

Node만으로 버거울 때, IPC

AISEED 장진호

TypeScript Backend Meetup 2025.08.27

목차

1. 마주친 문제
2. IPC 소개
3. 실시간 영상 처리 사례
4. 운영 팁
5. 여담

마주친 문제

요구사항

- 의료 영상 라이브 스트리밍 및 녹화
- 의료 영상 익명화
 - 개인 정보 마스킹, 얼굴 블러 처리 등
- 실시간성
 - 라이브 스트리밍에도 익명화 적용
 - 수술이 끝난 즉시 녹화된 영상 볼 수 있어야 함

마주친 문제

요구사항



한 달 안에
시연 가능할까요?

마주친 문제

요구사항



(AI면 파이썬아닌가? 우리팀이 파이썬 '제품'도 잘하나?
제품 개발은 나말곤 경험이 없는데? 한 달 안에 프로덕션 레벨
개발이 가능한가? 이미지 처리도 필요한데 opencv 써야되는 거 아냐?
node에서 opencv 지원 잘 되나? 사업성 검증 빠르면 좋지 않나?)

**까짓거
한번 해보죠!**

마주친 문제

고려사항

- 폭발적인 I/O 작업량
- AI 기반 이미지 처리
- 실시간성

그리고 한 달 안에!

마주친 문제

고려사항

- 팀 구성
 - AI 엔지니어 4명(파이썬) + 제품 엔지니어 1명(Node.js)
- 파이썬 숙련도
 - 연구 코드와 제품 코드의 간극
 - 제품 개발자의 파이썬 무경험 (도메인 로직 + 복잡한 I/O)

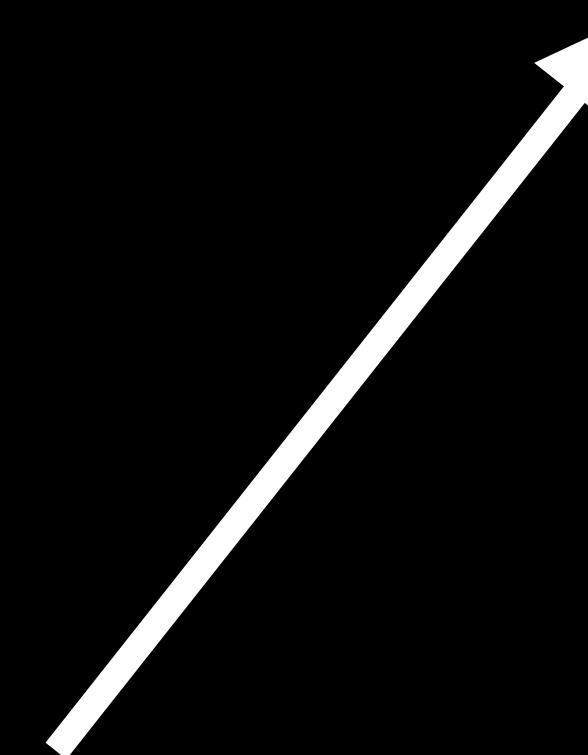
마주친 문제

시간이 없으니... Node.js로 구현?

- Node.js 생태계
 - 불안정한 opencv 라이브러리들
 - 파이썬 라이브러리와 인터페이스 불일치
 - 코드 컨버팅 문제
 - 100% 동일한 결과가 나올까?
 - 새로운 코드나 수정사항이 생기면?

마주친 문제 딜레마

익숙한 Node.js로?



숙련도는?
간편한 I/O 처리는?



이미지 처리 생각하면
Python?

마주친 문제

각자 잘하는 거 시키면 되지 않을까?

- 도메인 로직, 네트워크 통신, 영상 I/O → Node.js
- AI 기반 이미지 처리→ 파이썬 (미숙한 작업 최소화)

여러 프로세스간 협력? IPC!

IPC 소개

Inter-Process Communication

- 프로세스 간 통신
 - 서비스가 커졌을 때 역할별로 분리하여 시스템 구성 (인증 서버, 결제 서버 등)
 - IPC는 이를 로컬 PC 단위로 축소시킨 것
 - 예) 데이터베이스, 웹브라우저, docker, electron, LSP, MCP, ipykernel

IPC 소개

특징

- 로컬 통신 == 네트워크 오버헤드 없음
→ TCP/UDP 보다 낮은 지연율, 높은 처리량
- 언어/환경 독립성
→ 서로 다른 언어나 런타임을 혼합해서 구성 가능

IPC 소개

기법

- Pipe, Socket, Signal, Shared Memory, Message Queue 등
- Node.js에서 공식적으로 지원하는 방식
 - Signal
 - Pipe
 - Unix Domain Socket (Windows Named Pipe)

