Node.js and Backend Development

Shan-Hung Wu
CS, NTHU

Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

Clone `hello-node`

```
$ npm install --save yargs
```

- Yargs
  - Command line interface (CLI) tools

- Usage:

```
$ node src/main.js <command> [options]
```

- Javscript runtime engine
- Why Node.js in the backend?
  - Event-based and asynchronous I/O; very fast
  - NPM and large ecosystem

Node Runtime

```
$ node app.js
$ node   // REPL
```

- Global objects:
  - `window` → `global` (scope)
  - `document` → `process` (`.env`, `.argv()`, `.exit()` etc.)
  - `os`, `fs`, `module.exports`
- Node.js supports ES6
  - No need for Babel
  - No `import` (use `require()` instead)
- See API Docs

Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

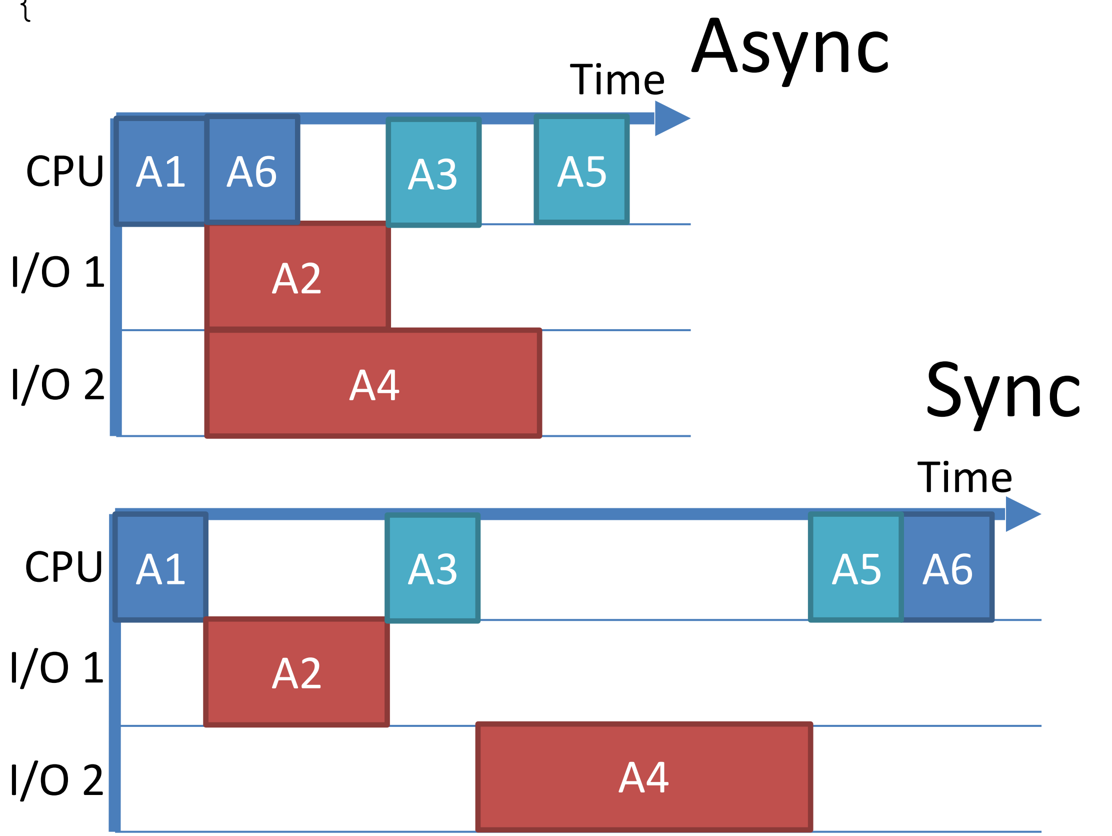
WeatherMood Posts on Node.js

Why fast as an backend engine?

Asynchronous I/O

```
// A.js
...    // 1
fs.readReadFile('file.name', (err, data) => {
  ... // 3
});    // 2
http.get('url', res => {
  ... // 5
});    // 4
...    // 6
```

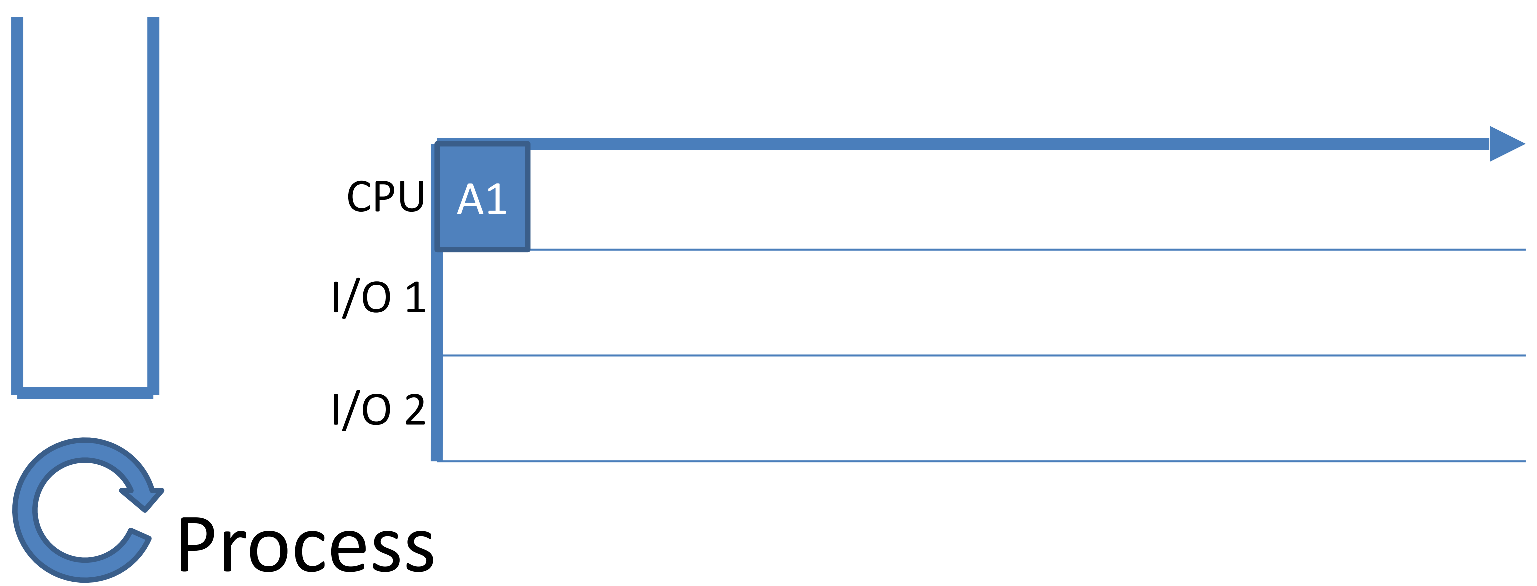
- Shortens exec time
  - If with parallel I/O channels



Single-Threaded Event Loop

```
// User A.js
...     // 1
fs.readReadFile(..., () => {
  ...   // 3
});     // 2
http.get(..., () => {
  ...   // 5
});     // 4
...     // 6
```

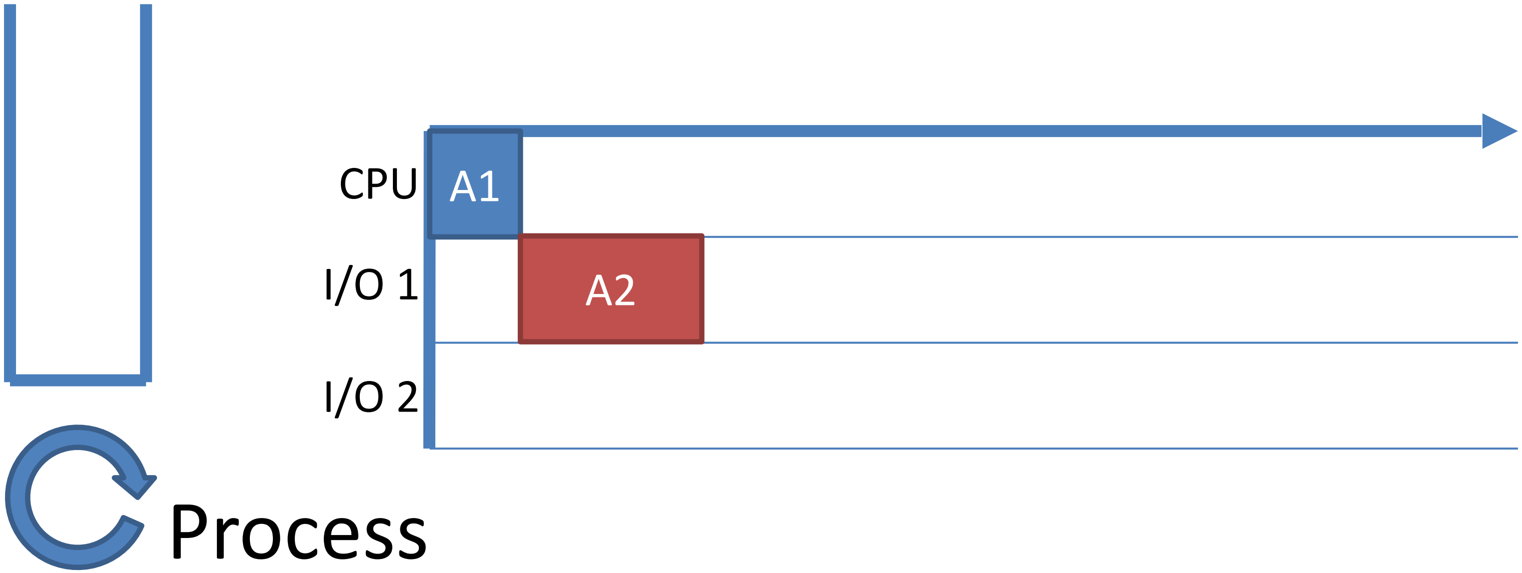
```
// User B.js
...     // 1
fs.readReadFile(..., () => {
  ...   // 3
});     // 2
http.get(..., () => {
  ...   // 5
});     // 4
...     // 6
```



Single-Threaded Event Loop

```
// A.js
...       // 1
fs.readReadFile(..., () => {
  ...     // 3
});       // 2
http.get(..., () => {
  ...     // 5
});       // 4
...       // 6
```

```
// B.js
...       // 1
fs.readReadFile(..., () => {
  ...     // 3
});       // 2
http.get(..., () => {
  ...     // 5
});       // 4
...       // 6
```



