An introduction to the strange APIs of IndexedDB

Jiayi Hu

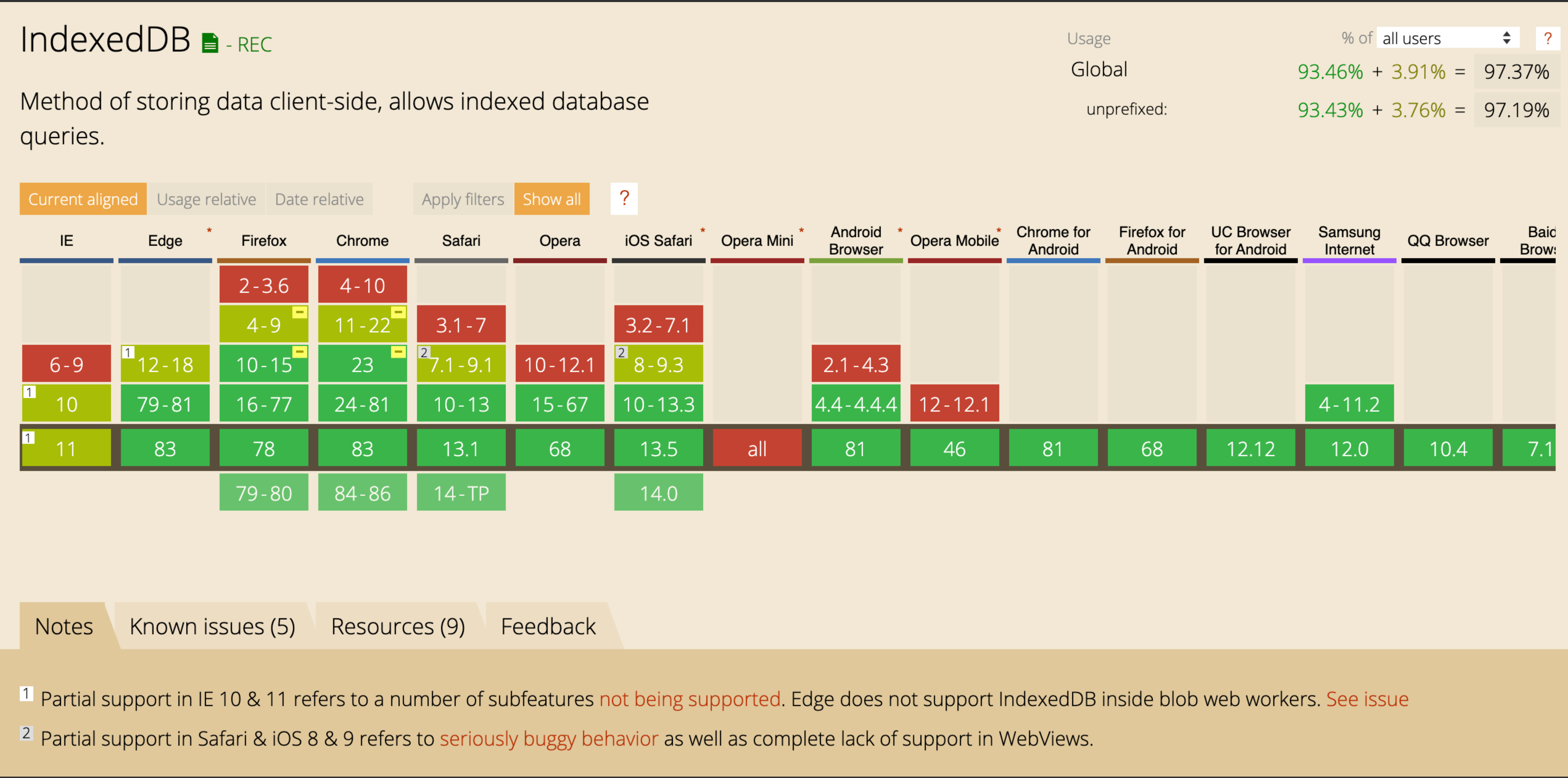
PWAConf

2 luglio 2020

aspitalia.com

IndexedDB

- In-browser NoSQL database
- Built-in async API
  - Results are dispatched as DOM events
- A *key-value* object-oriented database
- Transactional (ACI~~D~~)



IndexedDB - REC

Usage % of all users ?

Global 93.46% + 3.91% = 97.37%

unprefixed: 93.43% + 3.76% = 97.19%

Method of storing data client-side, allows indexed database queries.