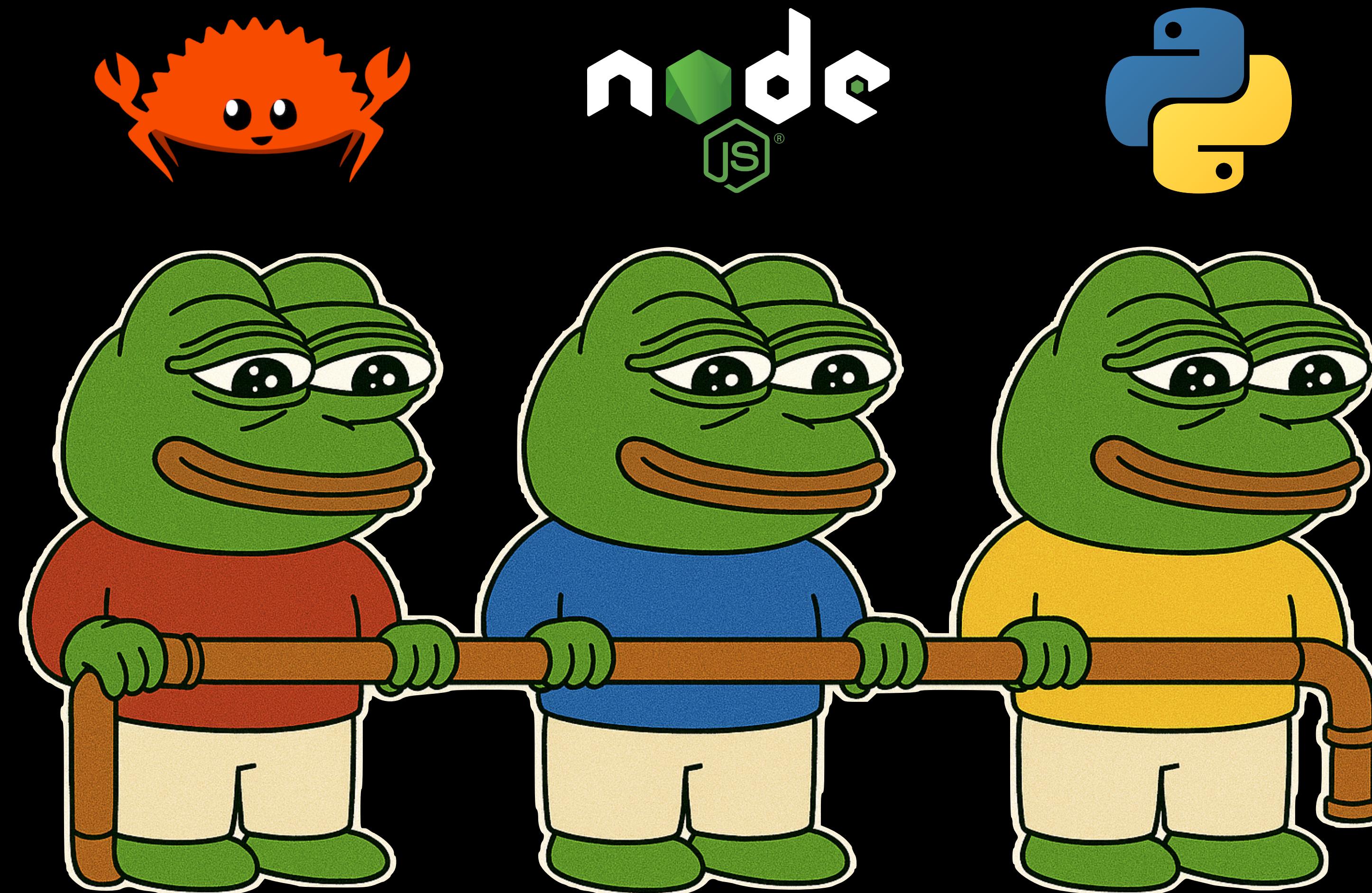
IPC 소개

Pipe

- 한 프로그램의 **출력**을 다른 프로그램의 **입력**으로 연결 (단방향)
- 여러 명령어를 파이프라인으로 연결해서 데이터를 흐르게 함
- 표준 출력(stdout)을 다음 명령어의 표준 입력(stdin)으로 전달
 - docker logs {id} | grep GET

