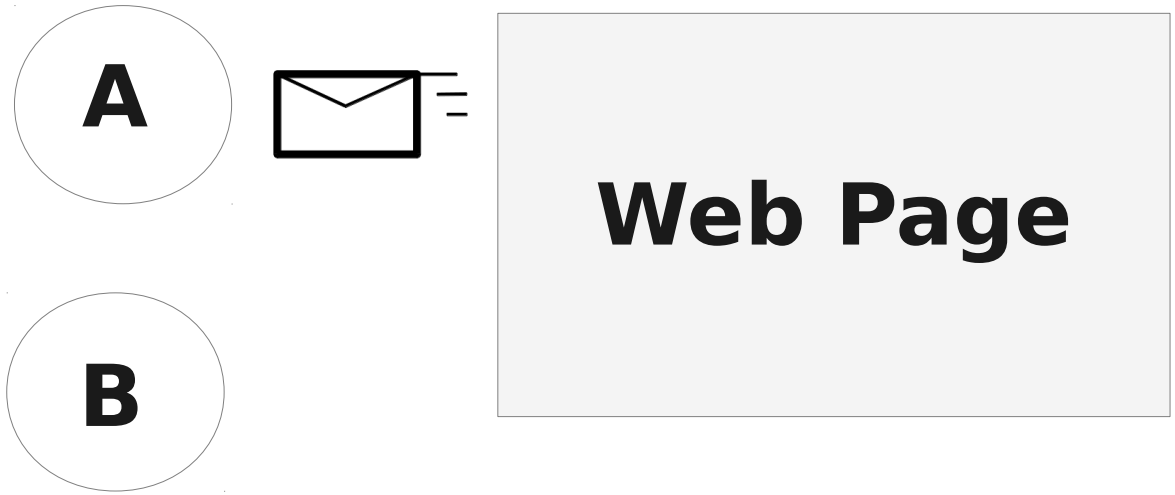


Not blocking the main thread

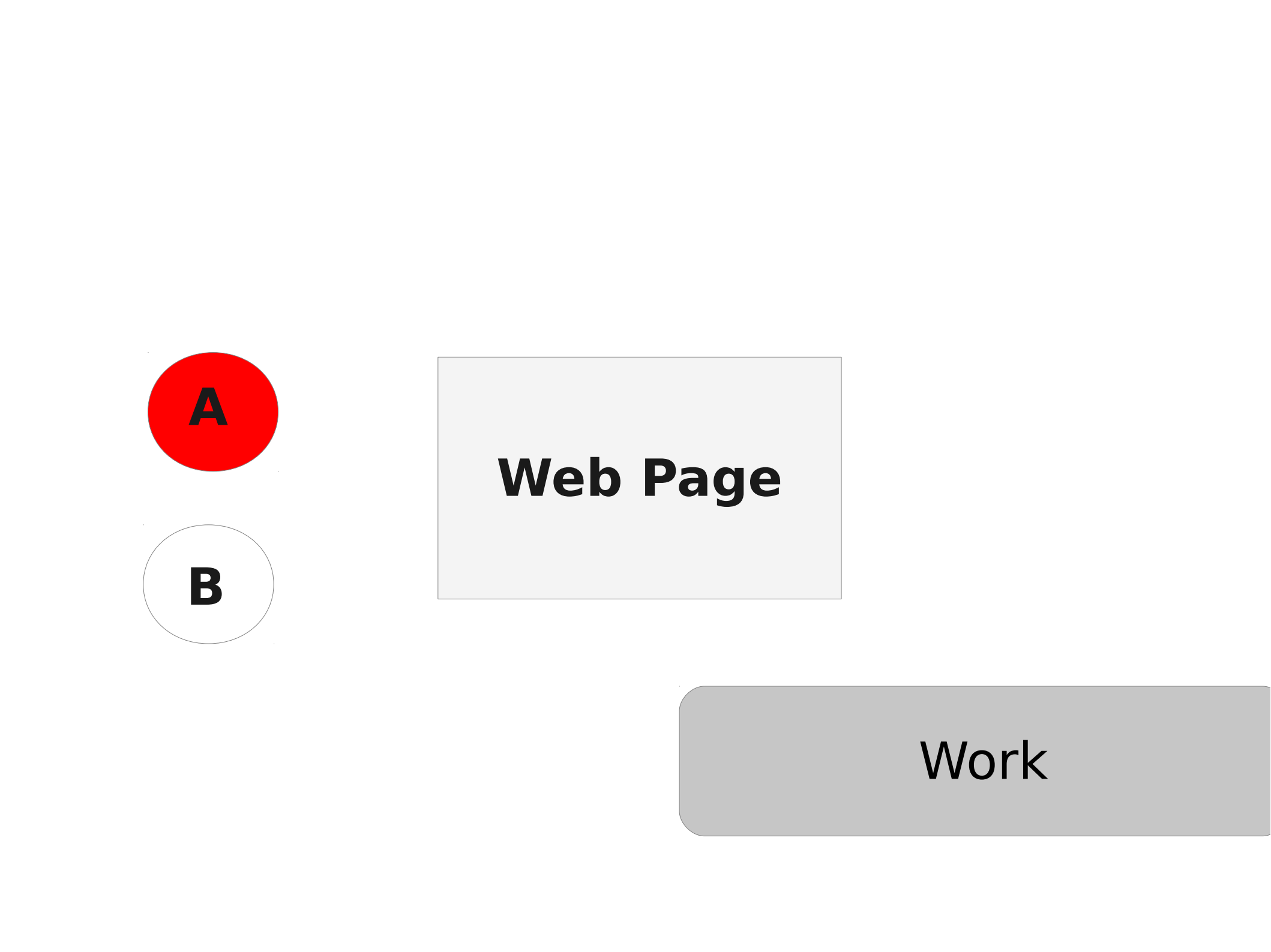
Create


Web Page

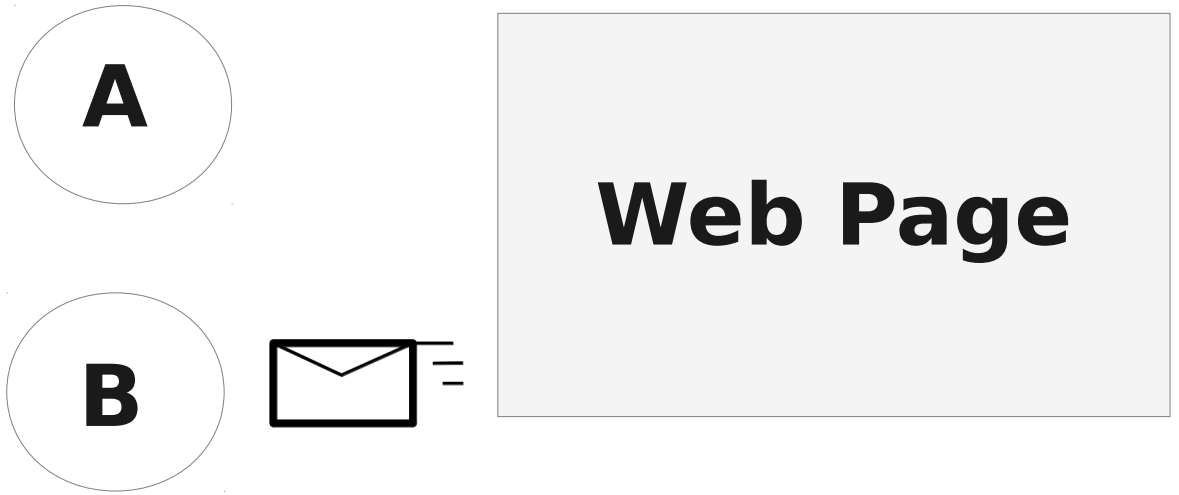

A
B
Web Page

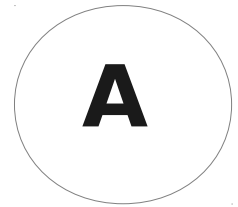


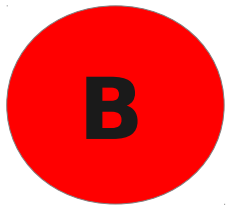
Send Message


A
B
Web Page
Work

Answer


A
B
Web Page


A


B


Web Page


A
B
Web Page


What about
thread safety?

„The Worker interface spawns real OS-level threads, and concurrency can cause interesting effects in your code if you aren't careful. However, in the case of web workers, (...)it's actually very hard to cause concurrency problems. (...) So you have to work really hard to cause problems in your code.“

Mozilla Developer Network





N
NE
E
SE
S
SW
W
NW


Limitations

- Can not access the DOM
- High setup and memory cost
- Data may not contain functions or cycles
- Debugging


Getting to work

Starting a web worker

```
var worker = new Worker('worker_script.js');
```

Starting a web worker

```
var worker = new Worker('worker/counter.js');
```

Listening to the worker

```
var worker = new Worker('worker/counter.js');
worker.addEventListener('message', function(event){
  // do stuff
});
```

Listening to the worker

```
var worker = new Worker('worker/counter.js');
worker.addEventListener('message', function(event){
  $('#result').html(event.data);
});
```