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baid Brows
		2-3.6	4-10												
		4-9	11-22	3.1-7		3.2-7.1									
6-9	[1] 12-18	10-15	23	[2] 7.1-9.1	10-12.1	[2] 8-9.3		2.1-4.3							
[1] 10	79-81	16-77	24-81	10-13	15-67	10-13.3		4.4-4.4.4	12-12.1				4-11.2		
[1] 11	83	78	83	13.1	68	13.5	all	81	46	81	68	12.12	12.0	10.4	7.1
		79-80	84-86	14-TP		14.0									

[1] Partial support in IE 10 & 11 refers to a number of subfeatures not being supported. Edge does not support IndexedDB inside blob web workers. See issue

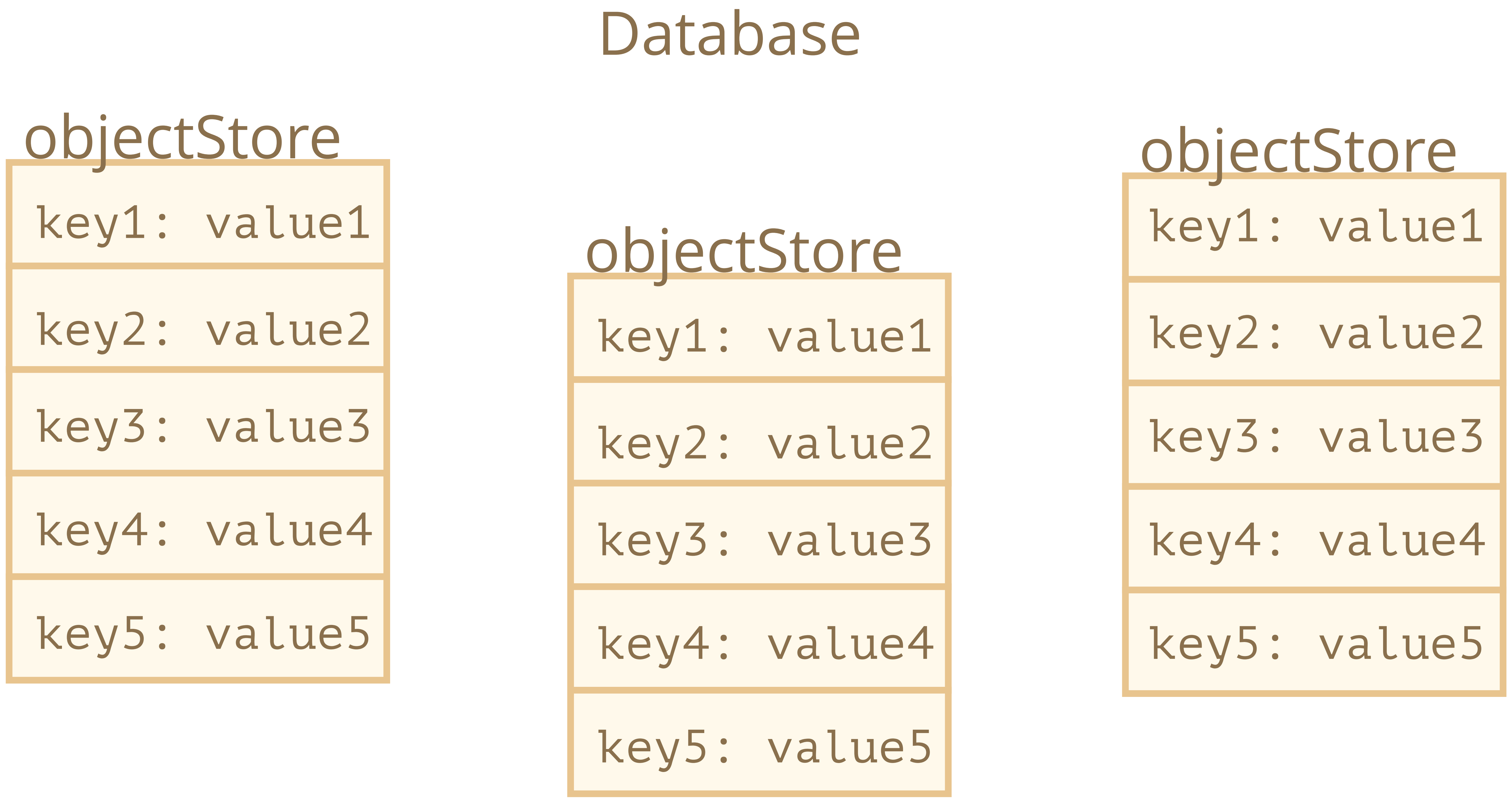
[2] Partial support in Safari & iOS 8 & 9 refers to seriously buggy behavior as well as complete lack of support in WebViews.