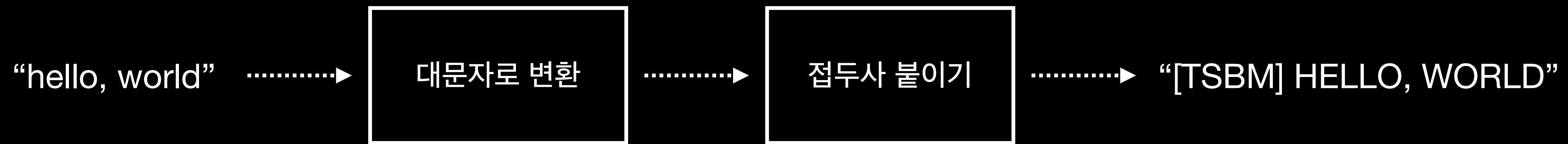
IPC 소개

Pipe



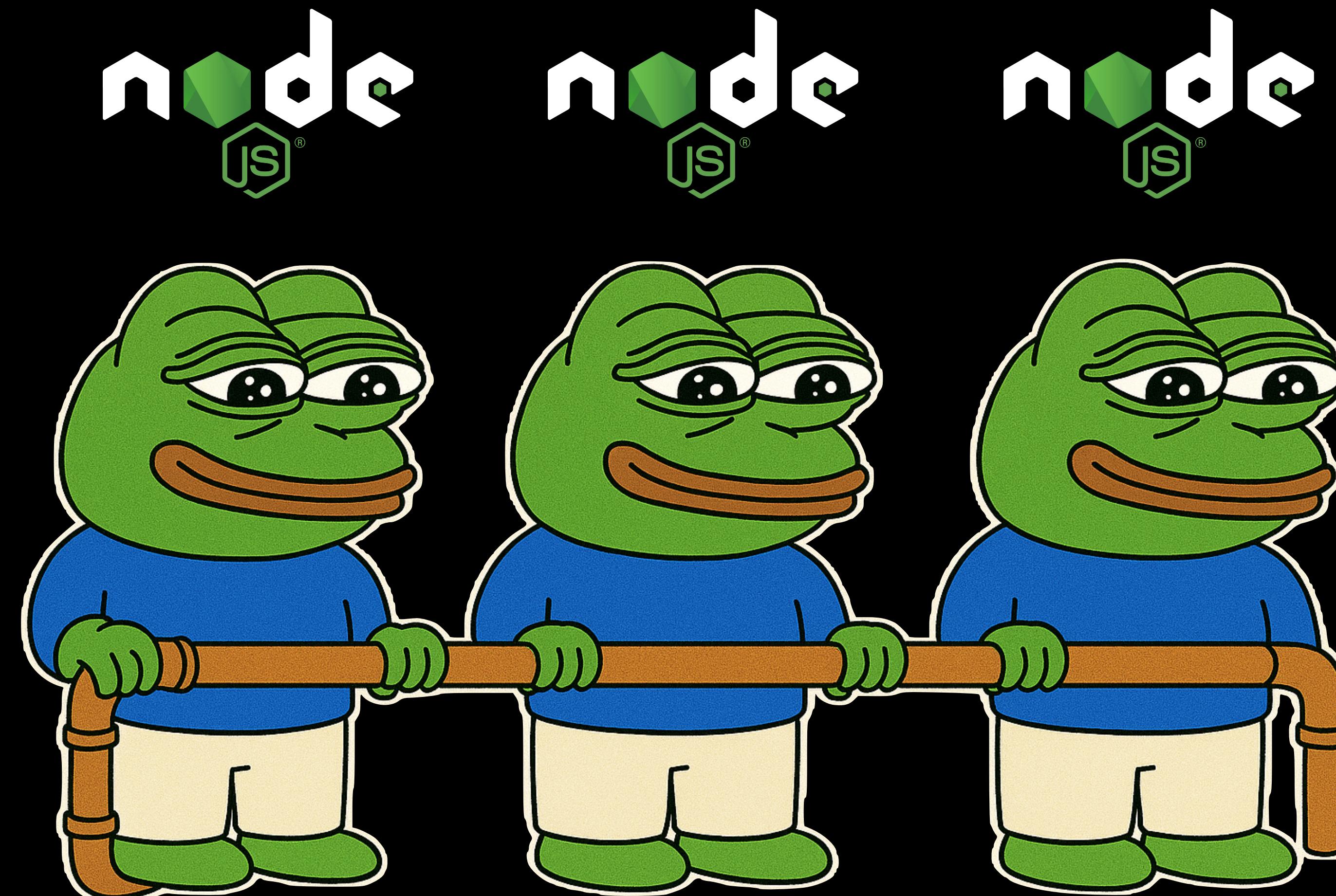
IPC 소개

Pipe



IPC 소개

Pipe



IPC 소개

Pipe



```
1 // upper-case.ts
2 process.stdin.on('data', (data: Buffer) => {
3   process.stdout.write(data.toString().toUpperCase());
4 });
5
6 // attach-prefix.ts
7 process.stdin.on('data', (data: Buffer) => {
8   process.stdout.write(`[TSBM] ${data.toString()}`);
9 });
```