Single-Threaded Event Loop

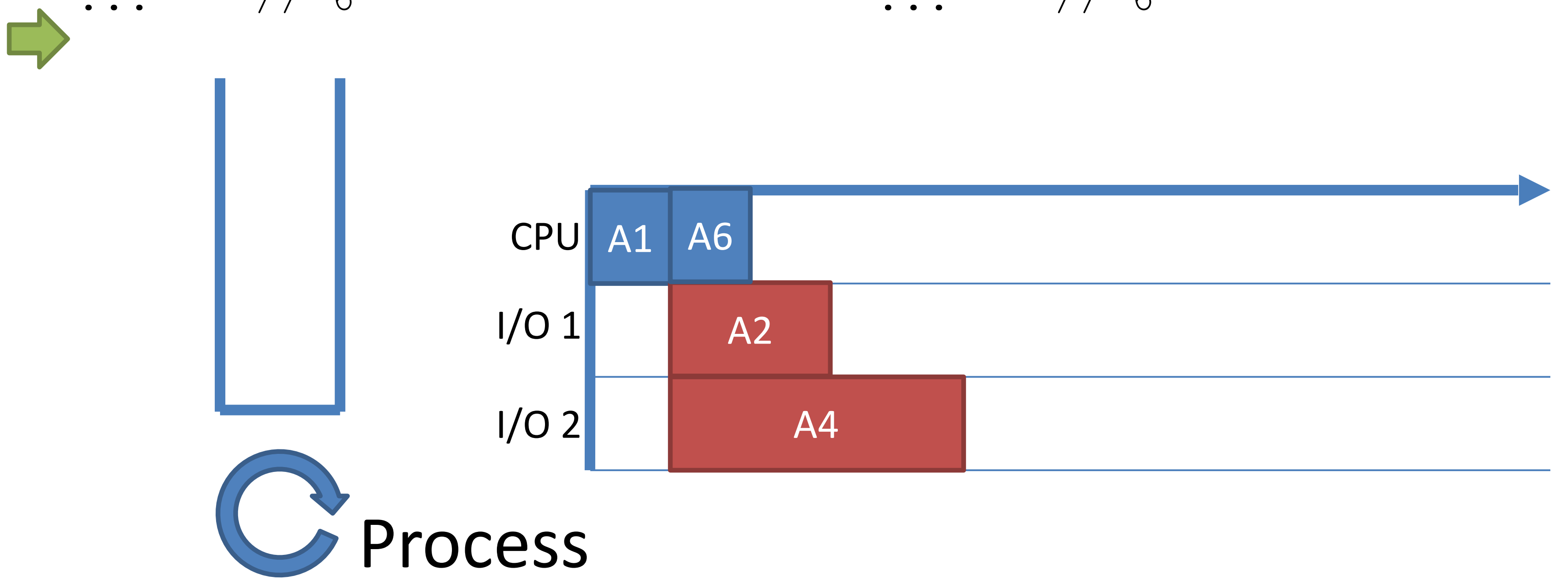
```
// A.js
...       // 1
fs.readReadFile(..., () => {
  ...   // 3
});       // 2
http.get(..., () => {
  ...   // 5
});       // 4
...       // 6
```

```
// B.js
...       // 1
fs.readReadFile(..., () => {
  ...   // 3
});       // 2
http.get(..., () => {
  ...   // 5
});       // 4
...       // 6
```

Single-Threaded Event Loop

```
// A.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```

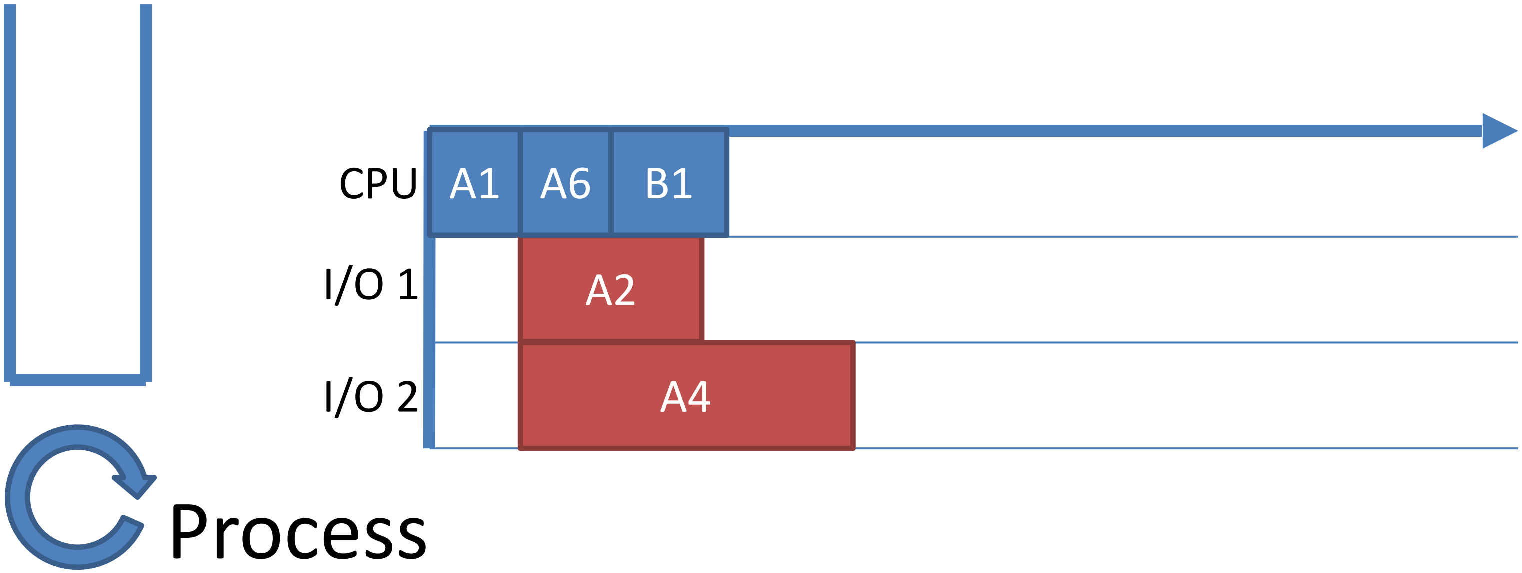
```
// B.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```



Single-Threaded Event Loop

```
// A.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```

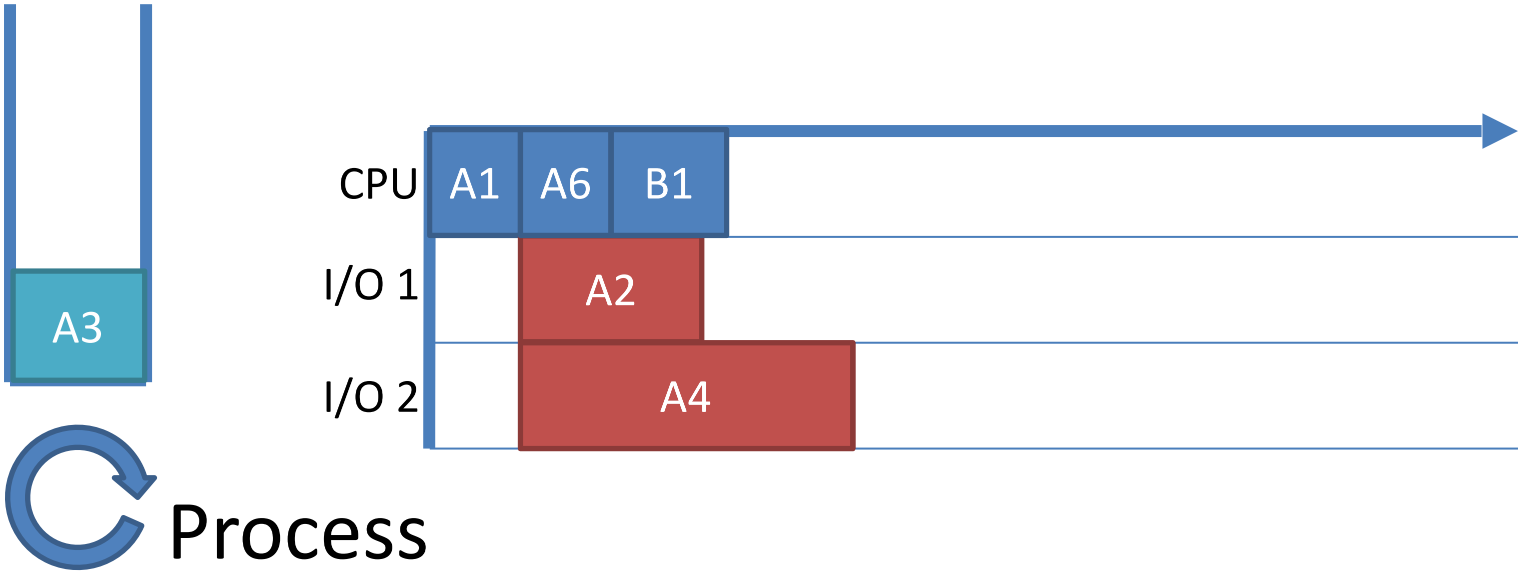
```
// B.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```



Single-Threaded Event Loop

```
// A.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```

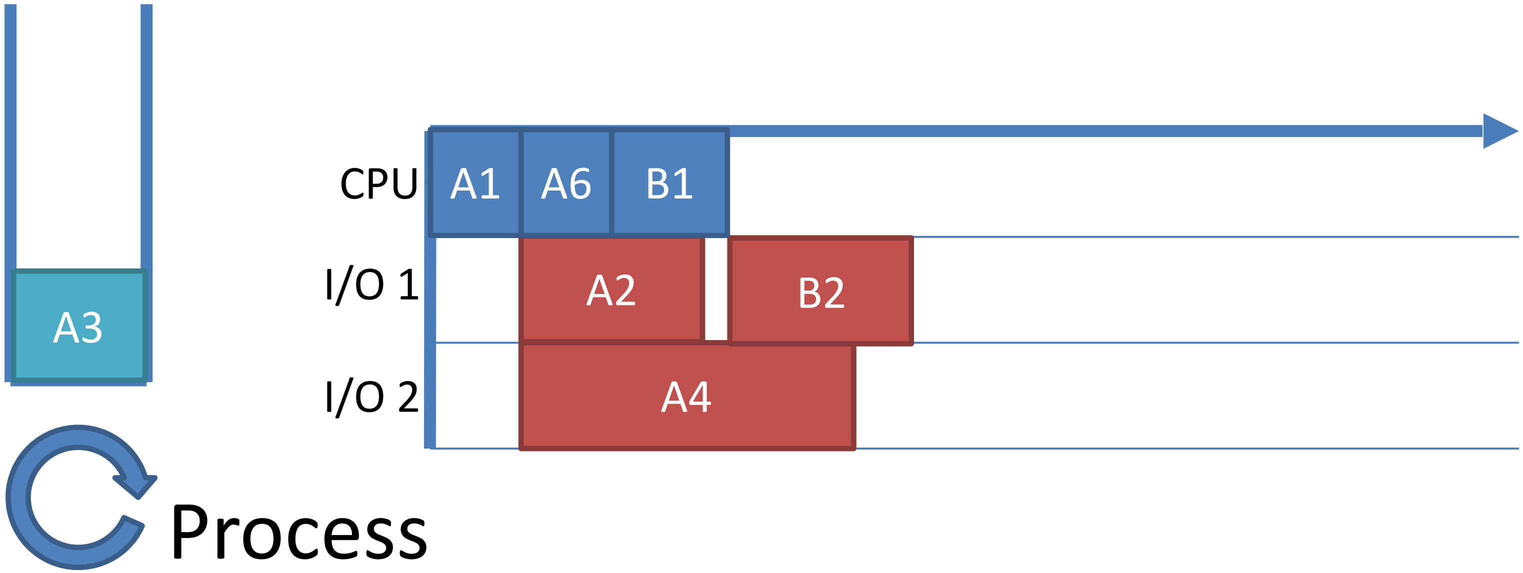
```
// B.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```