Support

```
window.indexedDB = window.indexedDB
   || window.mozIndexedDB
   || window.webkitIndexedDB
   || window.msIndexedDB;
```

Or

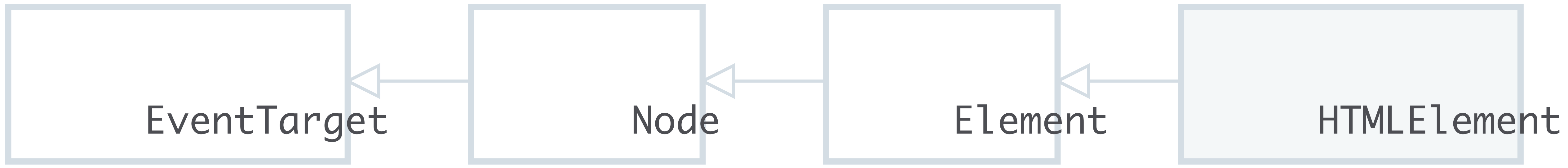
```
if (!window.indexedDB) {
    console.log("IndexedDB not fully supported");
}
```


Database
objectStore
key1: value1
key2: value2
key3: value3
key4: value4
key5: value5
objectStore
key1: value1
key2: value2
key3: value3
key4: value4
key5: value5
objectStore
key1: value1
key2: value2
key3: value3
key4: value4
key5: value5

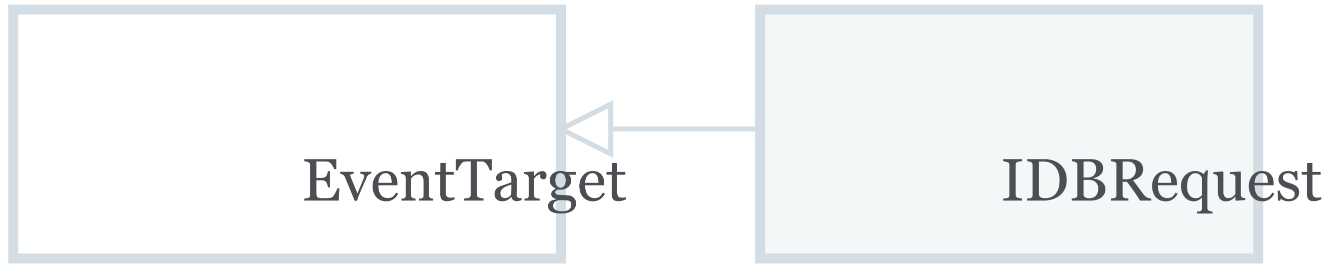
Async API as DOM Events

```
document.body.addEventListener('click', (event) => {
  console.log(event.target);
})
```

EventTarget



EventTarget



IDBRequest

```
interface IDBRequest<T = any> extends EventTarget {
    onerror: (ev: Event) => any) | null;
    onsuccess: (ev: Event) => any) | null;

    readyState: "done" | "pending";
    result: T;
    source: IDBObjectStore | IDBIndex | IDBCursor | null;
    transaction: IDBTransaction | null;

    addEventListener(type: string, listener: Listener): void;
    removeEventListener(type: string, listener: Listener): void;
}
```

Async API as DOM Events

```
const request = indexedDB.open('skypeweb', 1);

request.addEventListener('error', event => {
  const error = event.target.error;
  console.error(error)
});

request.addEventListener('success', (event) => {
  const db = event.target.result;

  console.log(db);
});
```

IndexedDB versioning - New version

```
request.addEventListener('upgradeneeded', () => {
  const db = request.result;
  if (!db.objectStoreNames.contains('new-messages')) {
    db.createObjectStore('new-messages', { keyPath: 'body.messageId' });
    db.createObjectStore('old-messages', { autoIncrement: true });
  }
});

request.addEventListener('blocked', () => {
  reject('Request to open IDB was blocked');
});
```