IPC 소개

Pipe



IPC 소개

Pipe

```
$ echo "hello, world" | node upper-case.ts | node attach-prefix.ts
```

IPC 소개

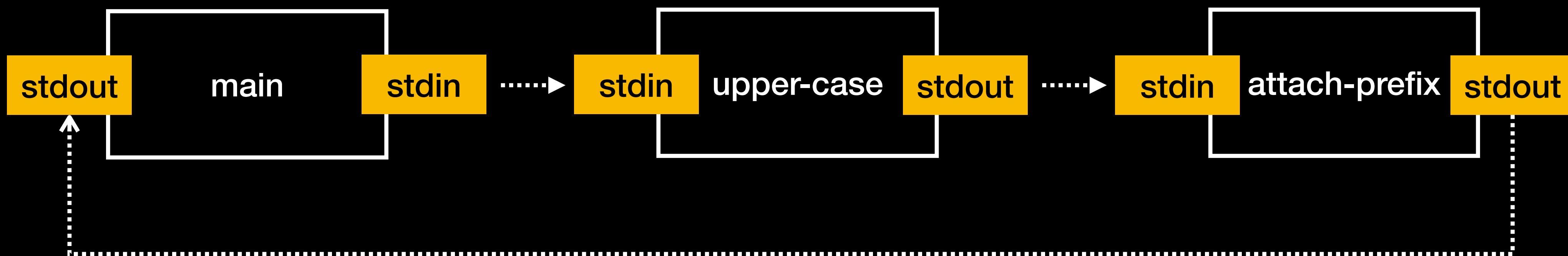
Pipe



```
1 import { spawn } from 'node:child_process';
2
3 const upperCase = spawn('node', ['upper-case.ts']);
4 const attachPrefix = spawn('node', ['attach-prefix.ts']);
5
6 process.stdin.pipe(upperCase.stdin);
7 upperCase.stdout.pipe(attachPrefix.stdin);
8 attachPrefix.stdout.pipe(process.stdout);
```

IPC 소개

Pipe



IPC 소개

Unix Domain Socket

- IPC 소켓이라고도 불림
- 소켓과 유사한 인터페이스 (양방향)
- 파일 시스템의 경로를 이용해 통신
 - /tmp/example.sock (일종의 엔드포인트)
 - 퍼미션 제한 가능

```
apple: ~ /L/Application Su/Code ➤ ls -l 1.81-main.sock
srwxr-xr-x 1 verycosy staff 0 9 1 22:28 1.81-main.sock
```

IPC 소개

Unix Domain Socket

```
$ lsof -U
```

Arc	37756	verycosy	5u	unix
Arc	37756	verycosy	79u	unix
Arc	37756	verycosy	85u	unix
Arc	37756	verycosy	86u	unix
Arc	37756	verycosy	197u	unix
Browser	37778	verycosy	99u	unix
Python	41860	verycosy	0u	unix
Python	41860	verycosy	1u	unix
Python	41860	verycosy	2u	unix
Python	41953	verycosy	2u	unix
Keynote	42662	verycosy	17u	unix
com.apple	42663	verycosy	5u	unix
Electron	44150	verycosy	45u	unix
Electron	44150	verycosy	112u	unix
Code\x20H	44698	verycosy	36u	unix

IPC 소개

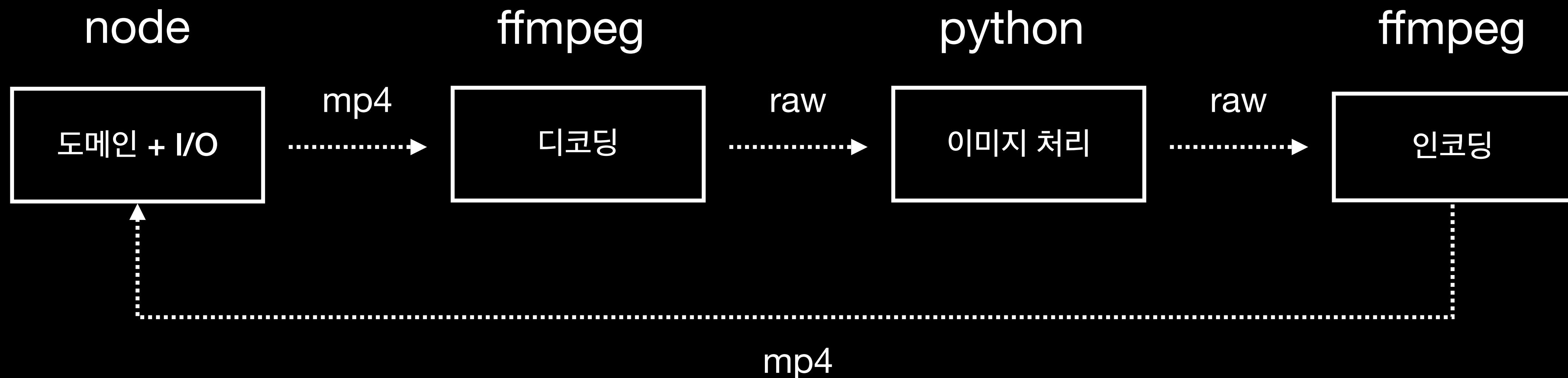
Unix Domain Socket

```
● ● ●  
1 // server.ts  
2 import net from 'node:net';  
3  
4 const socketPath = '/tmp/uds.sock';  
5  
6 const server = net.createServer((socket) => {  
7   console.log('Client connected');  
8  
9   socket.on('data', (data) => {  
10     console.log(`Received: ${data}`);  
11     socket.write('pong');  
12   });  
13 };  
14  
15 server.listen(socketPath, () => {  
16   console.log(`Server listening on ${socketPath}`);  
17 });
```

```
● ● ●  
1 // client.ts  
2 import net from 'node:net';  
3  
4 const socketPath = '/tmp/uds.sock';  
5  
6 const client = net.createConnection({  
7   path: socketPath,  
8 });  
9  
10 client.on('connect', () => {  
11   client.write('ping');  
12 });  
13  
14 client.on('data', (data) => {  
15   console.log(data.toString());  
16   client.end();  
17 });
```