Single-Threaded Event Loop

```
// A.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```

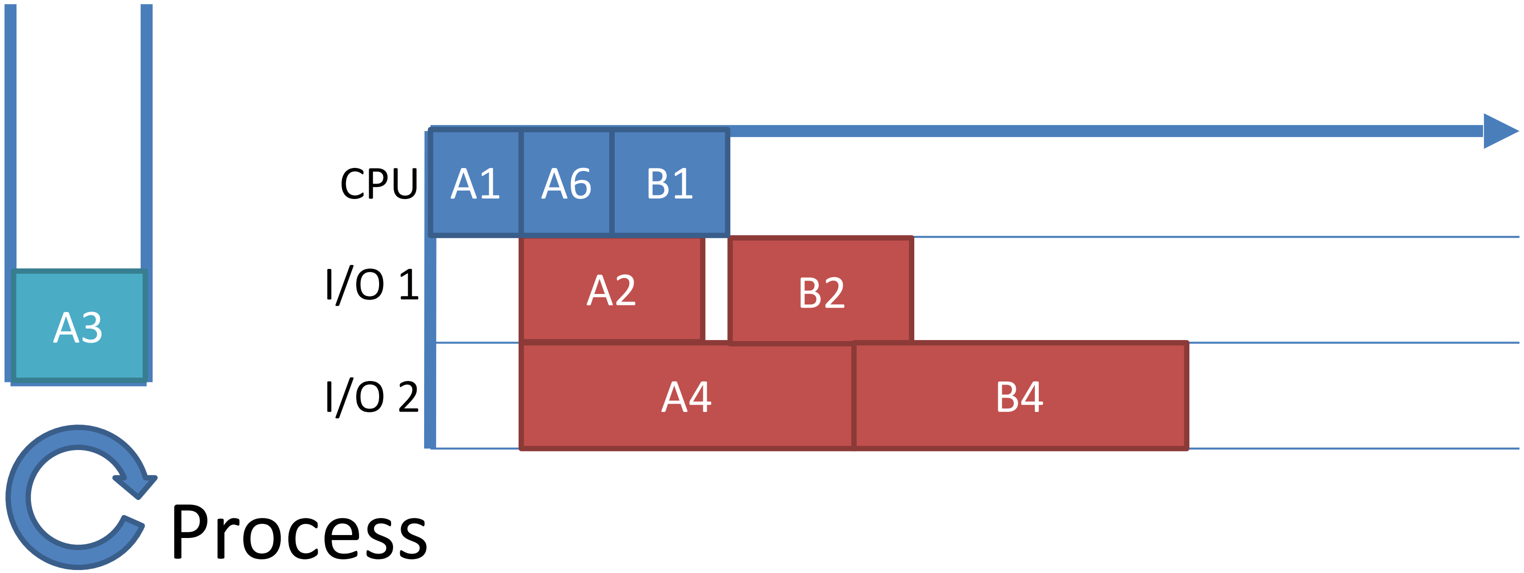
```
// B.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```



Single-Threaded Event Loop

```
// A.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```

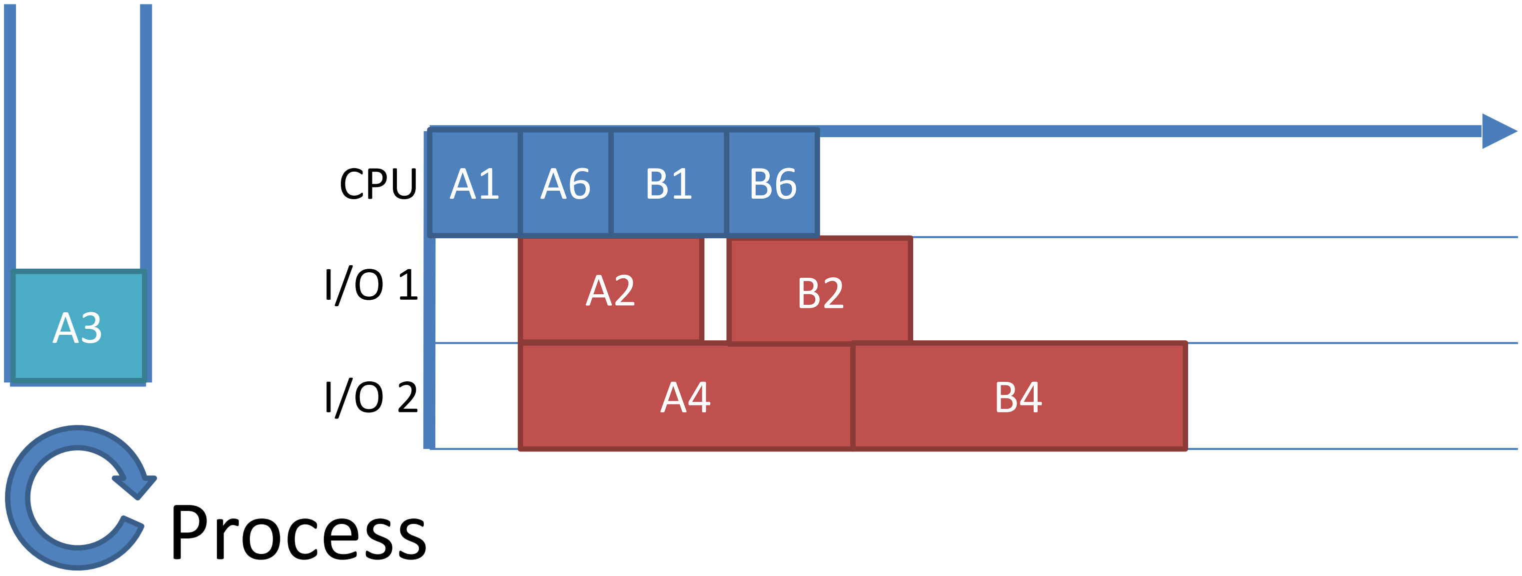
```
// B.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```



Single-Threaded Event Loop

```
// A.js
...     // 1
fs.readReadFile(..., () => {
  ...   // 3
});     // 2
http.get(..., () => {
  ...   // 5
});     // 4
...     // 6
```

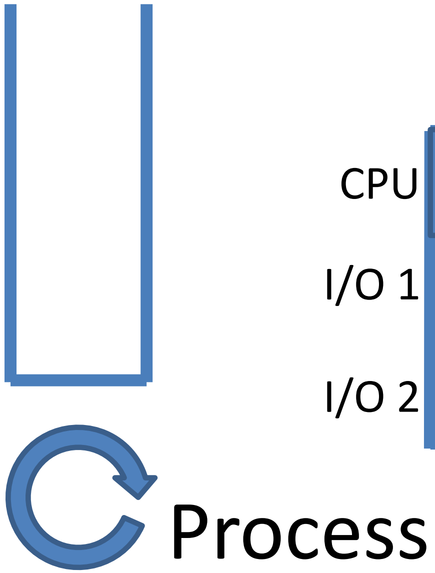
```
// B.js
...     // 1
fs.readReadFile(..., () => {
  ...   // 3
});     // 2
http.get(..., () => {
  ...   // 5
});     // 4
...     // 6
```

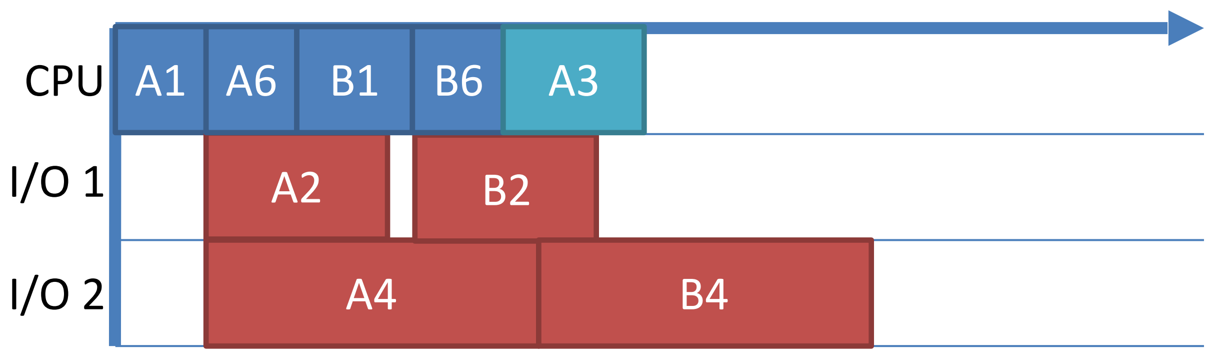


Single-Threaded Event Loop

```
// A.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```

