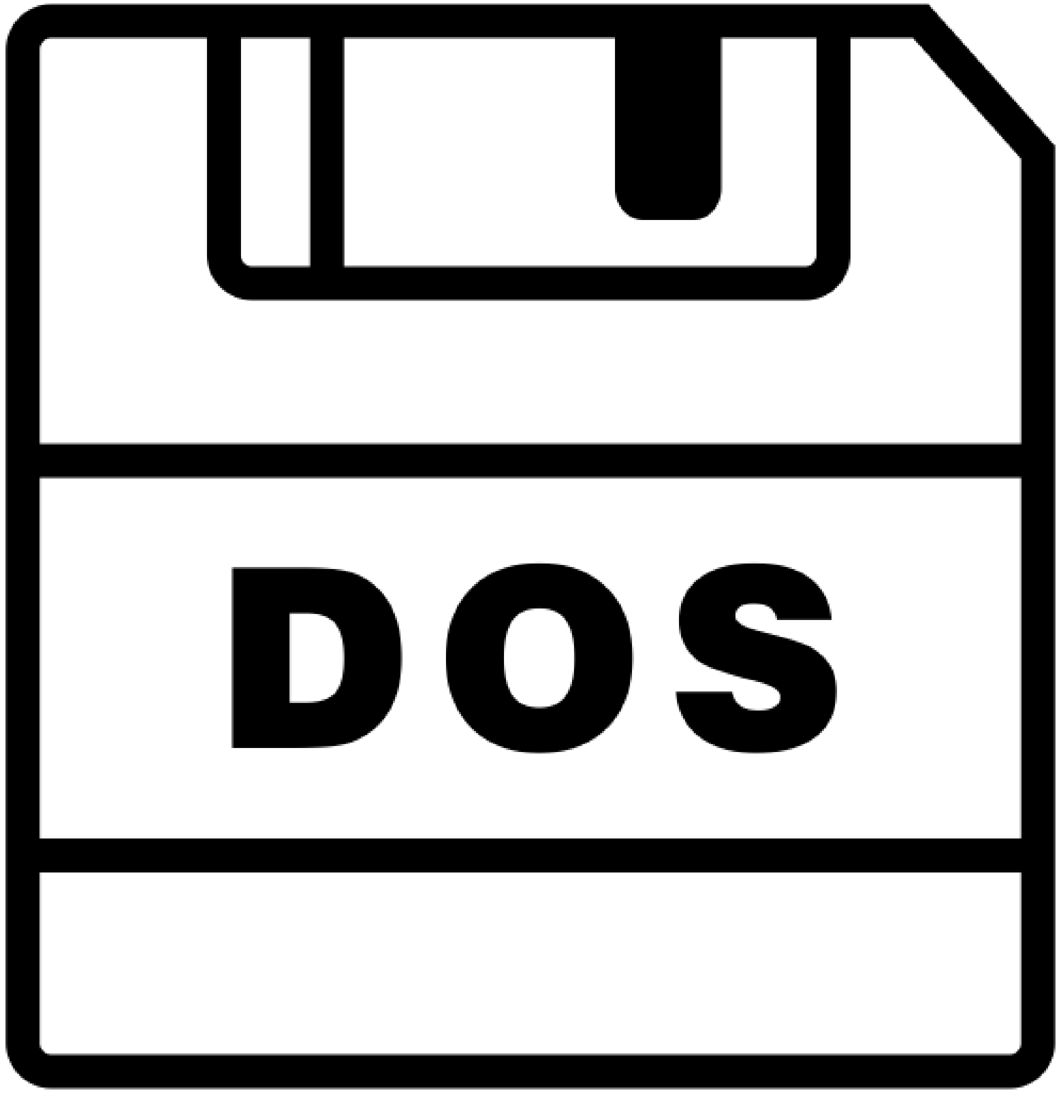
This DoS goes loop-di-loop

Preventing DoS attacks on your Node.js application

Allon Mureinik
Senior Manager, Seeker (IAST) Agents R&D
Black Duck

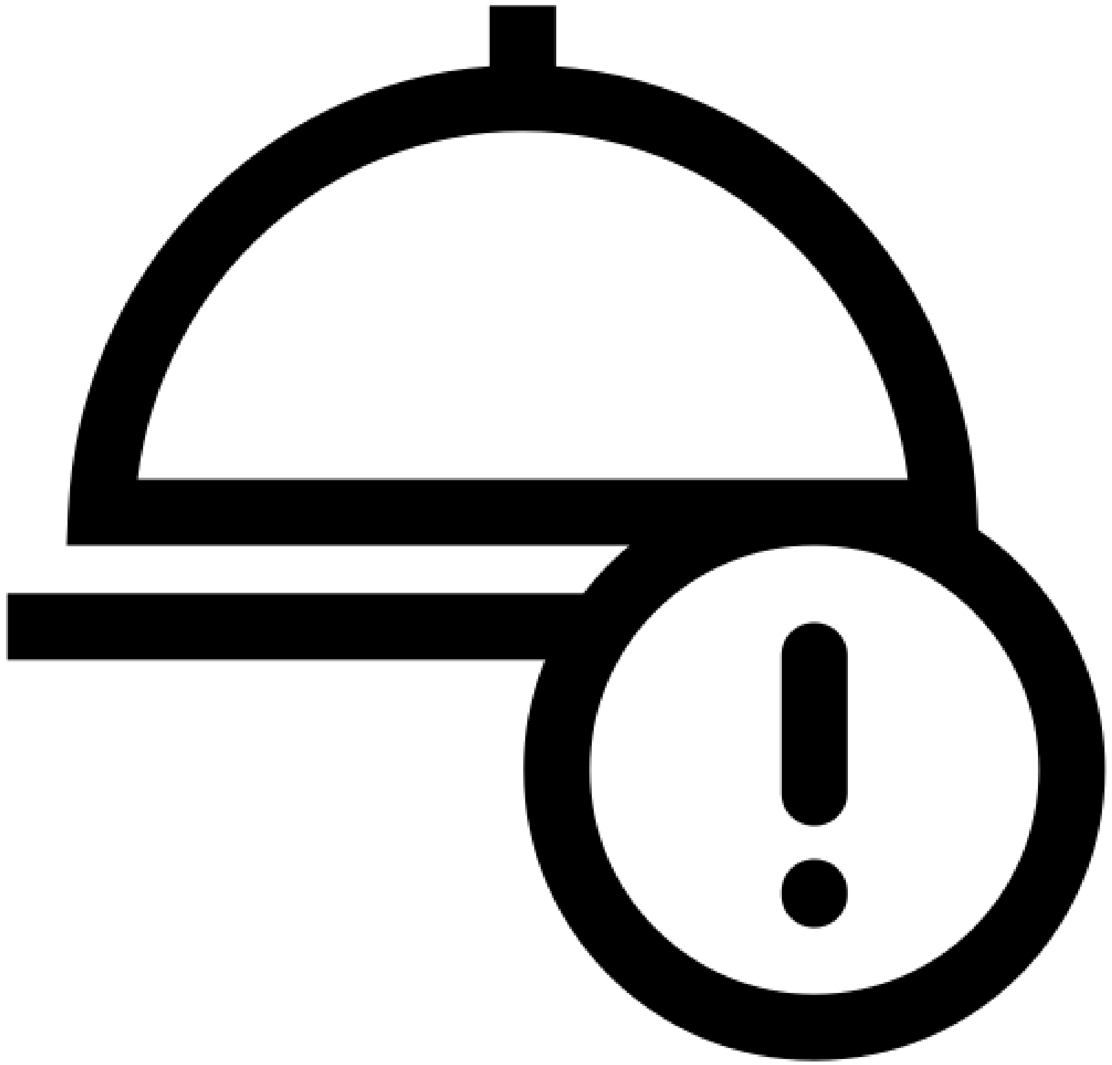