실시간 영상 처리 사례

데이터 흐름



실시간 영상 처리 사례

구현 - ffmpeg

```
● ● ●  
1 // ffmpeg.ts  
2 import { spawn } from 'node:child_process';  
3  
4 export const spawnMp4ToRaw = () => {  
5   return spawn('ffmpeg', [..., '-i', 'pipe:0', ..., 'pipe:1']);  
6 };  
7  
8 export const spawnRawToMp4 = () => {  
9   return spawn('ffmpeg', [..., '-i', 'pipe:0', ..., 'pipe:1']);  
10};
```

실시간 영상 처리 사례

구현 - 파이썬



```
1 export const spawnAI = () => {
2   return spawn(`.venv/bin/python`, [`src/main.py`], {
3     cwd: './ai',
4   });
5 }
```

실시간 영상 처리 사례

구현 - 파이썬

```
● ● ●  
1 # main.py  
2 frame_size = width * height * channel  
3  
4  
5 while buffer := sys.stdin.buffer.read(frame_size):  
6     frame = buffer_to_frame(buffer, width, height, channel)  
7     darkened_frame = darken(frame)  
8  
9     sys.stdout.buffer.write(darkened_frame.tobytes())
```

실시간 영상 처리 사례

구현 - Node.js

```
● ● ●  
1 const ai = spawnAI();  
2 const mp4ToRaw = spawnMp4ToRaw();  
3 const rawToMp4 = spawnRawToMp4();  
4  
5 const srcVideo = fs.createReadStream('src.mp4');  
6 const destVideo = fs.createWriteStream('dest.mp4');  
7  
8 srcVideo.pipe(mp4ToRaw.stdin);  
9 mp4ToRaw.stdout.pipe(ai.stdin);  
10 ai.stdout.pipe(rawToMp4.stdin);  
11 rawToMp4.stdout.pipe(destVideo);
```

실시간 영상 처리 사례

결과물



운영 팁

실수 방지 및 가독성 향상



```
1 srcVideo.pipe(mp4ToRaw.stdin);
2 mp4ToRaw.stdout.pipe(ai.stdin);
3 ai.stdout.pipe(rawToMp4.stdin);
4 rawToMp4.stdout.pipe(destVideo);
```



```
1 srcVideo
2     .pipe(mp4ToRaw)
3     .pipe(ai)
4     .pipe(rawToMp4)
5     .pipe(destVideo);
```

운영 팁

실수 방지 및 가독성 향상



```
1 srcVideo.pipe(mp4ToRaw.stdin);
2 mp4ToRaw.stdout.pipe(ai.stdin);
3 ai.stdout.pipe(rawToMp4.stdin);
4 rawToMp4.stdout.pipe(destVideo);
```

... Duplex ???

쓰는 곳(stdin)과 읽는 곳(stdout)이 둘 다 있다?

운영 팁

실수 방지 및 가독성 향상



```
1 import { Duplex } from 'node:stream';
2 import type { ChildProcessWithoutNullStreams } from 'node:child_process';
3
4 export class StdioStream extends Duplex {
5   private readonly child: ChildProcessWithoutNullStreams;
6
7   constructor(child: ChildProcessWithoutNullStreams) {
8     // ...
9   }
10
11   // ...
12 }
```

운영 팁

실수 방지 및 가독성 향상

- 일반적인 read 작업과 살짝 다름
- 배압 관리
: 표준 출력 스트림 중단/재개

```
1 constructor(child: ChildProcessWithoutNullStreams) {  
2     super();  
3  
4     this.child = child;  
5     this.child.stdout  
6         .on('data', (data: Buffer) => {  
7             if (!this.push(data)) {  
8                 this.child.stdout.pause();  
9             }  
10        })  
11        .on('end', () => {  
12            this.push(null);  
13        });  
14    }  
15  
16    _read(): void {  
17        if (!this.child.stdout.readableFlowing) {  
18            this.child.stdout.resume();  
19        }  
20    }  
21}
```

운영 팁

실수 방지 및 가독성 향상



```
1 _write(chunk: Buffer, encoding: BufferEncoding, callback: Callback): void {
2   if (!this.child.stdin.write(chunk)) {
3     this.child.stdin.once('drain', callback);
4   } else {
5     callback();
6   }
7 }
```

