JavaScript
fwdays

Node.js Middleware Never again!

Timur Shemsedinov

Chief Technology Architect at Metarhia
Lecturer at Kiev Polytechnic Institute

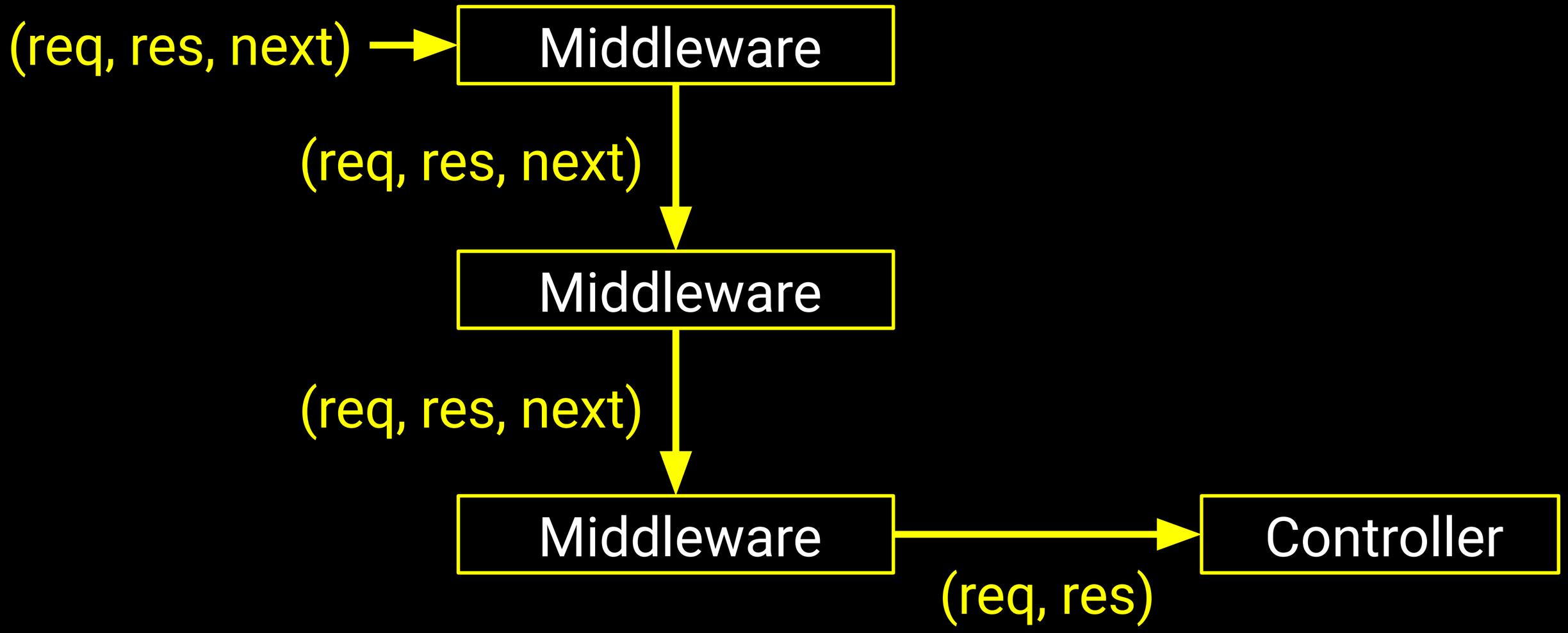
What is middleware?

- Connect contract
  (req: Request, res: Response, next: NextFunction)
  (req: Request, res: Response)

- Koa contract
  async (ctx: Context, next: NextFunction): Promise
  async (ctx: Context): Promise