Sending the worker messages

```
var worker = new Worker('worker/counter.js');
worker.addEventListener('message', function(event){
  $('#result').html(event.data);
});
worker.postMessage(data);
```

Sending the worker messages

```
var worker = new Worker('worker/counter.js');
worker.addEventListener('message', function(event){
  $('#result').html(event.data);
});
worker.postMessage({});
```

The Worker

```
var counter = 0;

self.onmessage = function(message) {
  counter++;
  self.postMessage(counter);
}
```

The Worker

```
var counter = 0;

self.onmessage = function(message) {
  counter++;
  self.postMessage(counter);
}
```

The Worker

```
var counter = 0;

self.onmessage = function(message) {
  counter++;
  self.postMessage(counter);
}
```

A Prime Number worker

```
findPrime = function(n) {
  for (var i = 2; i <= Math.sqrt(n); i += 1) {
    if (n % i == 0) {
     return false;
    }
  }
  return n;
}

self.onmessage = function(event) {
  var num = parseInt(event.data);
  self.postMessage(findPrime(num);
}
```

A Prime Number worker

```
findPrime = function(n) {
  for (var i = 2; i <= Math.sqrt(n); i += 1) {
    if (n % i == 0) {
     return false;
    }
  }
  return n;
}
```

```
self.onmessage = function(event) {
  var num = parseInt(event.data);
  self.postMessage(findPrime(num);
}
```

Importing Scripts in a worker

```
importScripts('script_path.js');
```

Background I/O

```
importScripts('io.js');
var timer;
var symbol;
function update() {
  postMessage(symbol + ' ' + get('stock.cgi?' +
symbol));
  timer = setTimeout(update, 10000);
}
onmessage = function (event) {
  if (timer)
    clearTimeout(timer);
  symbol = event.data;
  update();
};
```

Background I/O

```
importScripts('io.js');
var timer;
var symbol;
function update() {
  postMessage(symbol + ' ' + get('stock.cgi?' +
symbol));
  timer = setTimeout(update, 10000);
}
onmessage = function (event) {
  if (timer)
    clearTimeout(timer);
  symbol = event.data;
  update();
};
```

Background I/O

```
importScripts('io.js');
var timer;
var symbol;
function update() {
  postMessage(symbol + ' ' + get('stock.cgi?' +
symbol));
  timer = setTimeout(update, 10000);
}
onmessage = function (event) {
  if (timer)
    clearTimeout(timer);
  symbol = event.data;
  update();
};
```

Background I/O

```
importScripts('io.js');
var timer;
var symbol;
function update() {
  postMessage(symbol + ' ' + get('stock.cgi?' +
symbol));
  timer = setTimeout(update, 10000);
}
onmessage = function (event) {
  if (timer)
    clearTimeout(timer);
  symbol = event.data;
  update();
};
```

Listening for errors

```
worker.addEventListener('error', function(event){
  // happy debugging
});
```