Conf42, 31/10/2024

No, not that kind of DOS



https://thenounproject.com/icon/save-dos-818218/

This kind of DoS



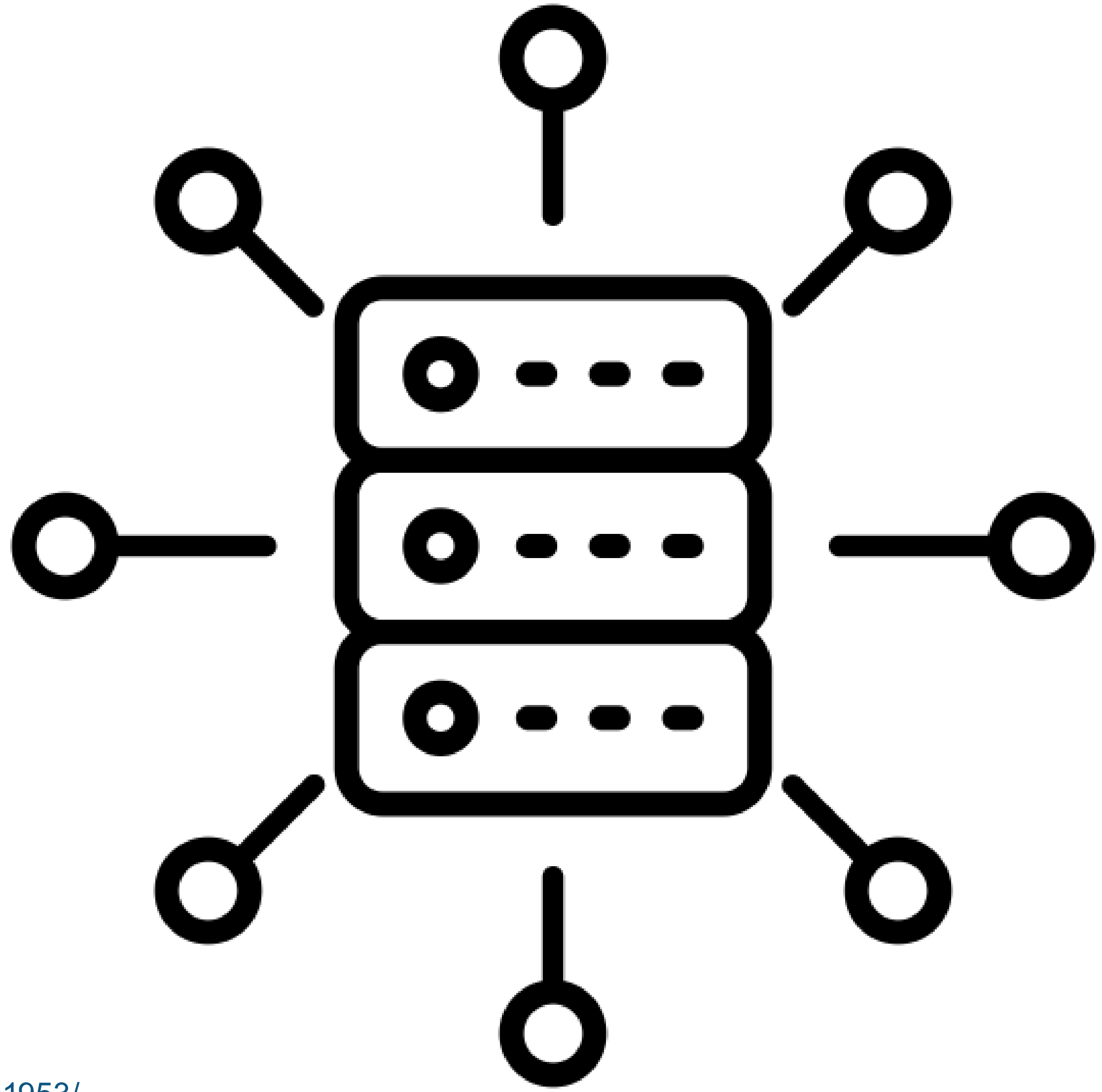
https://thenounproject.com/icon/no-service-1496954/