```
// B.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```

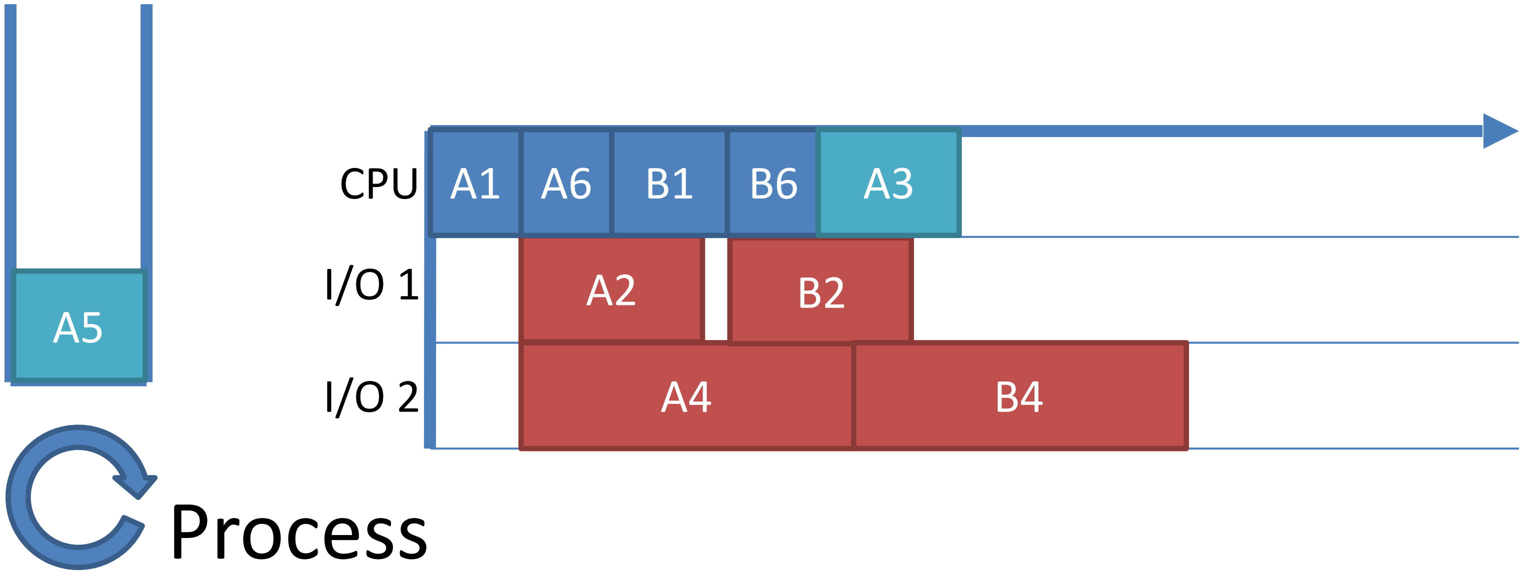




Single-Threaded Event Loop

```
// A.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```

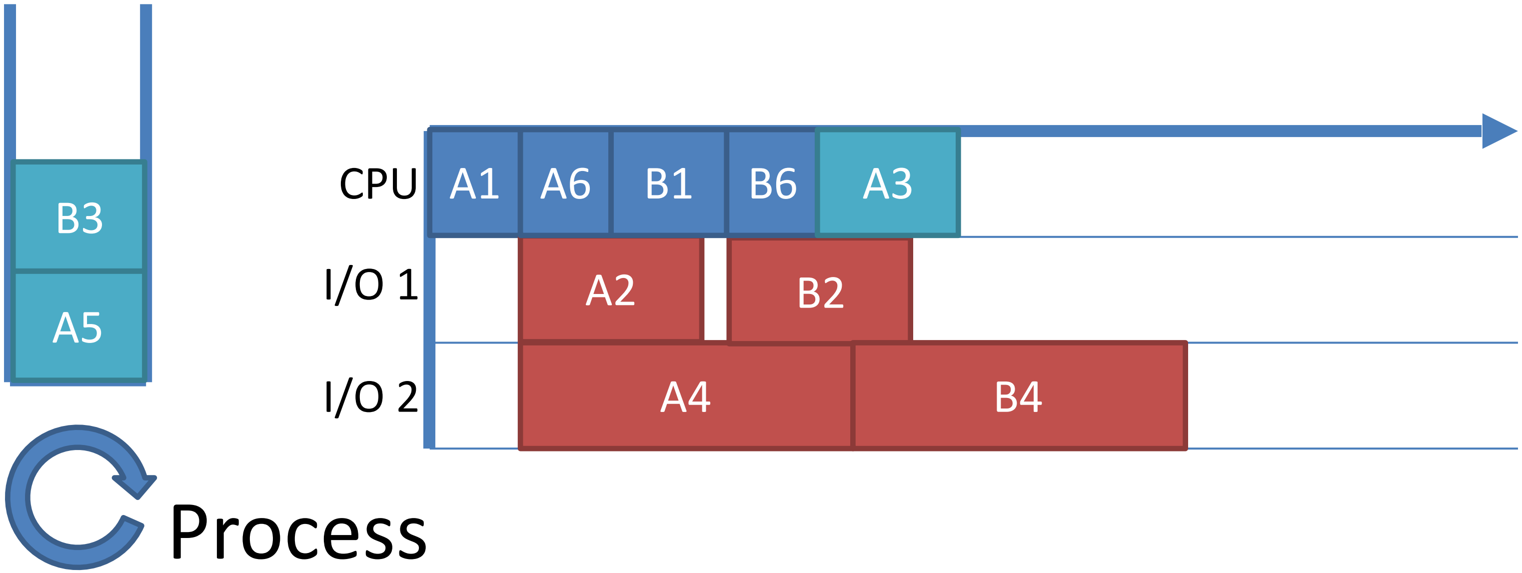
```
// B.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```



Single-Threaded Event Loop

```
// A.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```

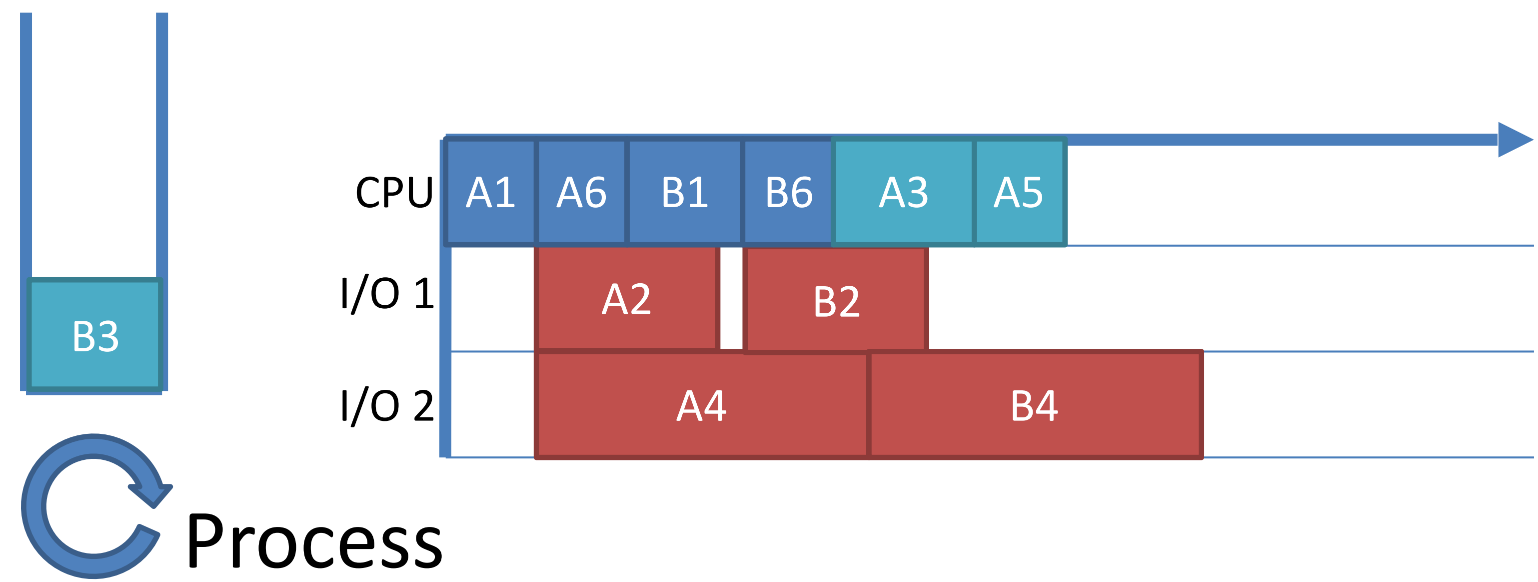
```
// B.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```



Single-Threaded Event Loop

```
// A.js
...       // 1
fs.readReadFile(..., () => {
  ...     // 3
});       // 2
http.get(..., () => {
  ...     // 5
});       // 4
...       // 6
```

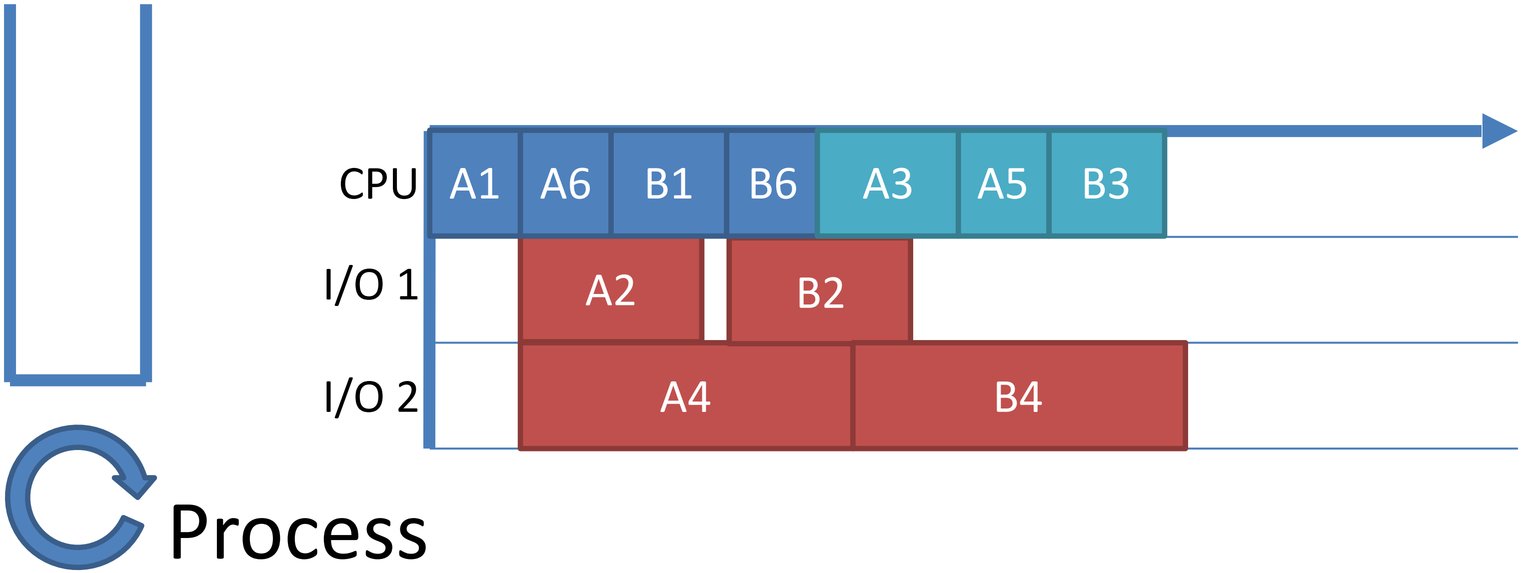
```
// B.js
...       // 1
fs.readReadFile(..., () => {
  ...     // 3
});       // 2
http.get(..., () => {
  ...     // 5
});       // 4
...       // 6
```