IndexedDB versioning - Old version

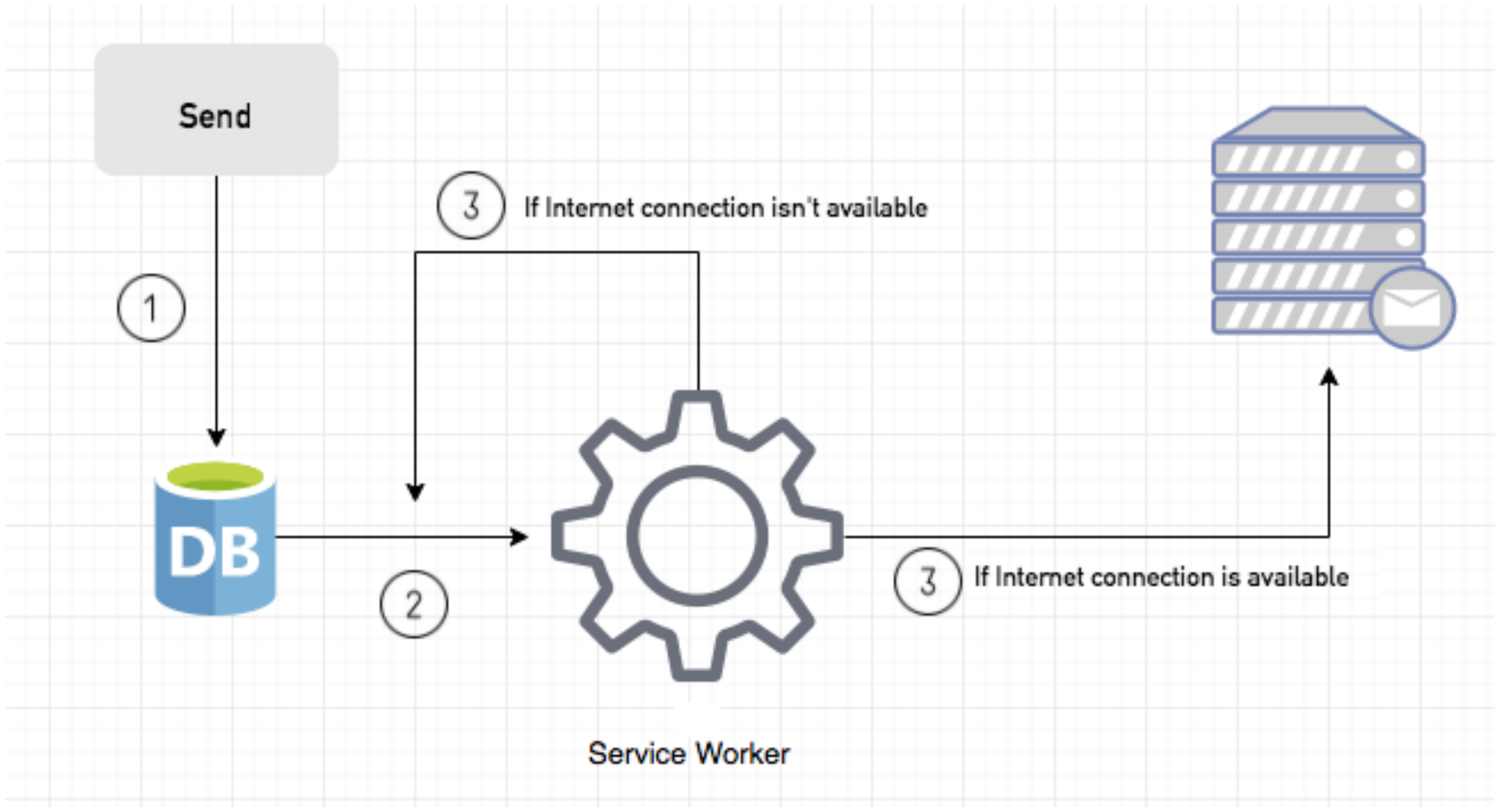
```
request.addEventListener('success', () => {
  const db = request.result;

  db.addEventListener('versionchange', () => {
    db.close();
    window.location.reload();
  });
});
```

```
const request = indexedDB.open("store", 2);

request.onupgradeneeded = function() {
  // Existing db version
  const db = request.result;

  switch(db.version) {
    case 0:
      ...
    case 1:
      ...
  }
};
```

```
export const getIDB = (name, version) => {
  return new Promise((resolve, reject) => {
    const request = indexedDB.open(name, version);

    request.addEventListener('error', event => {
      const error = event.target.error;
      reject(error);
    });

    request.addEventListener('success', () => {
      const db = request.result;

      resolve(db);
    });

    request.addEventListener('upgradeneeded', () => {
      const db = request.result;
      if (!db.objectStoreNames.contains('new-messages')) {
        db.createObjectStore('new-messages', { keyPath: 'body.authorId' });
      }
    });
  });
};
```

```
getIDB('skypeweb', 1)
  .then(db => {
    console.log(db)
  })
  .catch(error => console.error(error))
```