운영 팁

실수 방지 및 가독성 향상

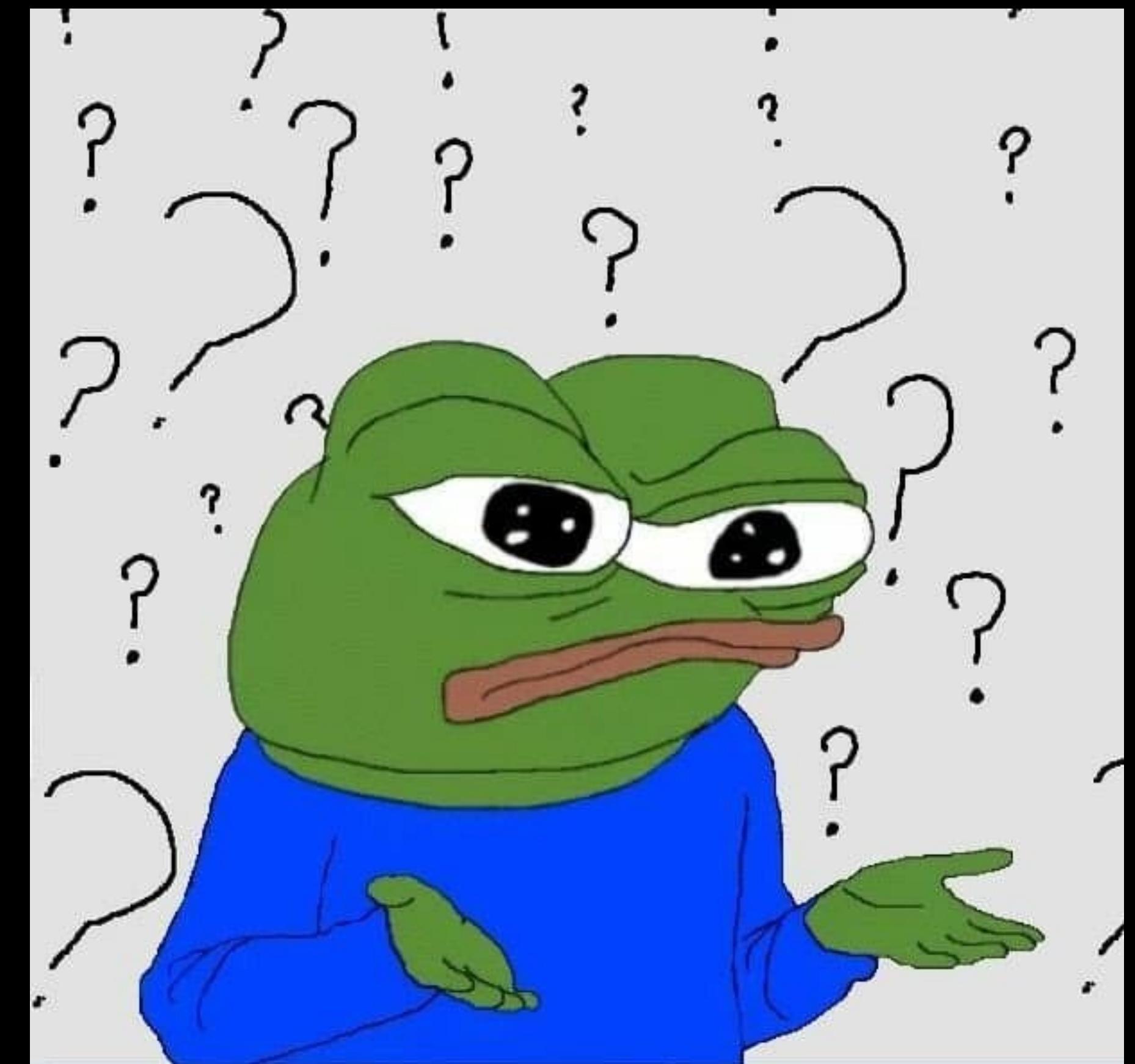


```
1 const ai = new StdioStream(spawnAI());
2 const mp4ToRaw = new StdioStream(spawnMp4ToRaw());
3 const rawToMp4 = new StdioStream(spawnRawToMp4());
4
5 const srcVideo = fs.createReadStream('src.mp4');
6 const destVideo = fs.createWriteStream('dest.mp4');
7
8 srcVideo
9   .pipe(mp4ToRaw)
10  .pipe(ai)
11  .pipe(rawToMp4)
12  .pipe(destVideo);
```