Middlewares and Controller

JavaScript
fwdays

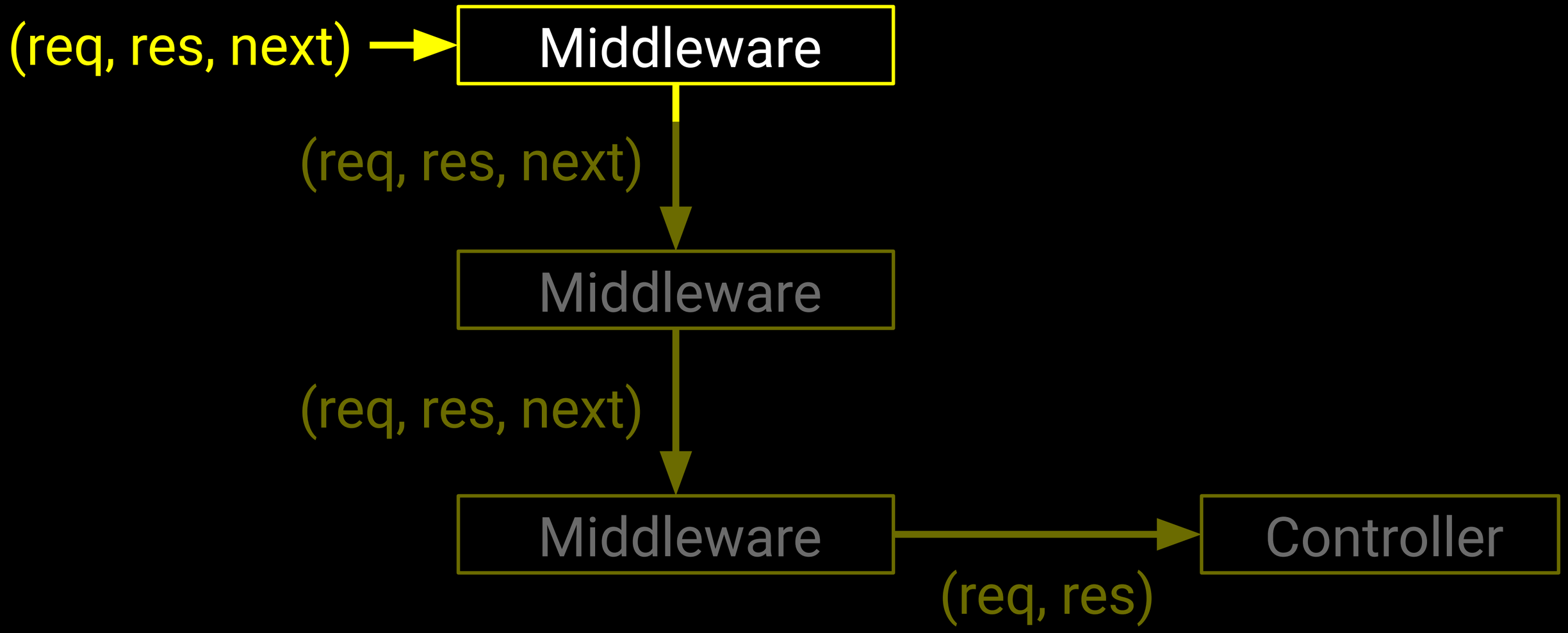


What is middleware?

- Mixin provocation
- Reference pollution and shared state provocation
- Race condition provocation
- Abstraction leak provocation
- Fat controller and layers mix provocation
- High coupling provocation
- Error ignoring provocation