This kind of DoS

“The Denial of Service (DoS) attack is focused on making a resource (site, application, server) unavailable for the purpose it was designed.”

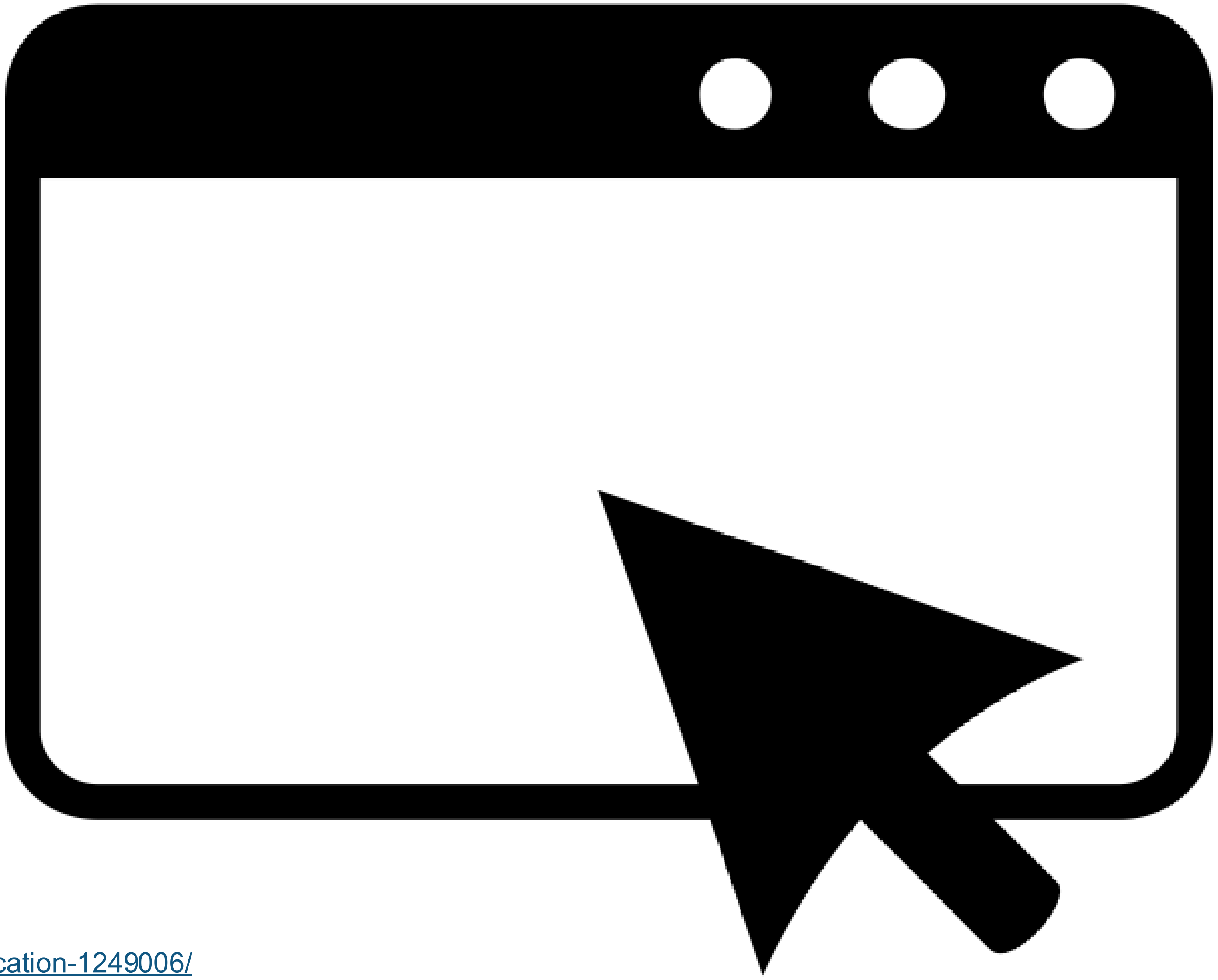
(https://owasp.org/www-community/attacks/Denial_of_Service)