Single-Threaded Event Loop

```
// A.js
...       // 1
fs.readReadFile(..., () => {
  ...    // 3
});       // 2
http.get(..., () => {
  ...    // 5
});       // 4
...       // 6
```

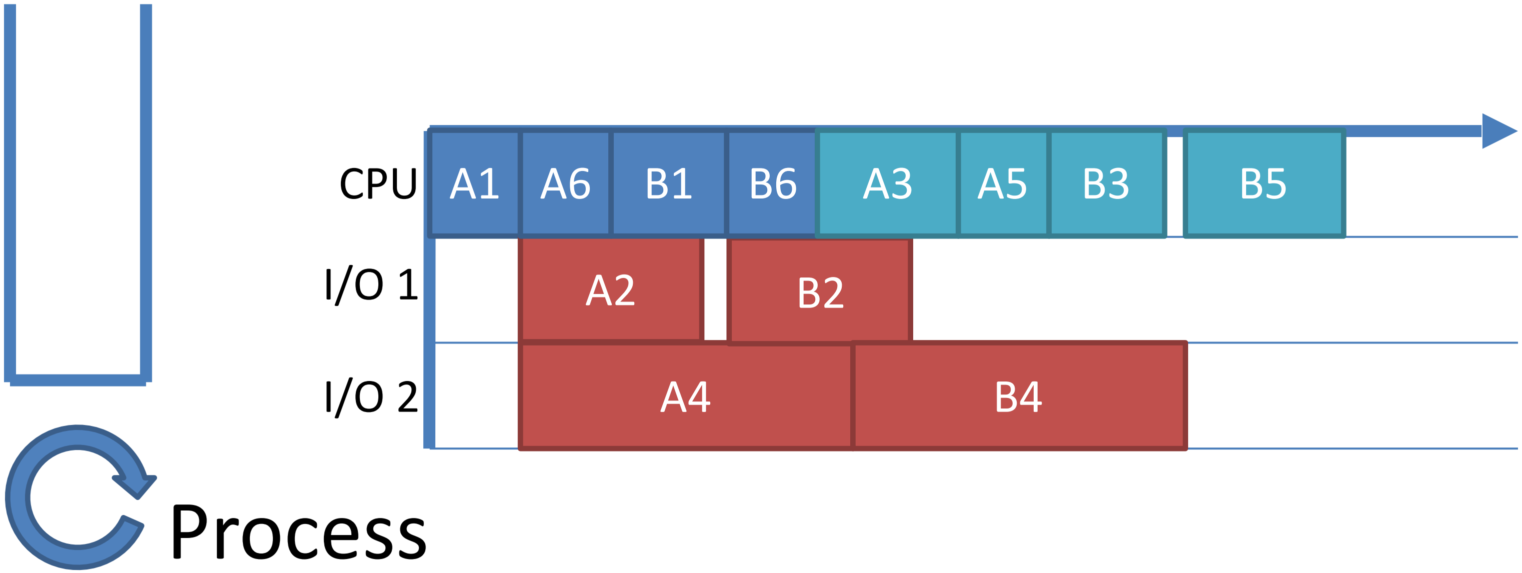
```
// B.js
...       // 1
fs.readReadFile(..., () => {
  ...    // 3
});       // 2
http.get(..., () => {
  ...    // 5
});       // 4
...       // 6
```

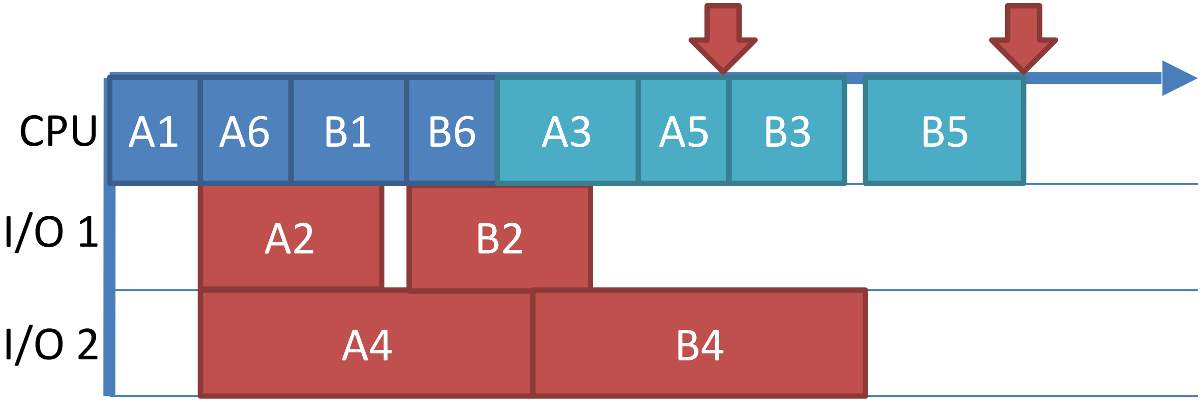


Single-Threaded Event Loop

```
// A.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```

```
// B.js
...      // 1
fs.readReadFile(..., () => {
  ...    // 3
});      // 2
http.get(..., () => {
  ...    // 5
});      // 4
...      // 6
```





Advantages

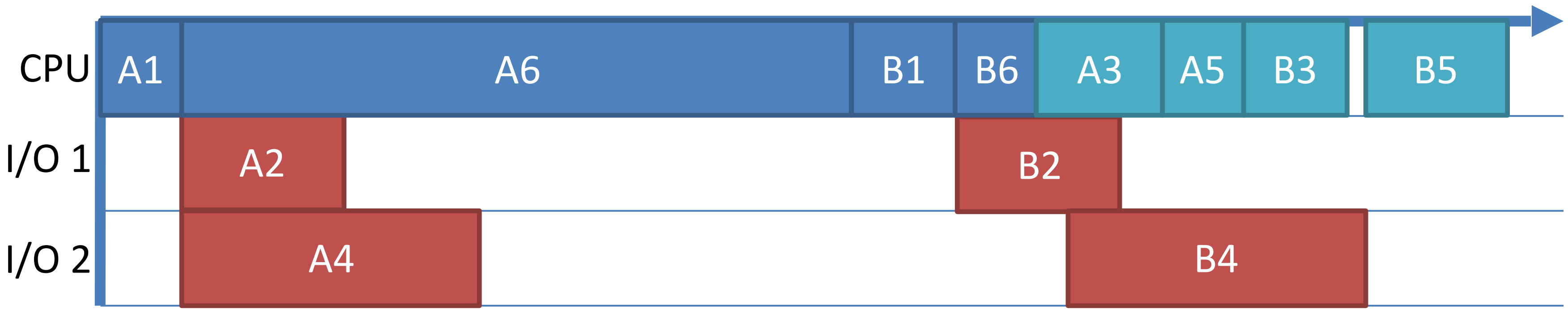
- Interleaved exec of multiple code blocks
- Short response time
- Pipelining between CPU and I/Os
  - High throughput
- Avoids overhead of concurrency control (e.g., locks, thread scheduling, etc.)
  - ***Higher throughput than muti-threaded engines***

Multi-Core Machines?

- Cluster module that runs one Node.js process per core
- Scales up ***linearly***, if
  - Workload can be partitioned evenly by processes
  - I/Os are not saturated

When ***Not*** to Use Node.js?

- CPU-bound tasks



- Or, use `child_process.exec()` (or `spawn()` for streaming output from child)
  - Foreground core + background cores

Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

NPM

- Package manager

```
$ npm init
$ npm install --[save|save-dev] <pkg-name>

var m = require('module');
```

- Packages vs. modules?

`--save` vs. `--save-dev`?

- Given dependency tree:

```
Your proj → Pkg 1
          → Pkg 2
Pkg 1 → Pkg 3
      → Pkg 4
Pkg 2 → Pkg 3
      → Pkg 5
```

- People who clone/fork your package will download the following packages:

```
{Pkg 1, Pkg 2, Pkg 3}  // via 'npm install'
```

Command Line Interface (CLI)

- Option 1: to parse `process.argv` yourself
- Option 2: `yargs.argv`

```
$ npm install --save yargs
```

  - Defines commands and their options
  - Help
  - Sanity checks

Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

Debugging

```
// in src
debugger;

$ node debug src/main.js <command> [options]
debug> n
debug> c
debug> repl
```

- Chrome inspector equivalent?

Node Inspector (Experimental)

```
$ node --inspect --debug-brk src/main.js ...
```

- Then paste “chrome-devtools://..” into Chrome

Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

Node.js as Backend Engine

- Typically, web workloads are I/O bound
  - High throughput
  - Low latency
- If designed well, API requests can be easily partitioned across multiple processes
  - Scales linearly

Clone `weathermood-server`

- Checkout the `file` branch