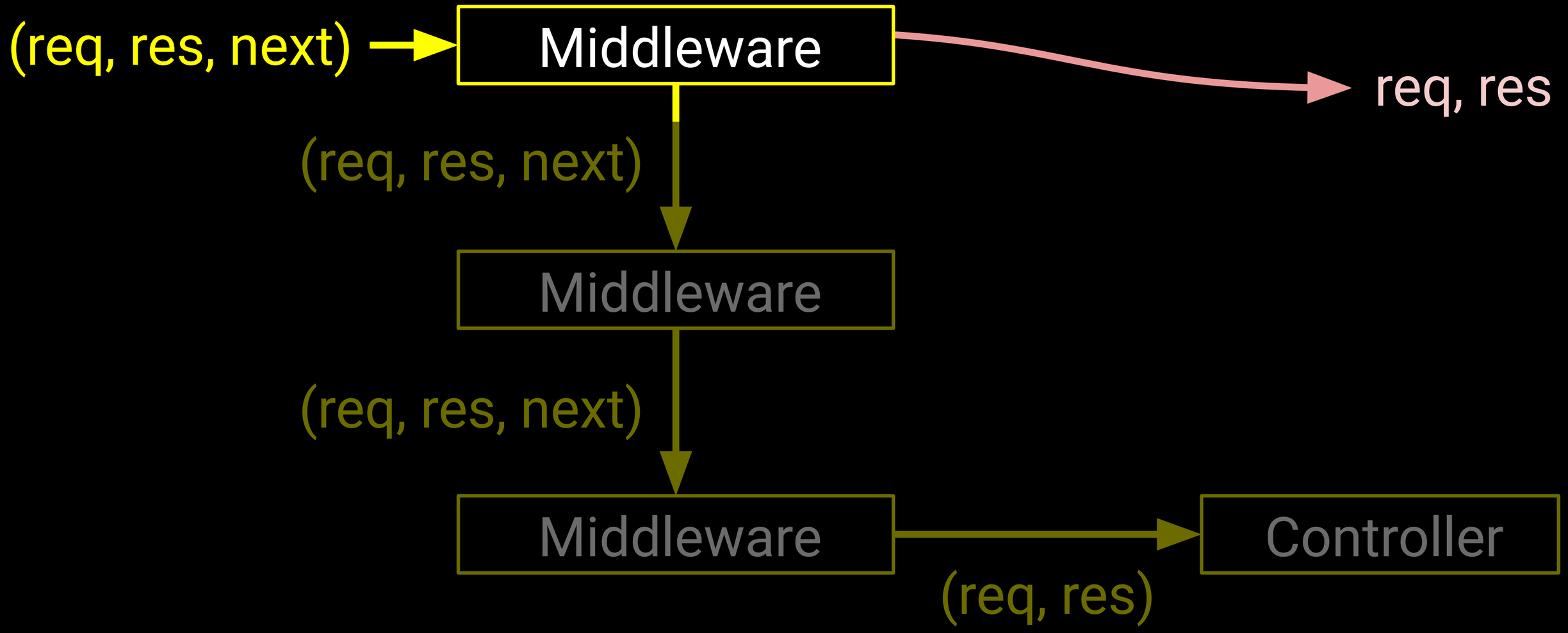
Middleware: Reference pollution

JavaScript
fwdays

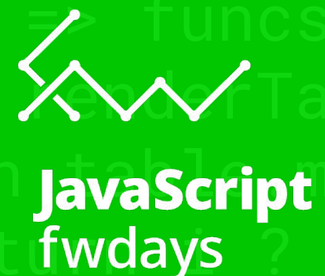


Middleware: Reference pollution

JavaScript fwdays



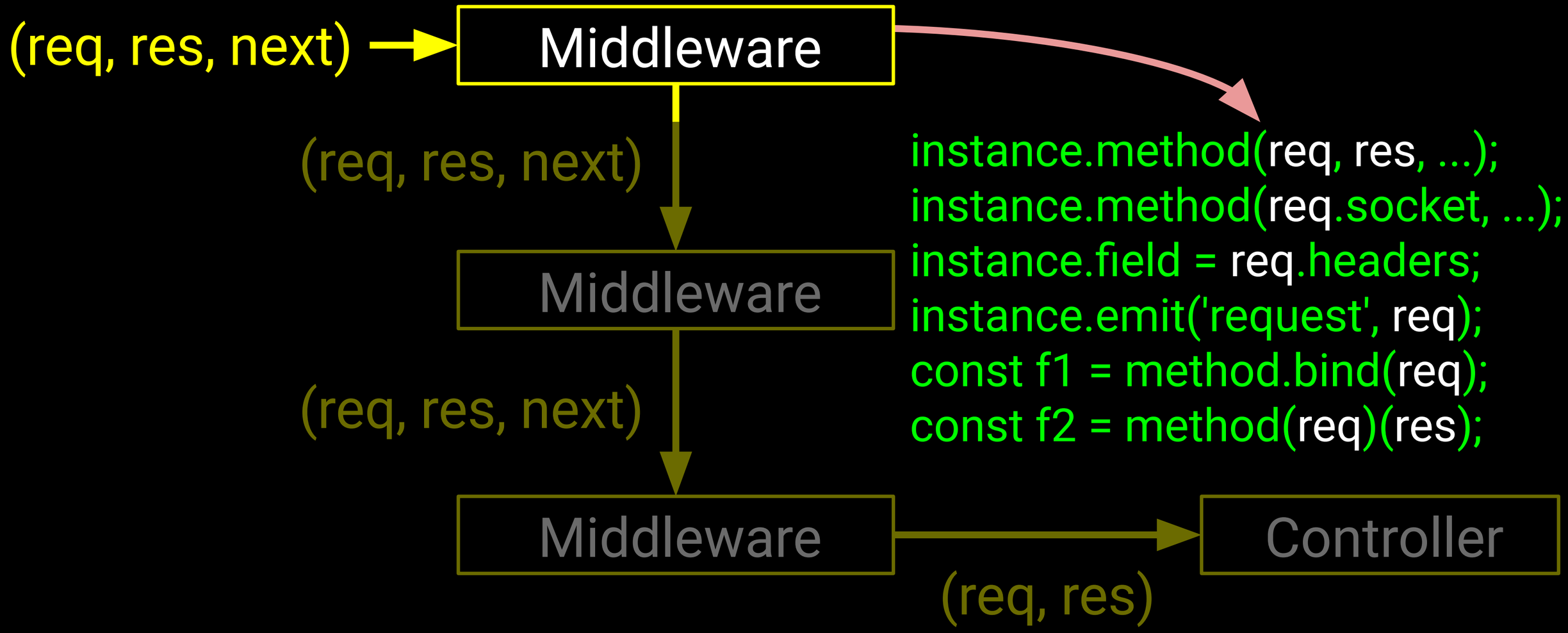
Middleware: Pass ref outer




```
const sockets = new Map();

app.use((req, res, next) => {
  ...
  sockets.set(userId, req.socket);
  ...
  next();
});

for (const socket of sockets) {
  doSomethingWithSocket(socket);
}
```