DDoS – in a different lecture



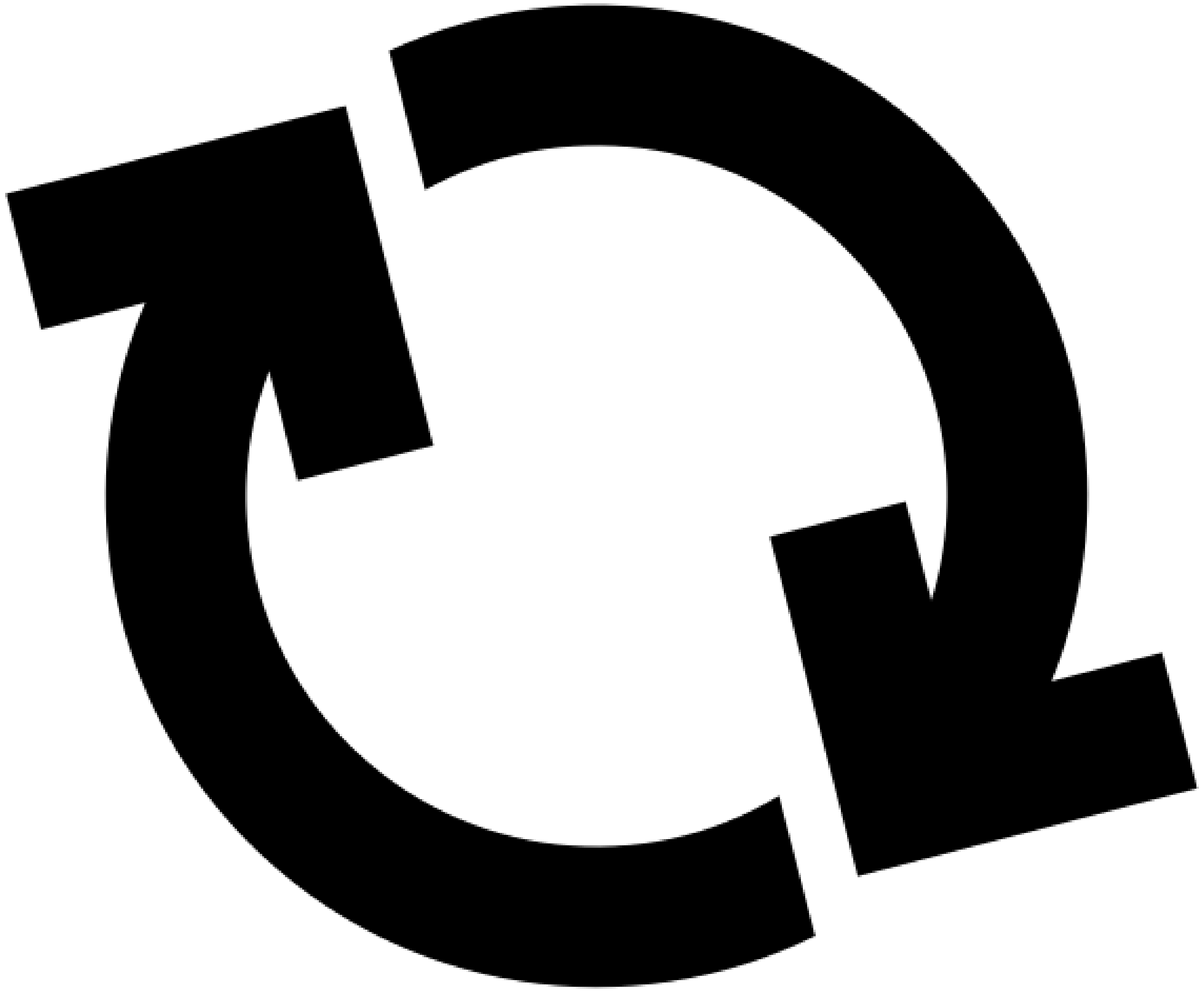
https://thenounproject.com/icon/distributed-6001953/

We want to focus on the application



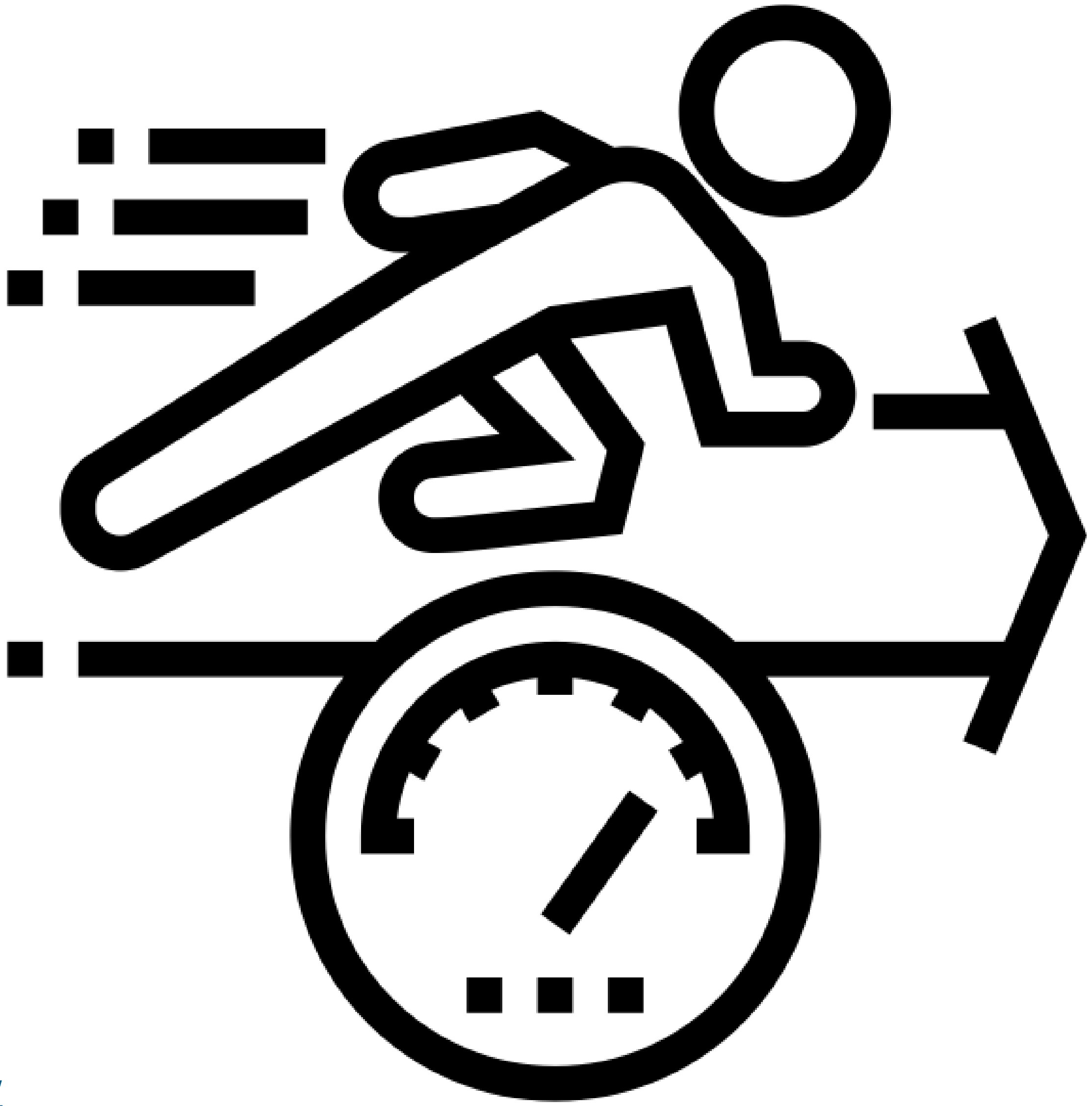
https://thenounproject.com/icon/application-1249006/