```
$ npm install --save express body-parser
$ npm install -save-dev nodemon
```

- [Express](#)
  - A web app framework based on Node.js
- [Body-Parser](#)
  - An Express middleware for parsing request body
- [Nodemon](#)
  - Auto-restarter (remember "`webpack -w`"?)

Web App Backend Development

1. Prepare static resources
   - E.g., *.html, *.css, client-side JS, images, etc.
   - In `dist/` of branch `server-file` of `weathermood`
2. Define API for AJAX calls
   - Dynamic resources, i.e., API for AJAX calls
3. Code API
4. Deploy web app to hosting server(s)
   - Web App ≠ web server(s)
   - For now, local machine as development server

Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

Defining APIs

- E.g., listing posts, create a post, vote, etc.
- But HTTP defines only 4 methods
  - GET, POST, PUT, DELETE
- Option 1: define new “verbs”
  - Always POST to the same URL
  - Define body (can follow SOAP)
- Option 2: define new “nouns”
  - E.g., vote → POST vote
  - Different URLs for different nouns/resources
  - REST makes your backend simple and scalable

URL Mappings

URLs\Methods	GET	POST	PUT	DELETE
http://${host}/${resource}s	List all resources (satisfying query "?…").	Create a new resource (unknown ID).	Replace the entire collection.	Delete the entire collection.
http://${host}/${resource}s/${id}	Read a specific resource.	Treat this resource as a collection and create a new member.	Update this resource or create one (known ID).	Delete this resource.

- Each resource type maps to 2 URL types
  - Collection URLs vs resource URLs

- List post: `GET /posts?seatchText=...&...`
- Create post: `POST /posts`
- Vote "clear": `POST /posts/${id}/clearVotes`

HTTP Response Codes

- GET:
  - 200 OK (with body)
- POST:
  - 200 OK (with body)
  - 201 Created (with header `Location` showing ID)
  - 204 No Content
- PUT and DELETE:
  - 200 OK (with body)
  - 204 No Content
- Error:
  - 400 Bad Request, 401 Unauthorized, 404 Not Found, or 500 Internal Server Error

Requirements

- ***Stateless***: session state (e.g., shopping cart) cannot be kept in client side
  - Session state sent with requests using [cookies]
  - So, requests from clients can be partitioned easily (scalable web servers)
- GET requests much have ***no side effect***
  - Allows proxy nodes in the routing path
- PUT and DELETE requests must be ***idempotent***: duplicated requests has no effect

Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

Express

- Normal flow: routers → model calls
  - Request parsing: `req.query`, `req.params`, `req.body`
  - Responses: `res.status()`, `res.json()`, `res.sendStatus()`
- Error flow: `throw` → error handler middleware → responses

Middleware

- Error handling
- Serving static files
- Filtering
- Logging
- Validation
- Authentication and authorization
- and more...

Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

Debugging

- Nodemon:

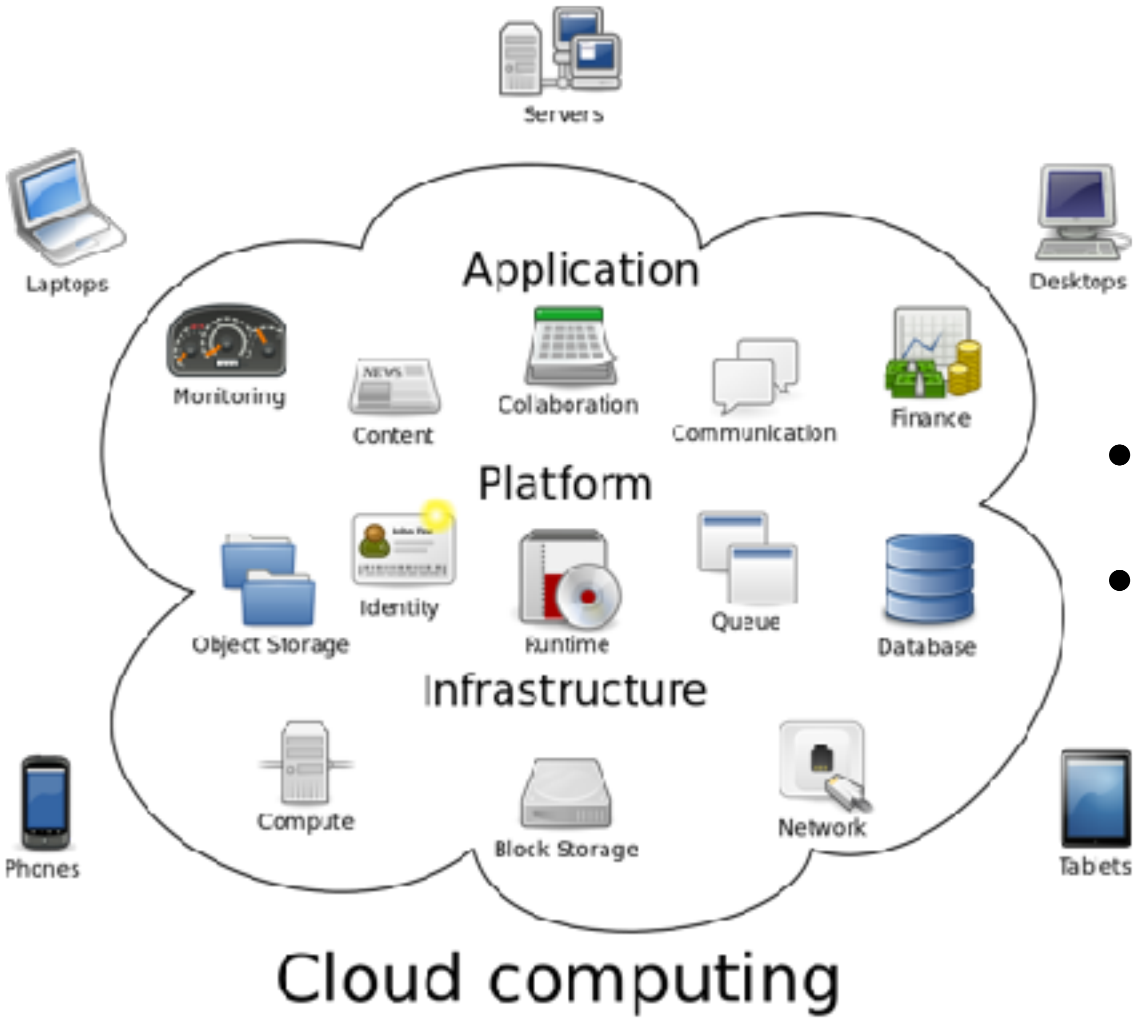
```
$ npm run watch
```

- Postman

Outline

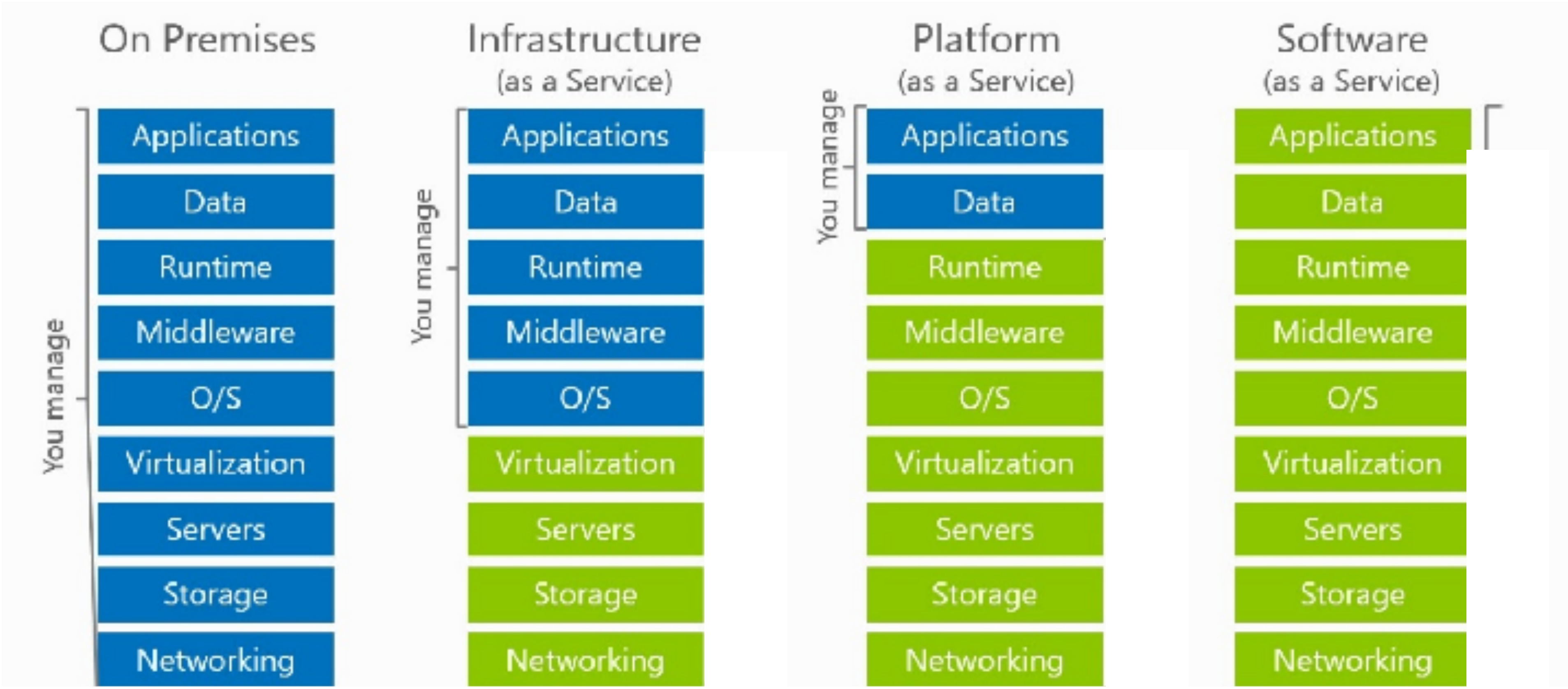
- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

Cloud Computing



- Not only services
- Also platforms or infrastructure
  - Pay only what you used

IaaS, PaaS, and SaaS



Web App Backend: PaaS or IaaS?

Flexibility?

Cost?

Performance?

Ease of Migration?

Development Efficiency

- Iterations: bug fixes, updates, finding PMF, etc.
- Time is money!

- IaaS: high management cost, low ***elastisity***
- PaaS: low cost and elastic, but

“Well, it works on my computer…”

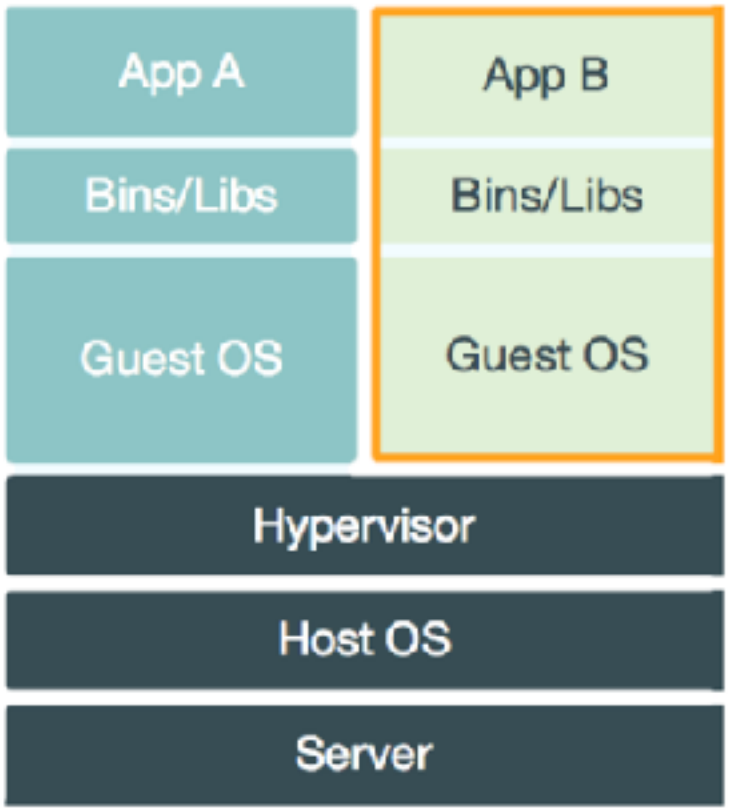
Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk

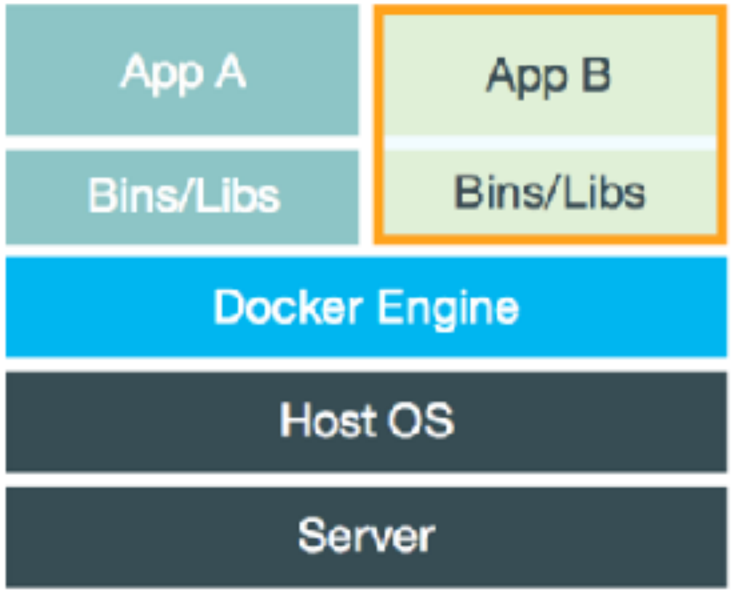
docker

- PaaS
- You define your runtime

Virtual Machines vs. ***Containers***

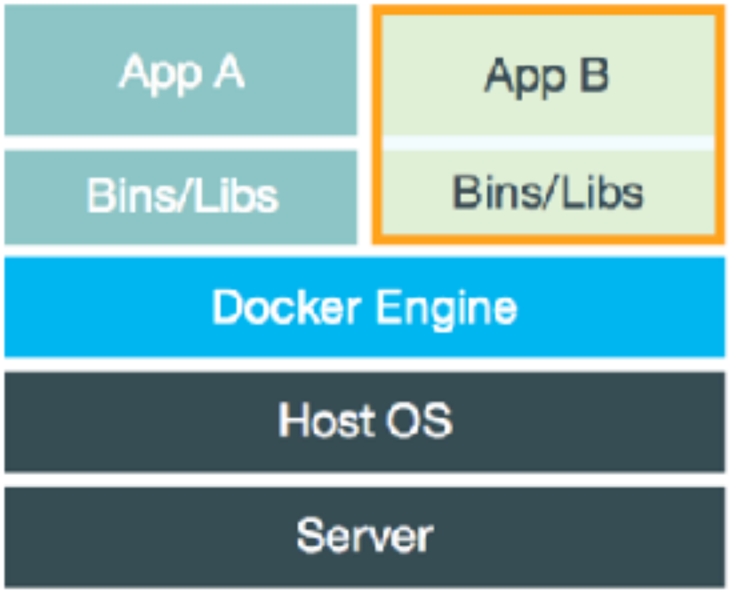


- Lightweight, elastic
- Consistent runtime



“Dockerizing” an App

1. Build a Docker Image
2. Upload the image to targeted server(s)
3. Launch ***container*** from image



Example: Local Development Server

- Install Docker Community Edition first