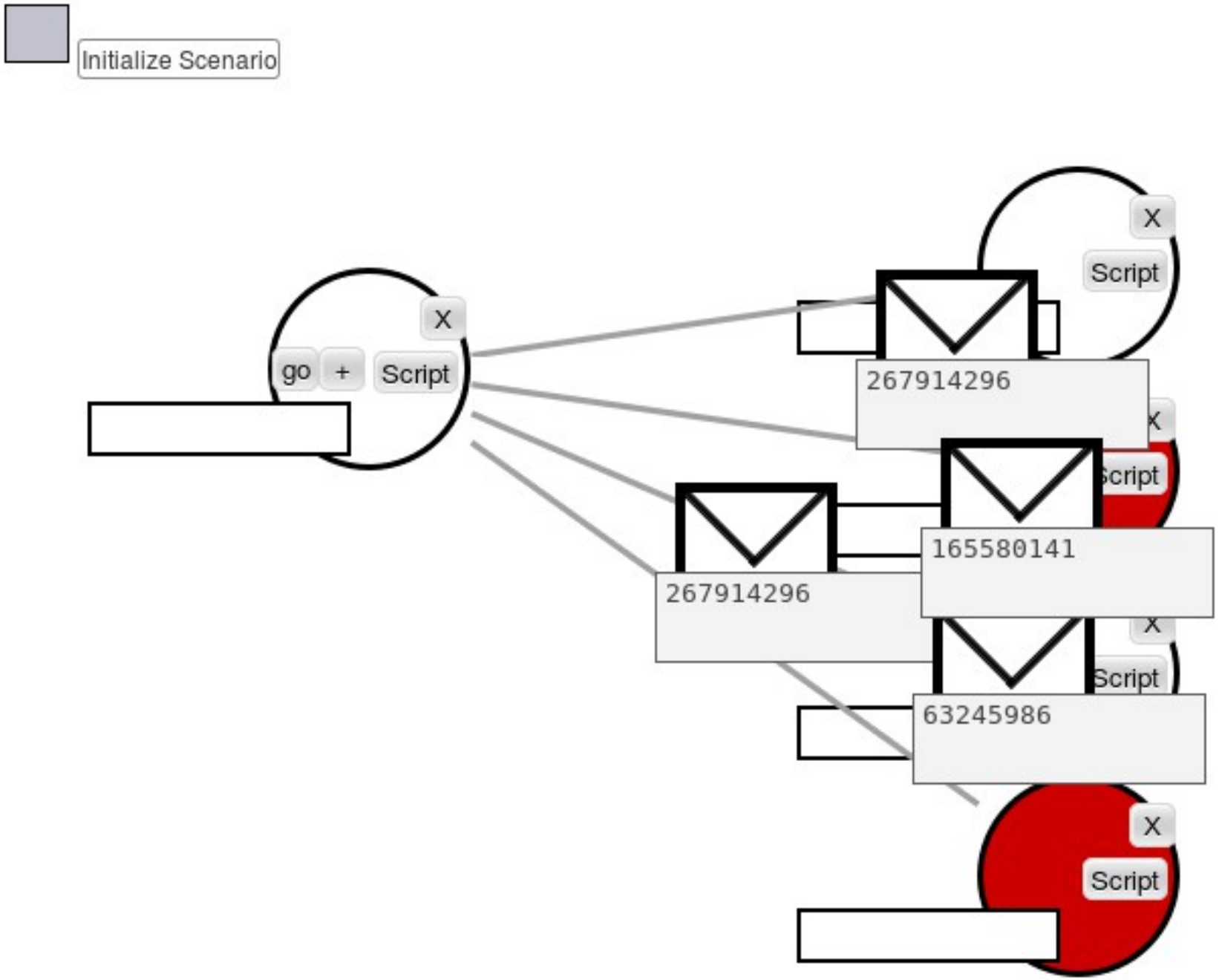
Use Cases

- Expensive non UI computations
- Handling big data
- Background I/O
- Ray Tracers
- AI
- ...
- https://developer.mozilla.org/en-US/demos/tag/tech:webworkers/


Actors?

Actors

- Run in their own thread
- Do not share state
- Communicate via asynchronous messages


Initialize Scenario
X
go
+
Script
X
Script
267914296
X
Script
165580141
267914296
X
Script
63245986
X
Script

Web Workers

- Using modern multi core computers on the client side
- Allow you to do the heavy lifting on the client side
- Relatively easy to use

Resources

- Web Hypertext Application Technology Working Group. HTML Specification - Web Workers.Website, 2013. http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html
- Mozilla Developer Network. Using Web Workers. Website, 2013. https://developer.mozilla.org/en-US/docs/DOM/Using_web_workers
- MDN: Web Worker Demos https://developer.mozilla.org/en-US/demos/tag/tech:webworkers
- Visualization from the end (don't use actor and actor2, use the others) http://www.lively-kernel.org/repository/webwerkstatt/users/tmetzke/

Photo Credit

- Creative Commonts Attribution:
  - http://www.w3.org/html/logo/downloads/HTML5_Logo.svg
  - http://www.flickr.com/photos/klearchos/5580186619/
  - http://www.flickr.com/photos/tedmurphy/8482500187/
- Creative Commons Attribution no derivative Works
  - http://www.flickr.com/photos/49333775@N00/2383975585/
  - http://www.flickr.com/photos/adplayers/5758743751/
- Creative Commons Attribution Share Alike
  - http://www.flickr.com/photos/amatuer_44060/2831112854/
- Logos
  - https://assets.mozillalabs.com/Brands-Logos/Firefox/logo-only/firefox_logo-only_RGB.png
  - http://de.wikipedia.org/wiki/Datei:Internet_Explorer_9.svg
  - https://en.wikipedia.org/wiki/File:Apple_Safari.png
  - http://commons.wikimedia.org/wiki/File:Opera_O.svg
  -
- Wikimedia Commons
  - http://en.wikipedia.org/wiki/File:Chromium_11_Logo.svg
  - http://en.wikipedia.org/wiki/File:Athlon64x2-6400plus.jpg

Photo Credit 2

- Creative Commons Attribution-No Derivs – No Commercial
  - http://www.flickr.com/photos/cheesy42/8414666632/