Quick Reminder: Node.js' Event Loop



https://thenounproject.com/term/redo/62716

It’s not about speed – it’s about [not] blocking others



https://thenounproject.com/icon/speed-1116526/

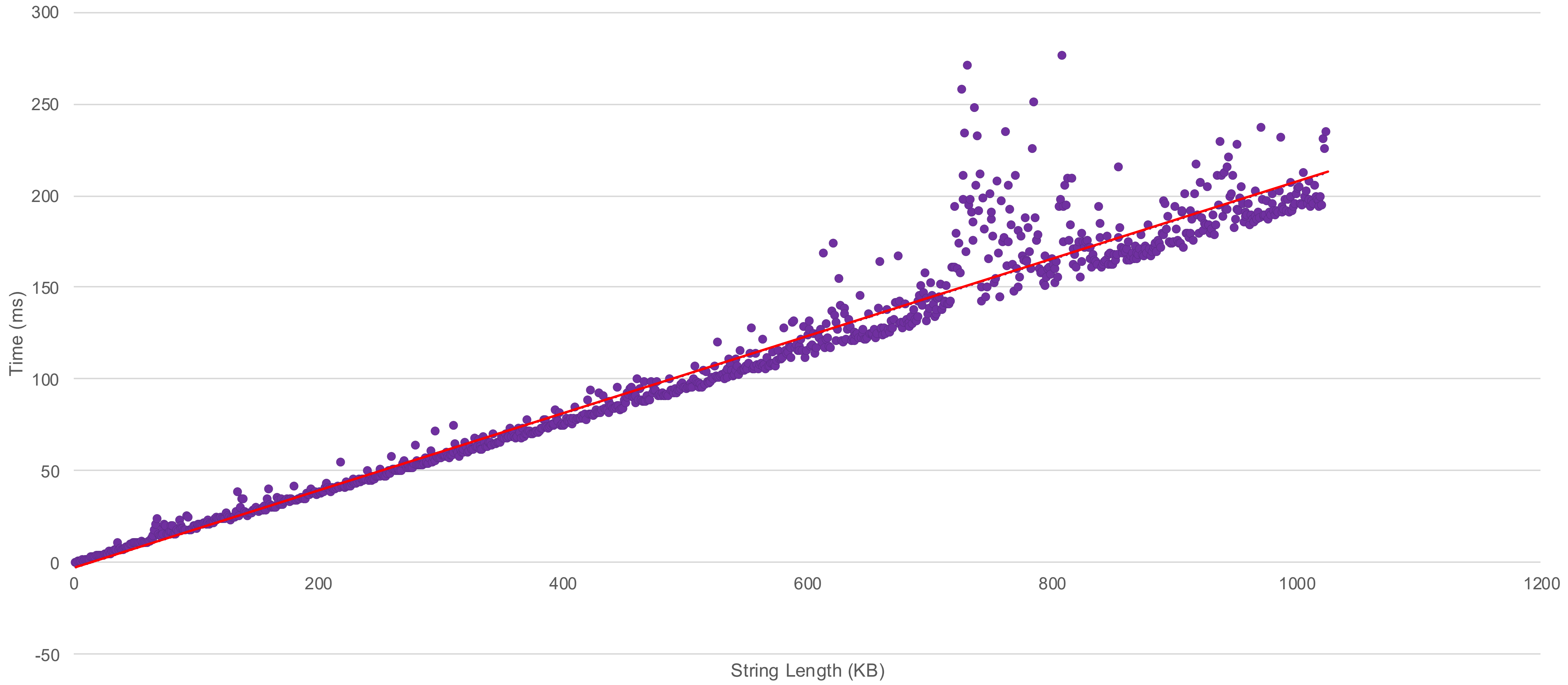
Overwork that parser (JSON Example)

```
const express = require('express');
const app = express();
app.use(express.json());

app.post('/json', (req, res) => {
    const numKeys = Object.keys(req.body).length;
    res.end(numKeys + ' keys in the payload');
});

app.listen(3000, () => console.log('Listening on port 3000'));
```

How bad is it really?