운영 팁

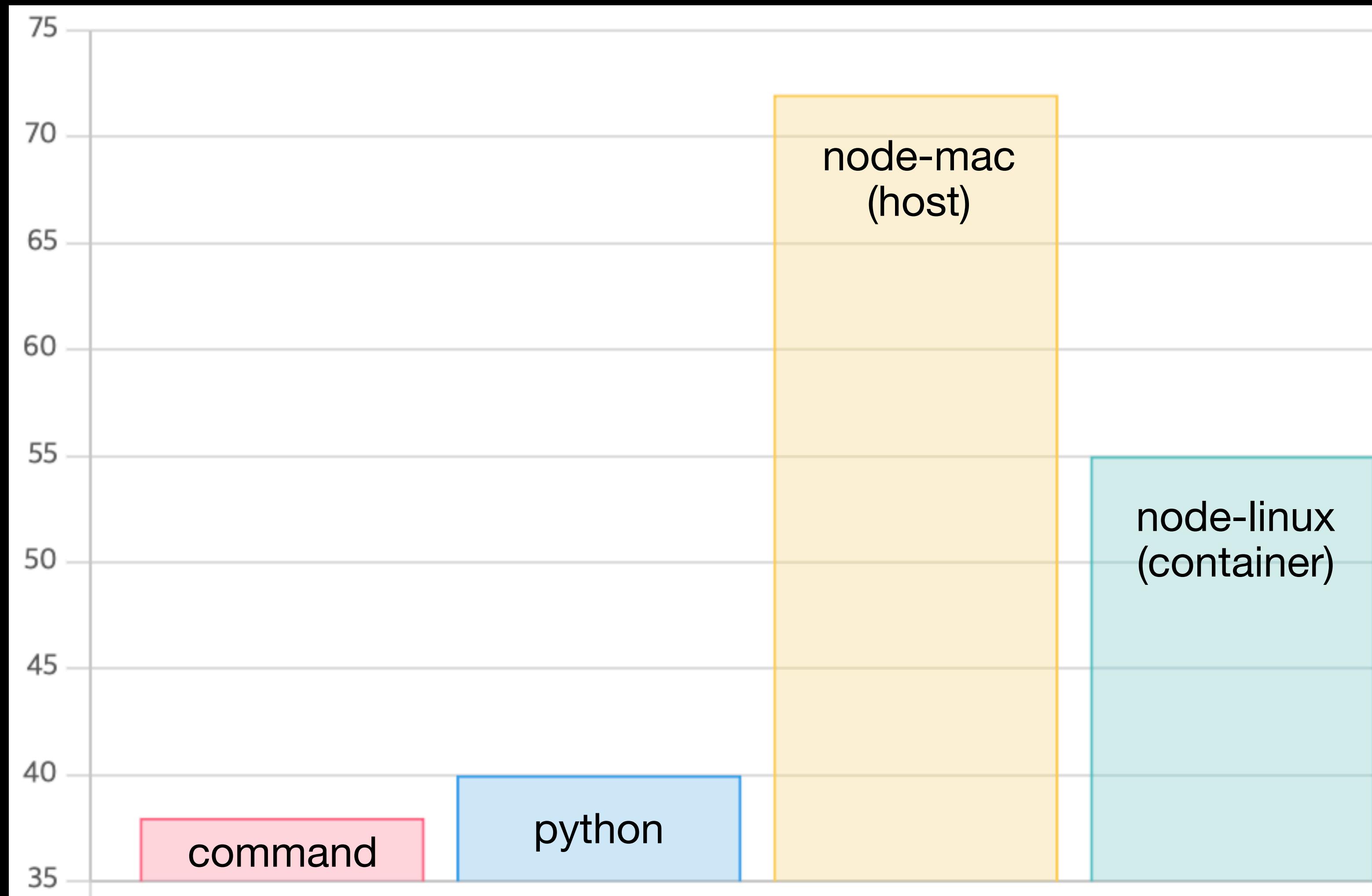
성능

- 개발 환경(macOS)에서 유독 느리다
 - macOS(호스트)가 리눅스(컨테이너)보다 1.3배
 - 동일한 로직의 python 코드 대비 1.8배