BackgroundSync example [1]


Send
1
3
If Internet connection isn't available
DB
2
Service Worker
3
If Internet connection is available

[1] https://davidwalsh.name/background-sync

BackgroundSync example

```
// service-worker.js
self.addEventListener('sync', event => {
  switch (event.tag) {
    case 'new-message': {
      console.log('New message to send')
      break;
    }
    default:
      break;
  }
});
```

Transactions

A transaction is a unit of work that you want to treat as "a whole." It has to either happen in full or not at all.

— **Atomic**: it must either complete in its entirety
— **Consistent**: it conforms to integrity constraints
— **Isolated**: it supports concurrency
— ~~**Durable**~~: written to persistent storage

Retrieve messages

```
const operation = getIDB('skypeweb', 1)
  .then(db => {
    const transaction = db.transaction('new-messages')
    const messagesStore = transaction.objectStore('new-messages')
    const request = messagesStore.getAll();

    request.addEventListener('success', () => {
      console.log(request.result);
    })
  })
```

Retrieve messages

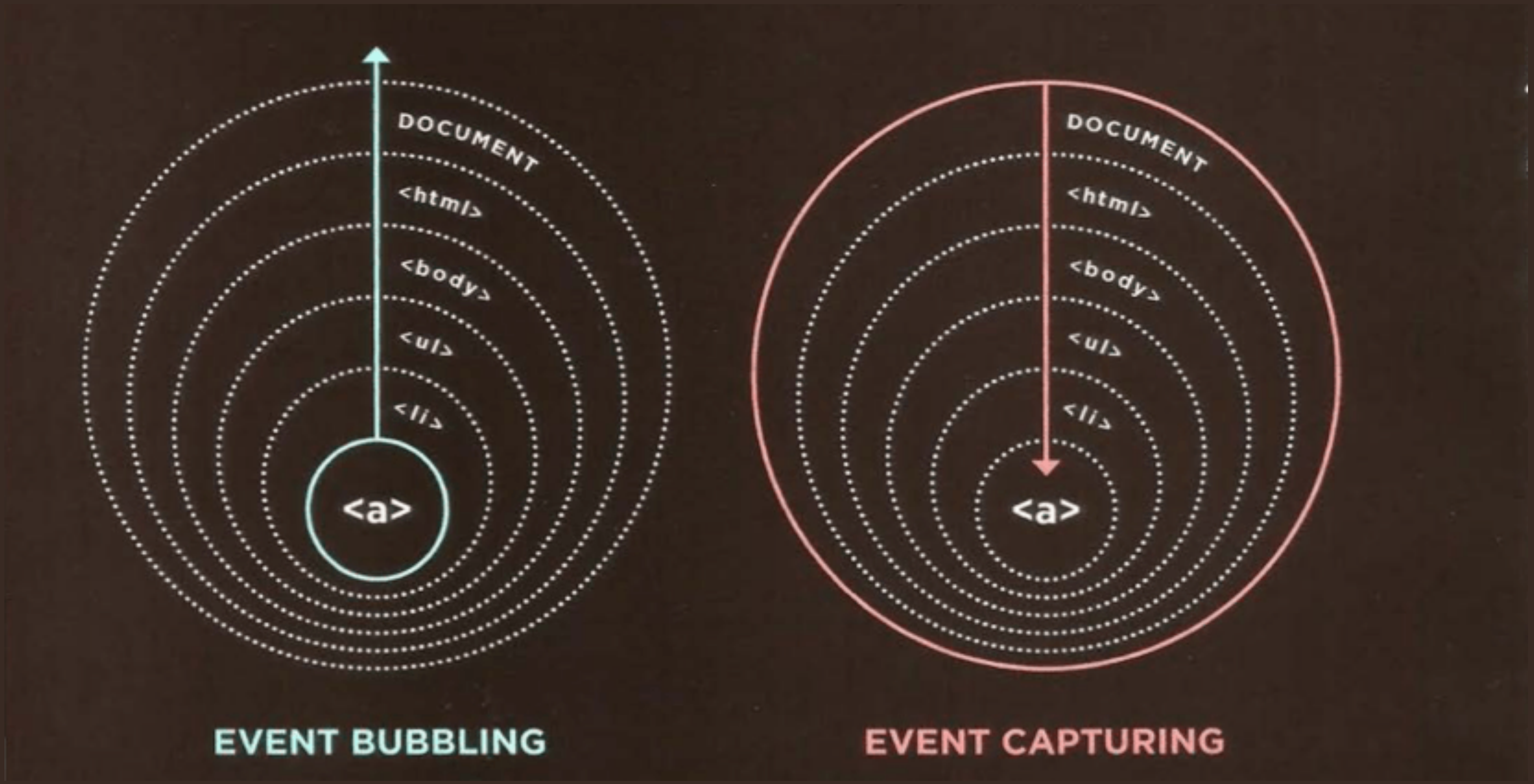
```
const operation = getIDB('skypeweb', 1)
  .then(db => {
    const request = db
      .transaction('new-messages')
      .objectStore('new-messages')
      .getAll();

    request.addEventListener('success', () => {
      console.log(request.result);
    })
  })
```