What can we do?

What can we do?

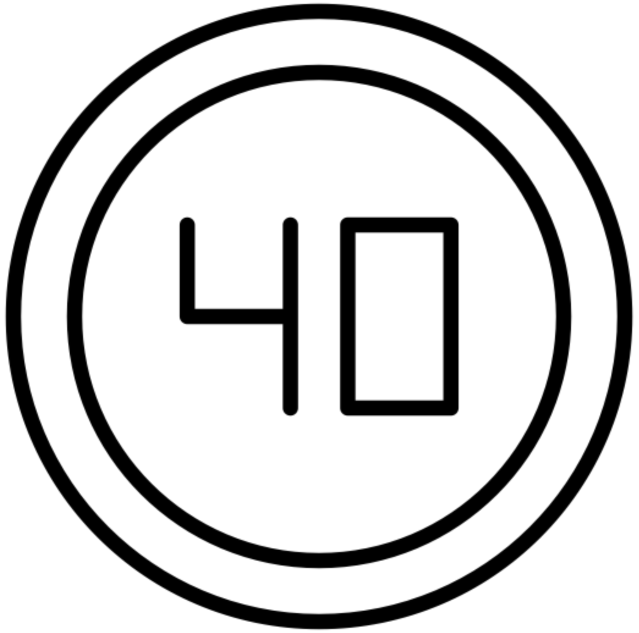
- Don’t allow tainted input to be parsed
  - Not realistic…

https://thenounproject.com/icon/no-entry-1379330/

What can we do?

- Don’t allow tainted input to be parsed
  - Not realistic…
- Limit the size of the input
  - E.g., in the above Express example:

```
app.use(express.json({limit: '40kb'}))
```



https://thenounproject.com/icon/speed-limit-4873715/

What can we do?

- Don’t allow tainted input to be parsed
  - Not realistic…
- Limit the size of the input
  - E.g., in the above Express example:
    `app.use(express.json({limit: '40kb’}))`
- Do it in the background, not the event loop
  - E.g., use a library like BFJ or JSONStream

https://thenounproject.com/icon/fade-2102225/

Bomb that parser (XML Example)

```
const express = require('express');
const app = express();
app.use(express.text({type: '*/*'}));

const libxmljs = require('libxmljs2');
const opts = {noent: true, nocdata: true, noblanks: true, huge: true};

app.post('/xml', (req, res) => {
    const parsed = libxmljs.parseXml(req.body, opts);
    res.end(parsed.childNodes().length + ' child nodes in the payload');
});

app.listen(3000, () => console.log('Listening on port 3000'));
```

Sounds serious, let’s have a laugh

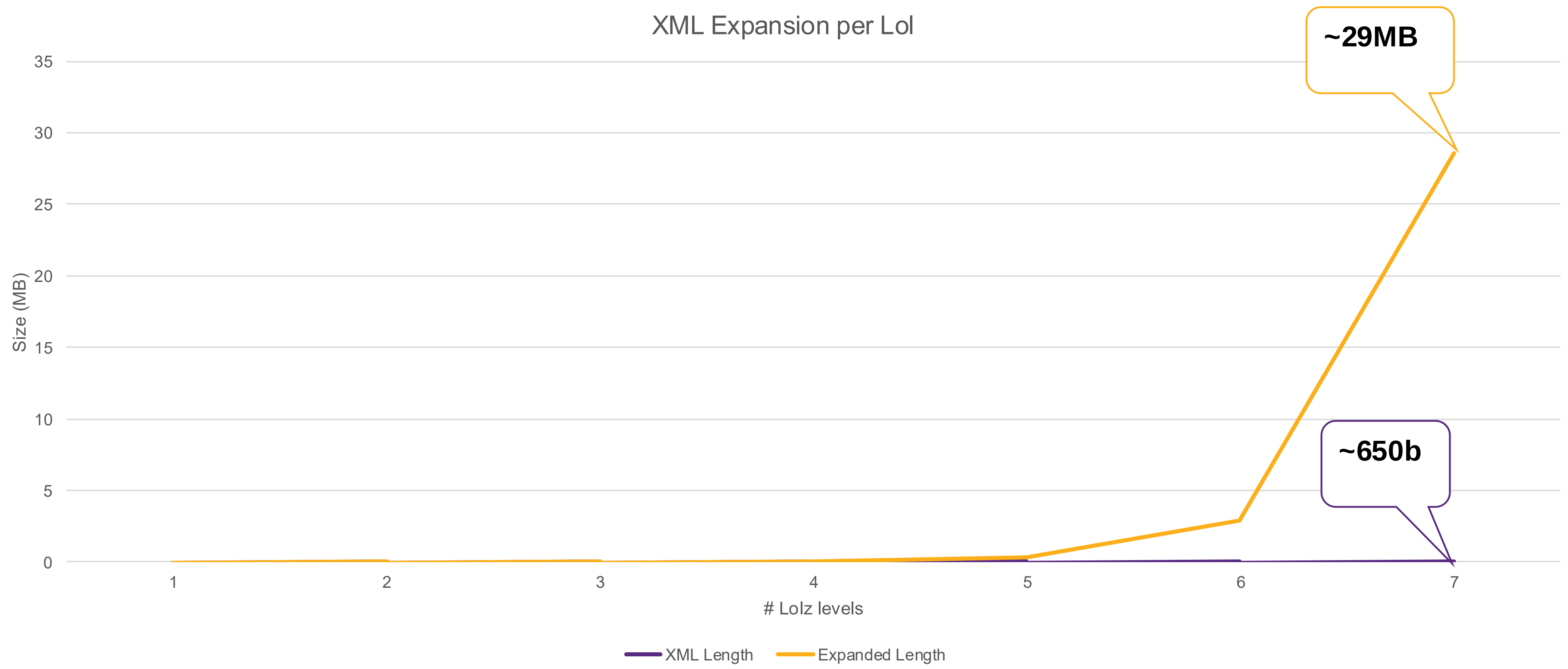
https://thenounproject.com/icon/joker-3976603/