```
// in project folder
$ vim Dockerfile
$ vim .dockerignore

$ docker build -t <name:tag> .
$ docker images

$ docker run -p 80:8080 -d <image>
$ docker <stop|restart> <container>
$ docker ps [-a]
$ docker system prune
```

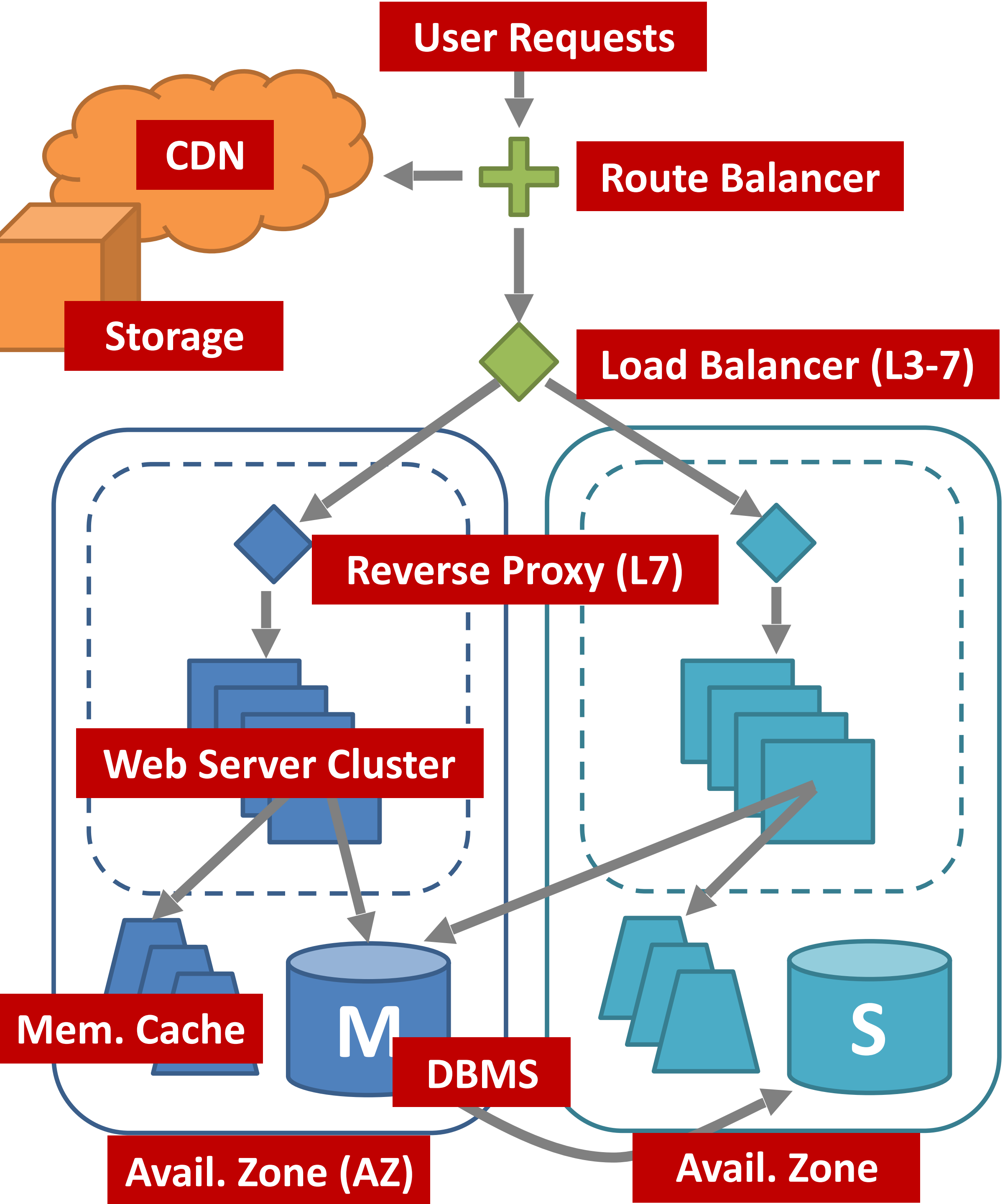
Watch out Data Loss!

- Modifications to filesystem are ***local*** to a container
  - Gone if container deleted
- If necessary, use data volumes to persists data across containers
  - Basically, specially-designated directories

Beyond development server?

Outline

- Node.js
  - Events and asynchronous I/O
  - NPM and CLI tools
  - Debugging
- Backend Development using Express
  - RESTful API
  - Express: routers and middleware
  - Testing and debugging with Postman
- Deployment
  - Cloud computing and Docker
  - Amazon Elastic Beanstalk



"Typical" Environment

- DDoS protection
- Static vs. dynamic resources
- High availability
- LB, caching, acceleration (SSL), etc.
- Auto-scaling groups

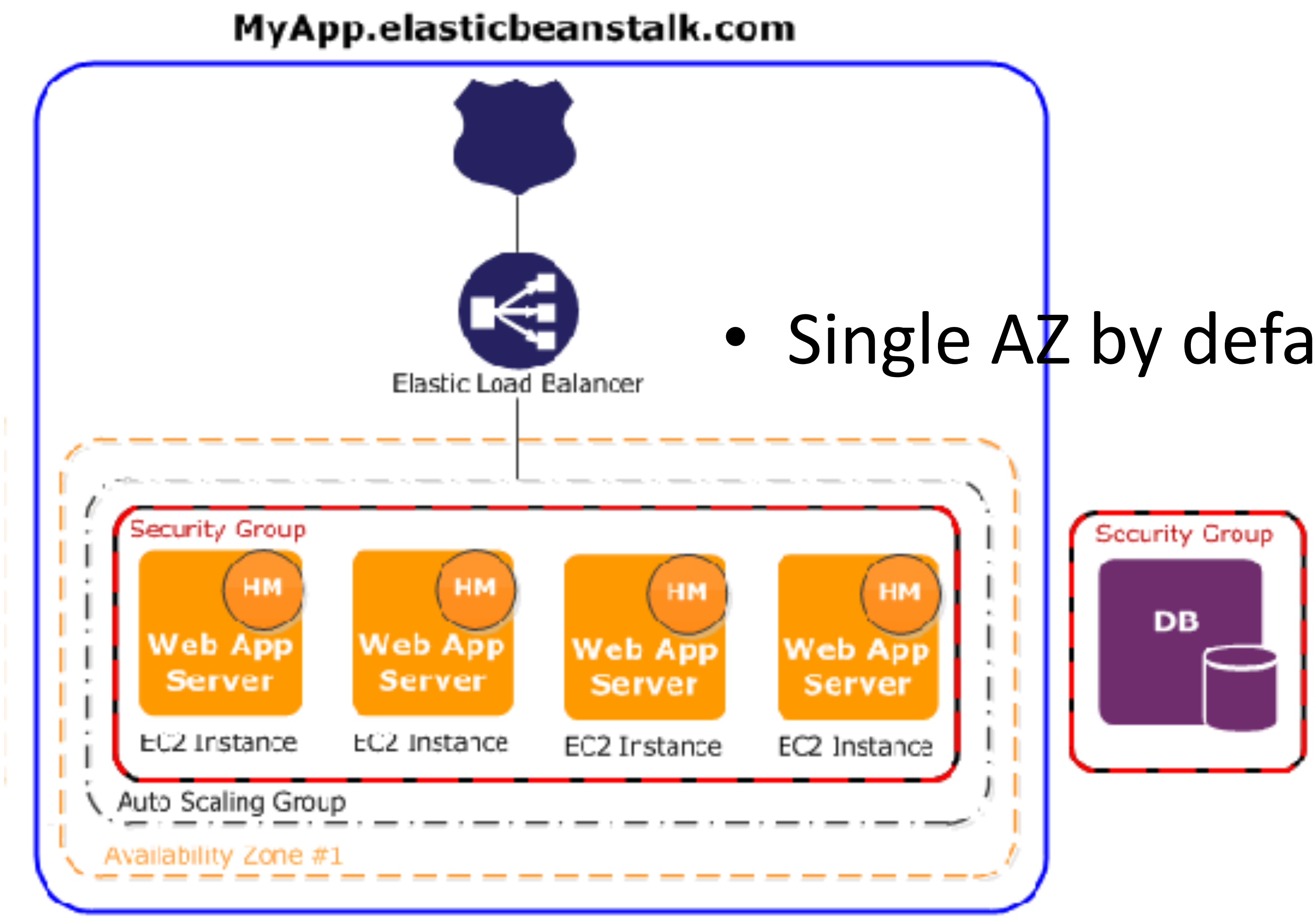
- Scale ***up*** vs. ***out***
- Slave DBMS can serve read-only requests

Solutions

Node	Amazon (AWS)	Alternatives
Route Balancer	***Route 53***	Cloudflare
CDN	***CloudFront***	Cloudflare, Google Cloud CDN
Storage	Simple Storage Service (***S3***)	HDFS
Load Balancer	Elastic Load Balancing (***ELB***)	Nginx
Reverse Proxy		***Nginx***
Web Server	Elastic Compute Cloud (***EC2***)	Heroku
Mem. Cache	***ElastiCache***	Redis, Memcached
DBMS	Relational DB Service (***RDS***)	PostgreSQL, MySQL, MongoDB