Retrieve messages

```
const operation = getIDB('skypeweb', 1)
  .then(db => {
    const request = db
      .transaction('new-messages')
      .objectStore('new-messages')
      .getAll();

    request.addEventListener('success', () => {
      console.log(request.result);
    })
  })
```

```
function promisifyRequest(request) {
  return new Promise((resolve, reject) => {
    request.onsuccess = function() {
      resolve(request.result);
    };

    request.onerror = function() {
      reject(request.error);
    };
  });
}
```

Event bubbling



```
const operation = getIDB('skypeweb', 1)
  .then(db => {
    const request = db
      .transaction('new-messages')
      .objectStore('new-messages')
      .getAll();

    return promisifyRequest(request)
      .then(messages => console.log(messages))
  })
```

Save message to be sent

```
export function addMessage(threadId, body) {
  return fetch(`threads/${threadId}/messages`, {
    method: 'POST',
    body: JSON.stringify(body),
  }).catch(() => {
    if (window.indexedDB && window.SyncManager) {
      getDb('skypeweb', 1)
        .then((db) => {
          const request = db
            .transaction('new-messages', 'readwrite')
            .objectStore('new-messages')
            .put({ body, threadId });

          return promisifyRequest(request);
        })
        .then(() => {
          return navigator.serviceWorker.ready.then((registration) => {
            return registration.sync.register('new-message');
          });
        });
    }
  });
}
```

```
self.addEventListener('sync', event => {
  switch (event.tag) {
    case 'new-message': {
      const operation = getIDB('skypeweb', 1)
        .then(db => {
          const request = db
            .transaction('new-messages')
            .objectStore('new-messages')
            .getAll();

          return promisifyRequest(request)
            .then(messages => console.log(messages))
        })
      break;
    }
    default:
      break;
  }
});
```