Or a billion laughs

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol0 "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol0;&lol0;&lol0;&lol0;&lol0;&lol0;&lol0;&lol0;&lol0;&lol0;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

https://en.wikipedia.org/wiki/Billion_laughs_attack

How bad is it really?



What can we do?

What can we do?

- Don’t use XML
  - If you can…

https://thenounproject.com/icon/no-entry-1379330/

What can we do?

- Don’t use XML
  - If you can…
- Don’t allow tainted input in your XML
  - If you can…

https://thenounproject.com/icon/no-entry-1379330/

What can we do?

- Don’t use XML
  - If you can…
- Don’t allow tainted input in your XML
  - If you can…
- Configure your library to not expand entities
  - If you can…
  - libxml wrappers: `{noent: false}` or `{huge: false}`

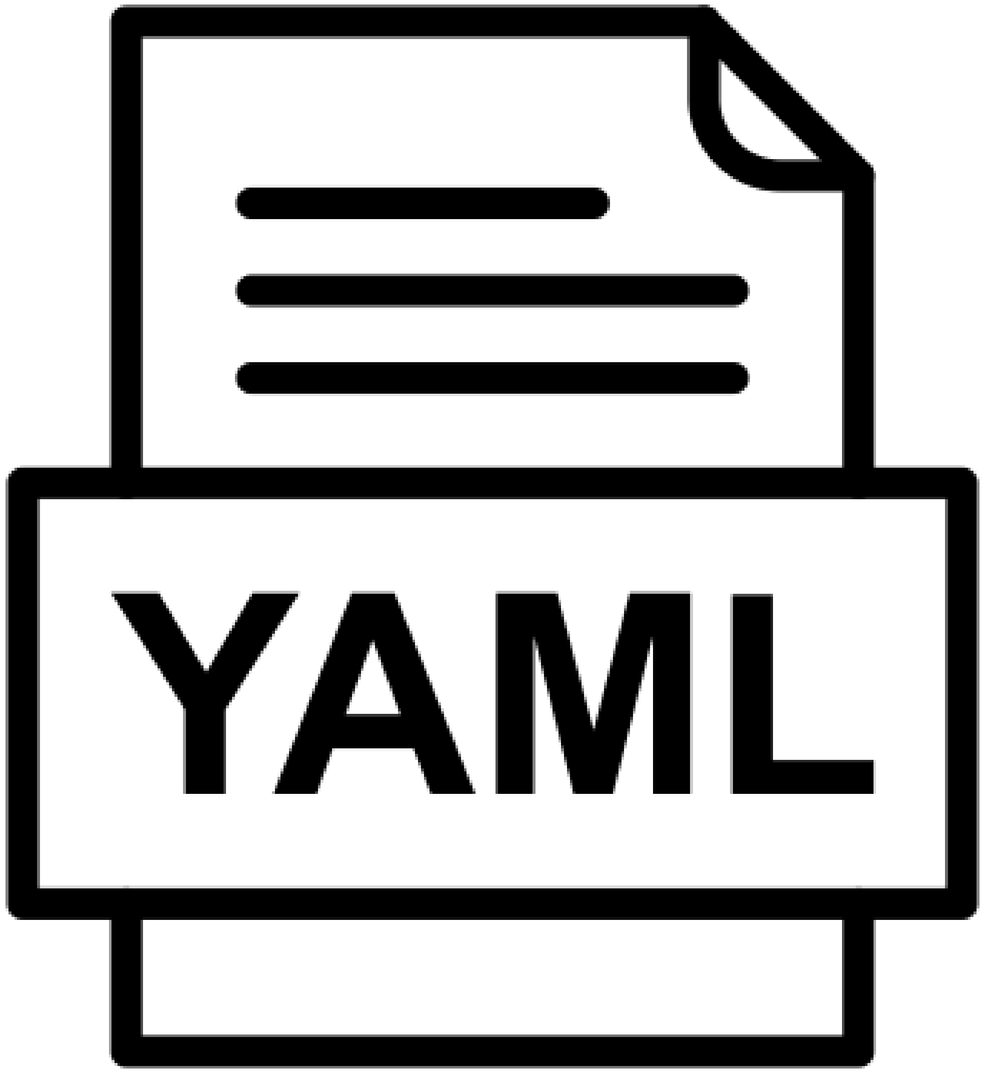
https://thenounproject.com/icon/configure-1883381/

What can we do?