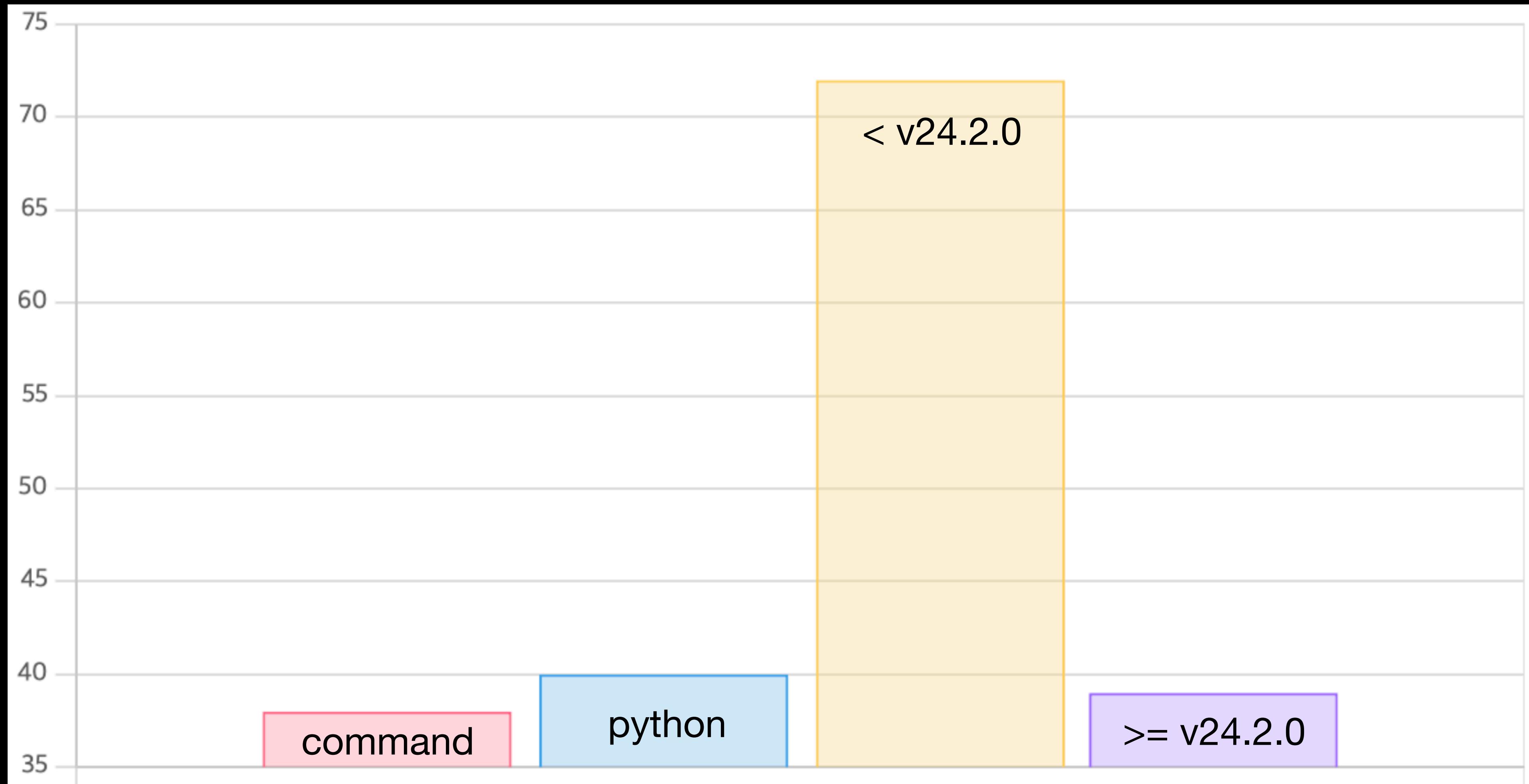
운영 팁

성능



운영 팁

성능



운영 팁

성능

- v24.2.0 주요 변경사항

libuv 1.50.0 → 1.51.0

[[3b1e4bdbbb](#)] - **deps:** update libuv to 1.51.0 (Node.js GitHub Bot) [#58124](#)

운영 팁

성능

- libuv는 자식 프로세스를 생성할 때 부모-자식 IPC 목적으로 UDS를 만듦
 - macOS의 UDS 기본 버퍼 크기는 8KB (리눅스는 최소 160KB 이상)
 - Node.js 자식 프로세스의 표준입출력 버퍼 크기(highWaterMark)는 64KB



→ libuv 레벨의 성능 문제
(1.51.0에서 UDS 64KB로 증량됨)

운영 팁

성능



```
1 srcVideo.pipe(mp4ToRaw.stdin);  
2 mp4ToRaw.stdout.pipe(ai.stdin);  
3 ai.stdout.pipe(rawToMp4.stdin);  
4 rawToMp4.stdout.pipe(destVideo);
```

... 근데 우리 pipe로 구현하지 않았나요?

운영 팁

성능

- libuv 코드를 뜯어보면, Node.js 자식 프로세스 표준입출력 pipe는 사실 UDS

```
● ● ●  
1 case UV_CREATE_PIPE:  
2     assert(container->data.stream != NULL);  
3     if (container->data.stream->type != UV_NAMED_PIPE)  
4         return UV_EINVAL;  
5     else  
6         return uv_socketpair(SOCK_STREAM, 0, fds, 0, 0);
```

운영 팁

성능

- libuv 코드를 뜯어보면, Node.js 자식 프로세스 표준입출력 pipe는 사실 UDS

1. 'pipe' : Create a pipe between the child process and the parent process. The parent end of the pipe is exposed to the parent as a property on the `child_process` object as `subprocess.stdio[fd]`. Pipes created for fds 0, 1, and 2 are also available as `subprocess.stdin`, `subprocess.stdout` and `subprocess.stderr`, respectively.
These are not actual Unix pipes and therefore the child process can not use them by their descriptor files, e.g. `/dev/fd/2` or `/dev/stdout`.

여담

ㅎ ㅎ...

The screenshot shows the GitHub repository page for libuv. At the top, there's a profile picture of a green lizard, the repository name "libuv", and a "Public" button. To the right are buttons for "Edit Pins", "Watch", and a notification count of "723". Below this, there are dropdown menus for "v1.x" (selected), "13 Branches", and "250 Tags". On the right side of the header are search, file addition, and code viewing buttons. A pull request from user "verycosy" is highlighted, titled "macos: increase child process stdio buffer size (#4694)". The PR has 7894072 reviews and 5,480 commits. The timestamp indicates it was created 12 hours ago.

참고 : <https://www.verycosy.net/posts/2025/03/nodejs-ipc-performance-via-libuv-contribution>

여담

예제는 예제일 뿐

- <https://github.com/verycosy/tsbm-202508-ipc>
- 프로덕션에선 보완 필요
 - 특히 외부 프로세스 관리 (재시작 로직, 메모리 누수 등)
 - 적합한 통신 방식 선택, 프로토콜 설계
 - execa, JSON-RPC, ZMQ(ipc://) 등 고려

여담
매출 지분 1위!



Q&A