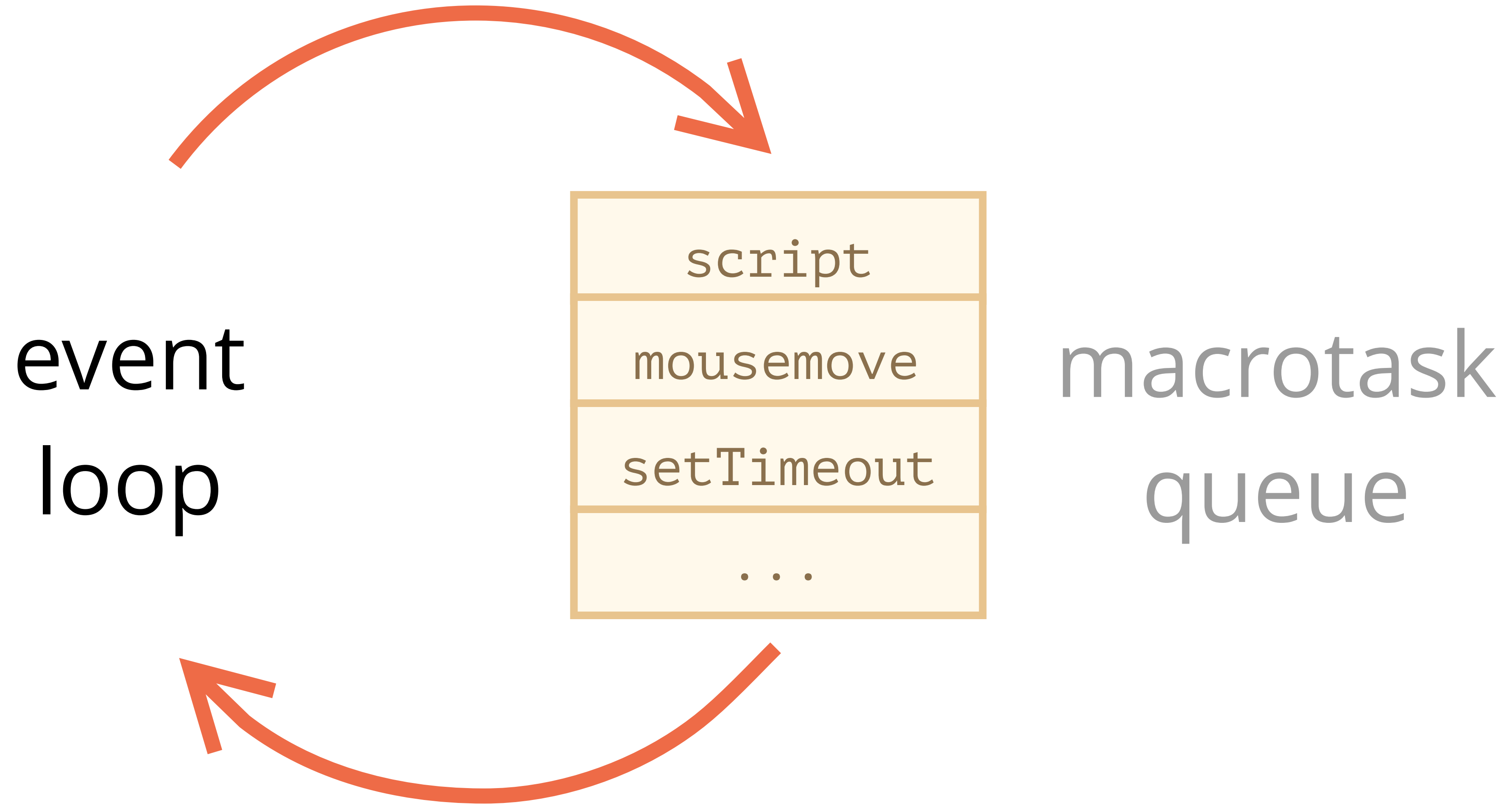
```
const operation = getIDB('skypeweb', 1)
  .then(db => {
    const request = db
      .transaction('new-messages')
      .objectStore('new-messages')
      .getAll();

    return promisifyRequest(request).then(messages => {
      const fetches = messages.map(message => {
        const { threadId, body } = message;

        return fetch('...', { body: JSON.stringify(body) })
      });

      return Promise.all(fetches);
    });
  })

event.waitUntil(operation);
```

```
const { threadId, body } = message;

return fetch('...', { body: JSON.stringify(body) })
  .then(() => {
    const request = db
      .transaction('new-messages', 'readwrite')
      .objectStore('new-messages')
      .delete(body.messageId);

    return promisifyRequest(request);
  });
```