- Don’t use XML
  - If you can…
- Don’t allow tainted input in your XML
  - If you can…
- Configure your library to not expand entities
  - If you can…
  - libxml wrappers:`{noent: false}` or `{huge: false}`
- Sanitize your input

https://thenounproject.com/icon/sanitizer-3470901/

Phew, I don't use XML



https://thenounproject.com/icon/yaml-file-document-icon-2598367/

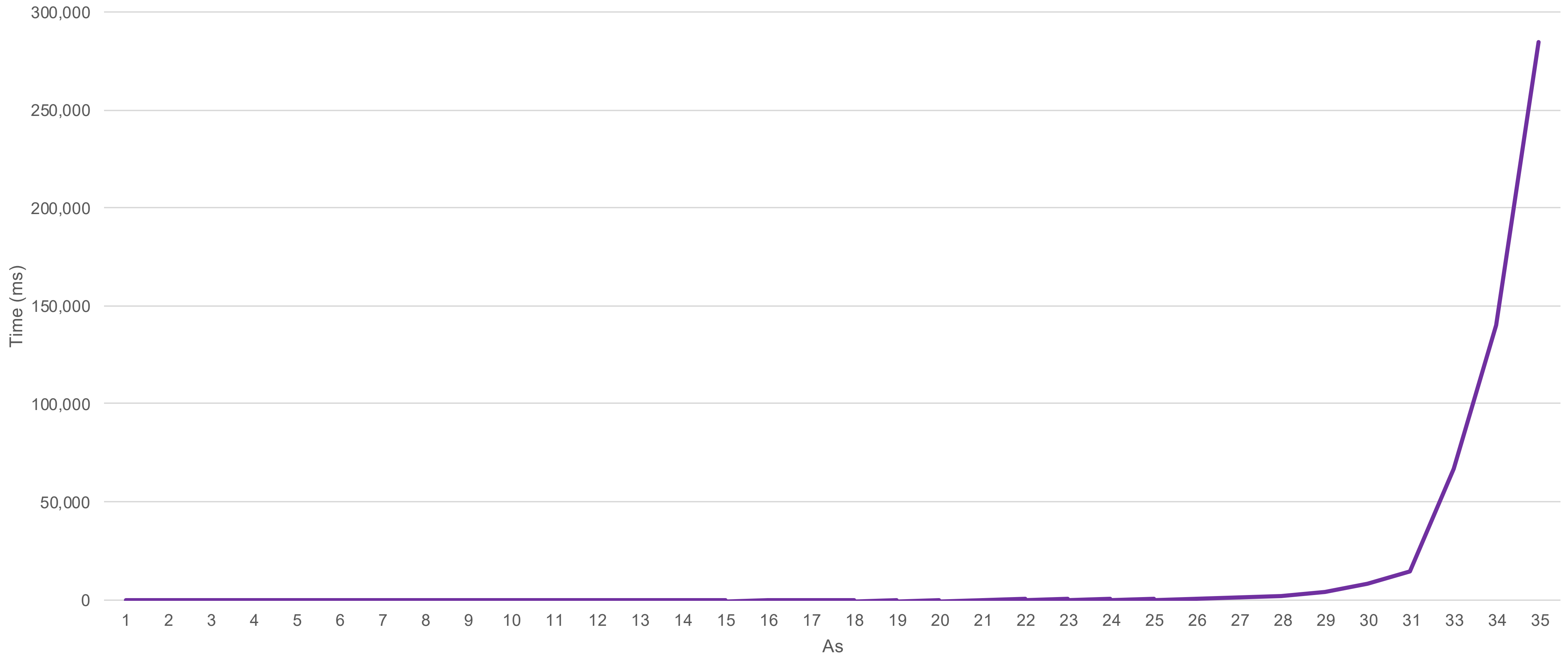
ReDoS

```
const express = require('express');
const app = express();

app.get('/regexp', (req, res) => {
   // Consider a regex like /(a+)+/
   const regexp = new RegExp(req.query.regexp);
   const text = req.query.text;
   res.end(regexp.test(text) ? 'Match!' : 'No match');
});

app.listen(3000, () => console.log('Listening on port 3000'));
```

How bad is it really?



What can we do?

What can we do?

- Check your regexes
  - SAST tools are usually pretty good at this

https://thenounproject.com/icon/check-1159941/

What can we do?

- Check your regexes
  - SAST tools are usually pretty good at this
- Don’t allow tainted input as regex
  - Not always possible…

https://thenounproject.com/icon/no-entry-1379330/

What can we do?

- Check your regexes
  - SAST tools are usually pretty good at this
- Don’t allow tainted input as regex
  - Not always possible…
- Don’t allow tainted input to be evaluated by a dodgy regex
  - Usually not possible…
  - Use length limits

https://thenounproject.com/icon/no-entry-1379330/

What can we do?

- Check your regexes
  - SAST tools are usually pretty good at this
- Don’t allow tainted input as regex
  - Not always possible…
- Don’t allow tainted input to be evaluated by a dodgy regex
  - Usually not possible…
  - Use length limits
- Think about alternatives to regex
  - re2 isn’t vulnerable to ReDoS
  - Use specific tools for specific needs (e.g., validator.js)

https://thenounproject.com/icon/alternative-3203434/

Storage (I/O) DoS

```
const fs = require('fs');
const path = require('path');
const express = require('express');
const app = express();

app.get('/lorem', (req, res) => {
    res.end(fs.readFileSync(path.join(__dirname, 'lorem.txt')));
});

app.listen(3000, () => console.log('Listening on port 3000'));
```

Storage (I/O) DoS – Remediation

The are two ways to perform storage operations in Node.js:

1. The async way
   - Delegate a storage operation to the OS, and wait for a callback
   - E.g.: `fs.readDir`, `fs.writeFile`, etc
   - 3rd parties follow similar patterns (e.g., `fs-extra`, `adm-zip`)

2. The **wrong** way

Some general take aways

https://thenounproject.com/icon/takeaway-3438027/

Don’t be a stranger

allon@blackduck.com

@mureinik

https://www.linkedin.com/in/mureinik/

Thank You