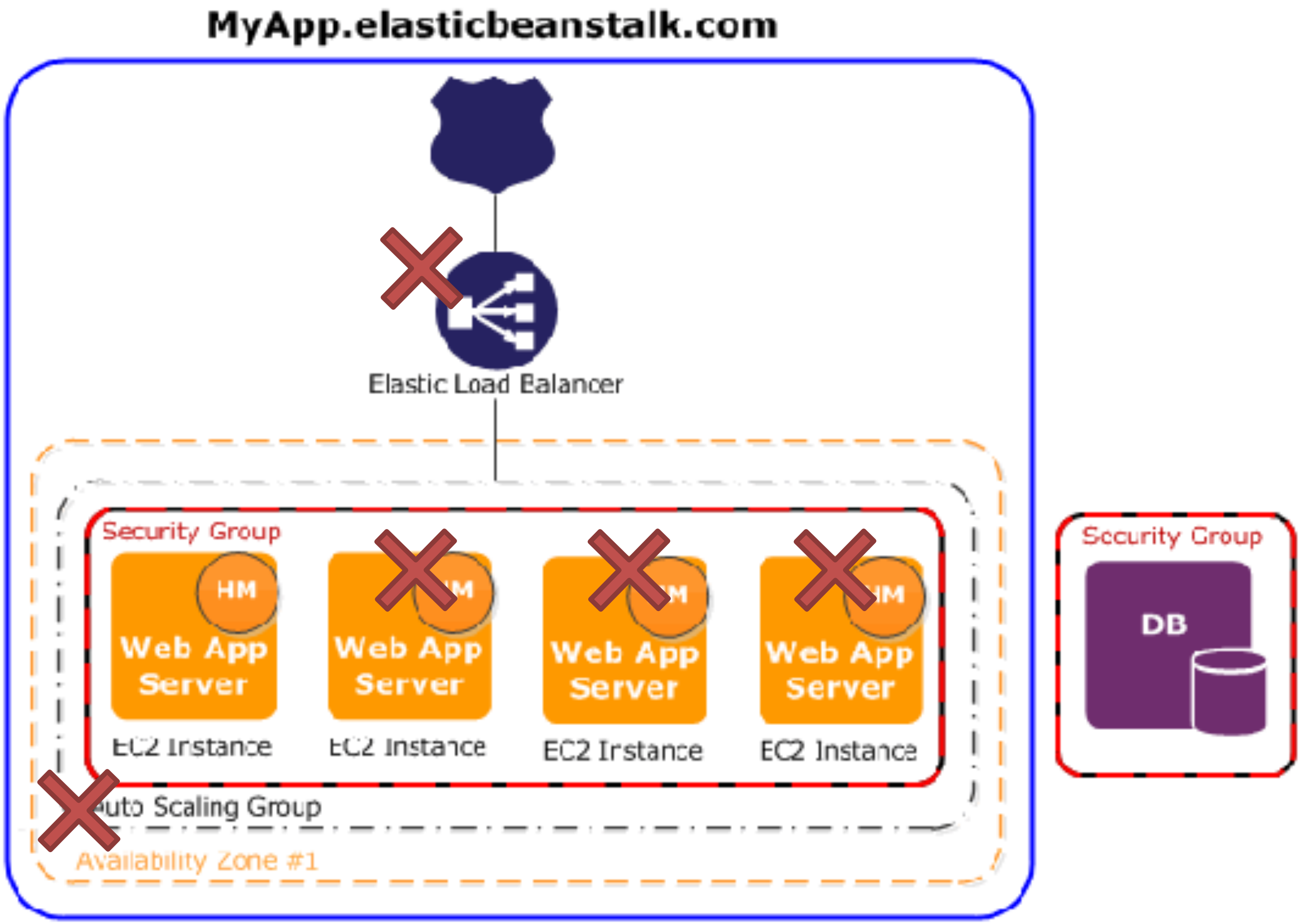
- Web app on EC2 (with Docker and Nginx preinstalled)
- But what about the rests?

AWS Elastic Beanstalk



- Single AZ by default

Staging Environment



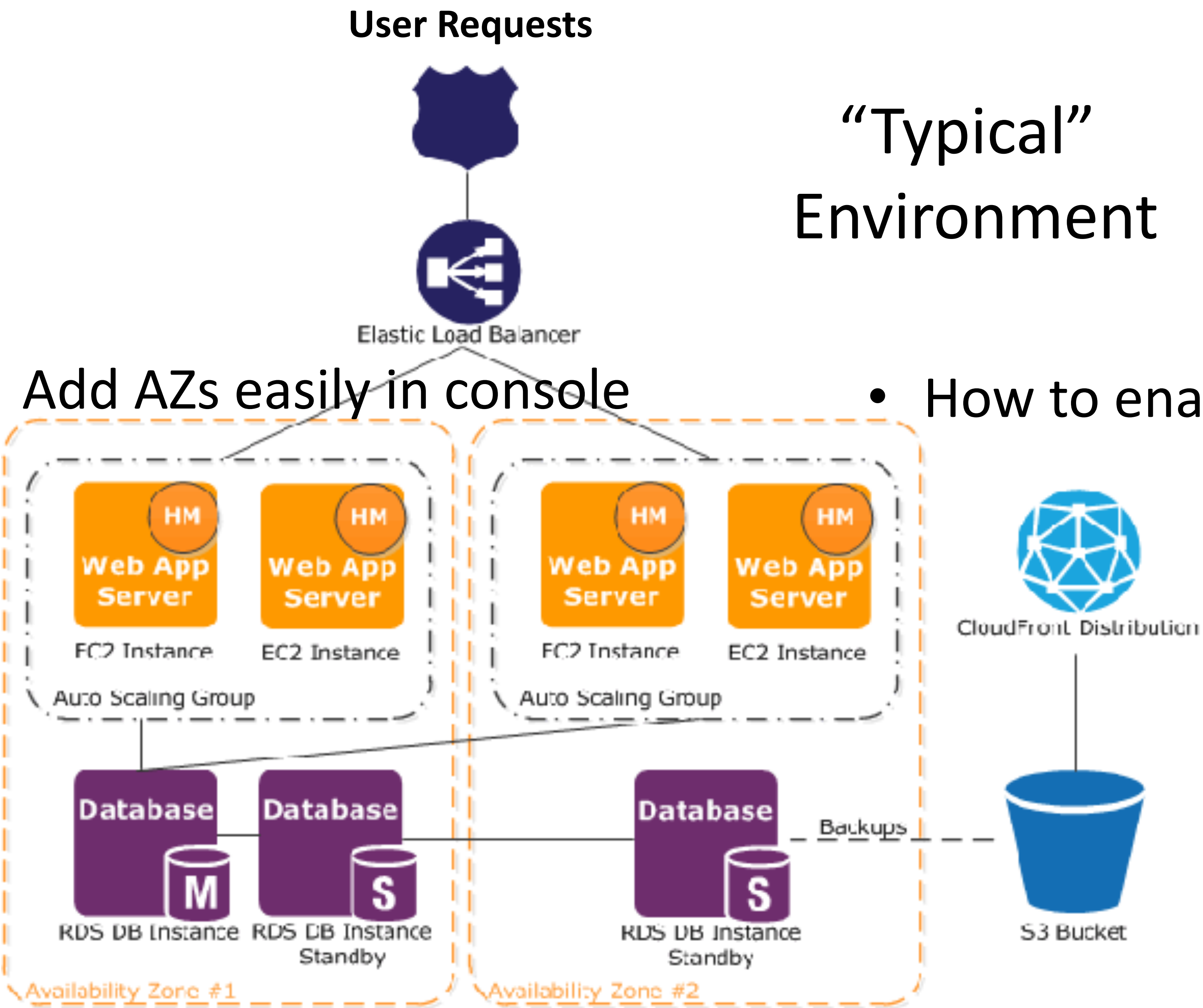
Instructions

1. Create an AWS account
   - Free-tier for the first year, and free credits from AWS Educate
   - ***Credit card needed*** (one per group); no immediate charge
2. Install Elastic Beanstalk CLI
   - Install Python runtime (e.g., Aanaconda) first

```
$ pip install --upgrade awsebcli
$ eb –version

// in project folder
$ eb init  // create eb application
$ eb create [--single] // create environment
$ eb terminate <env>
// upates
$ git commit
$ eb deploy <env>  // deploys the latest commit
$ eb use <env> // env for current branch
```

“Typical” Environment



- Add AZs easily in console
- How to enable CD

Cache Control

- CloudFront and client browsers use `Cache-Control` response header to cache static files

```
// in server.js
app.use(express.static('dist', {
  maxAge: ... // in ms. For client browsers and proxies
}));
```

- Be careful about file versioning

```
// or
app.use(express.static('dist', {
  setHeaders: (res, path, stat) => {
    // in seconds. For proxies only
    res.set('Cache-Control', 'public, s-maxage=...');
  }
}));
```

Readings (Optional)

- HTTPS
  - Production environments
  - Staging environments
- Custom domain names

Worker Environments

- For CPU-bound tasks (e.g., data analytics)