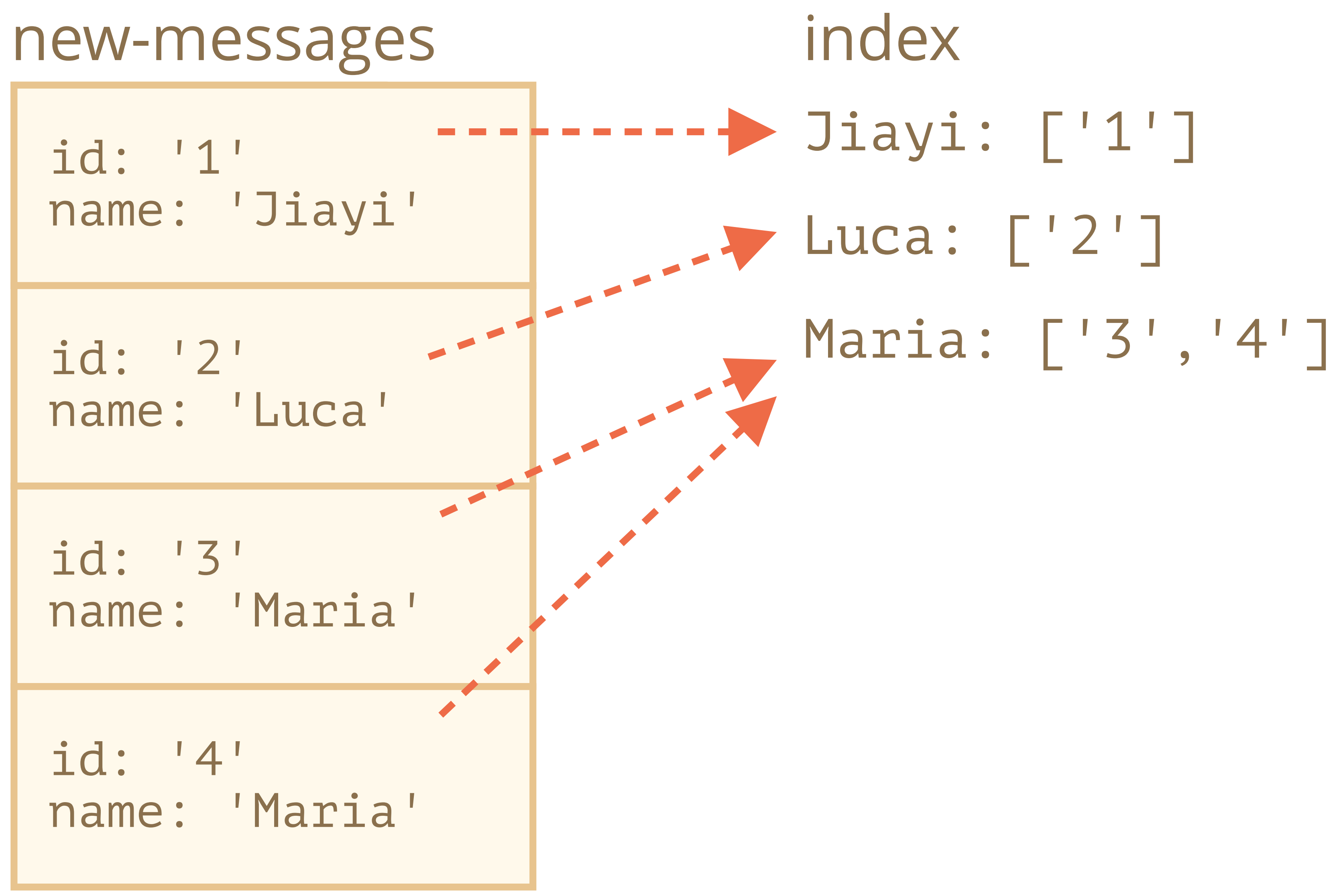
Transactions should be short-lived, for performance reasons.

```
const newMessages = db
  .transaction('new-messages', 'readwrite')
  .objectStore('new-messages')
const request = newMessages
  .add({ body, bulletinId });

request.onsuccess = function() {
  fetch('/messages/').then(response => {
    const request2 = newMessages.getAll(); // <=

    request2.onerror = function() {
      console.log(request2.error.name); // TransactionInactiveError
    };
  });
};
```

Event Loop



Indexes


new-messages
id: '1'
name: 'Jiayi'
id: '2'
name: 'Luca'
id: '3'
name: 'Maria'
id: '4'
name: 'Maria'
index
Jiayi: ['1']
Luca: ['2']
Maria: ['3','4']

Indexes

```
request.addEventListener('upgradeneeded', () => {
  const db = request.result;
  if (!db.objectStoreNames.contains('new-messages')) {
    const messagesStore = db.createObjectStore('new-messages');

    messagesStore.createIndex('name', 'name', { unique: false });
  }
});
```

```
const request = db
  .transaction('new-messages')
  .objectStore('new-messages')
  .index('name')
  .get('Jiayi'); // or getAll()
```

Simpler abstractions

— jakearchibald/idb-keyval

— jakearchibald/idb

```
if (!idb)
  self.importScripts(
    'https://unpkg.com/idb@5.0.2/build/iife/index-min.js'
  );
```

Storage limits

```
navigator.storage.estimate().then(({usage, quota}) => {
  console.log(`Using ${usage} out of ${quota} bytes.`);
});
```

Thanks

Jiayi Hu

- Front-end developer
- https://github.com/jiayihu
- @jiayi_ghu

Q & A?