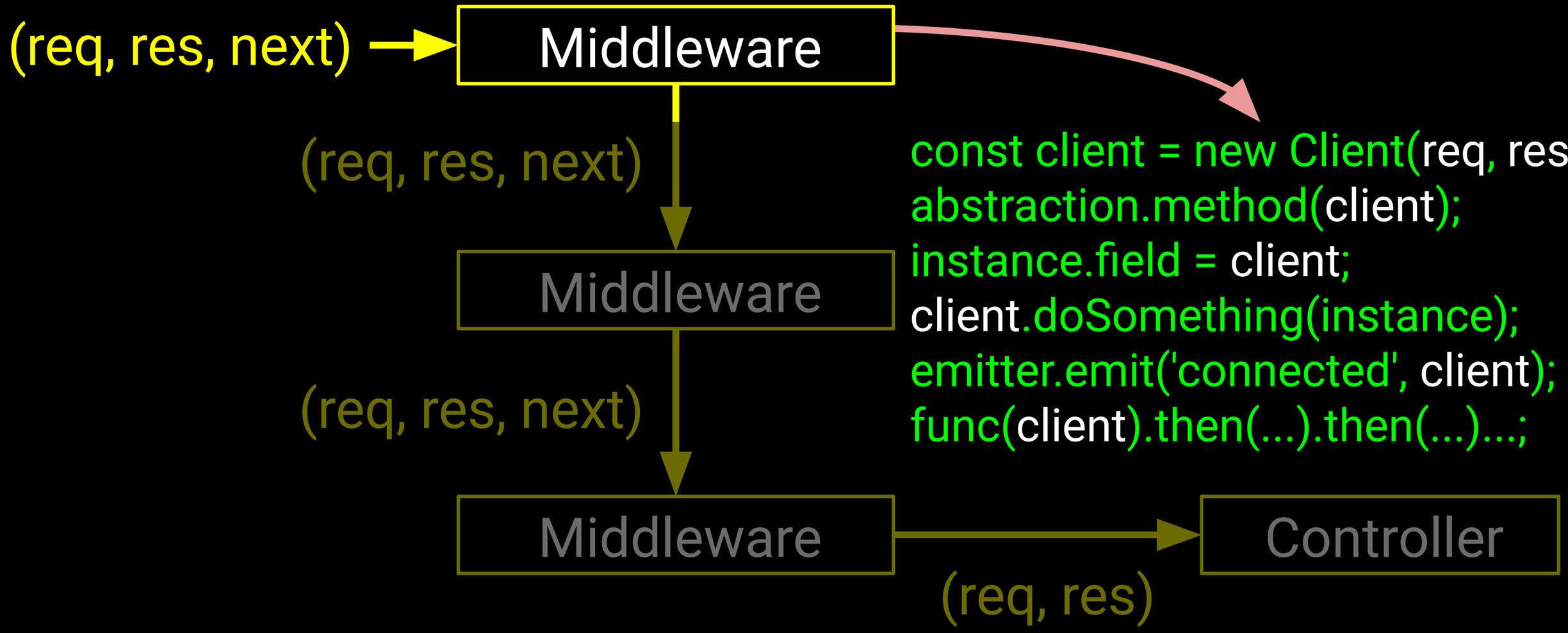
Middleware: Reference pollution

JavaScript
fwdays



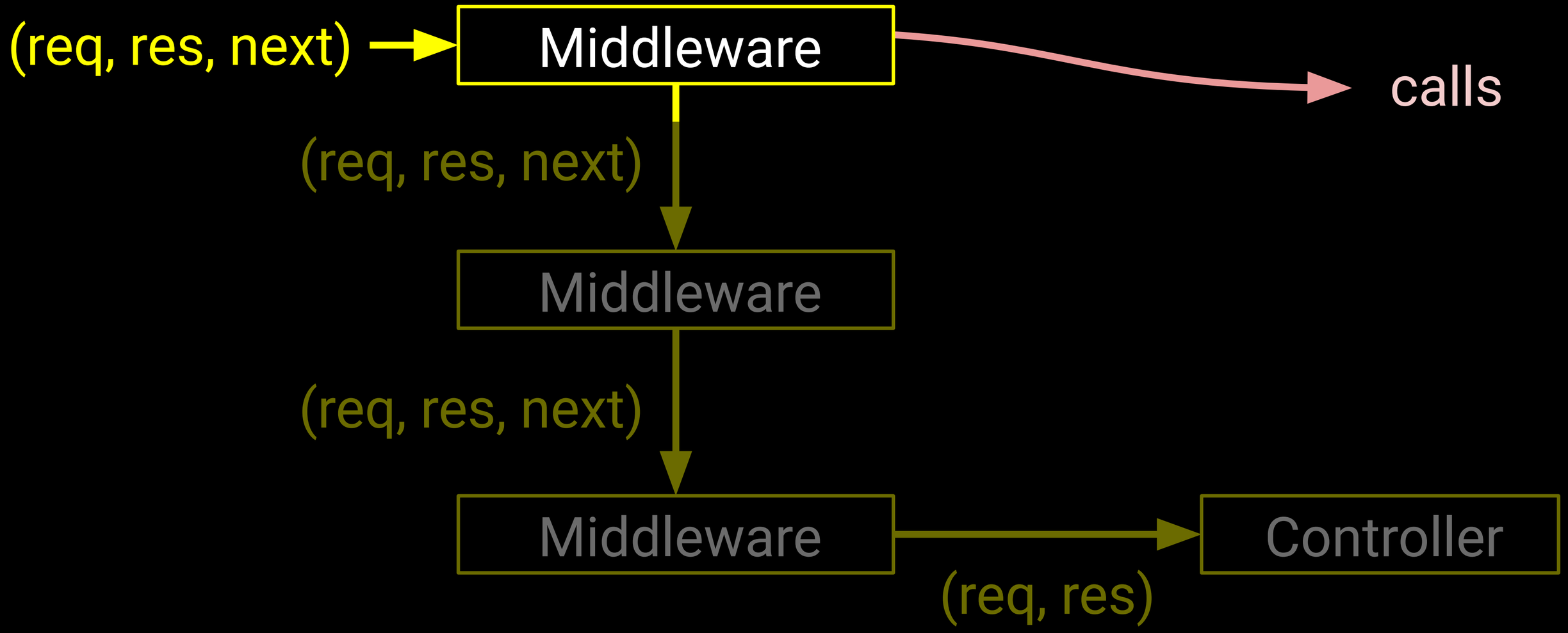
Middleware: Reference pollution

JavaScript fwdays



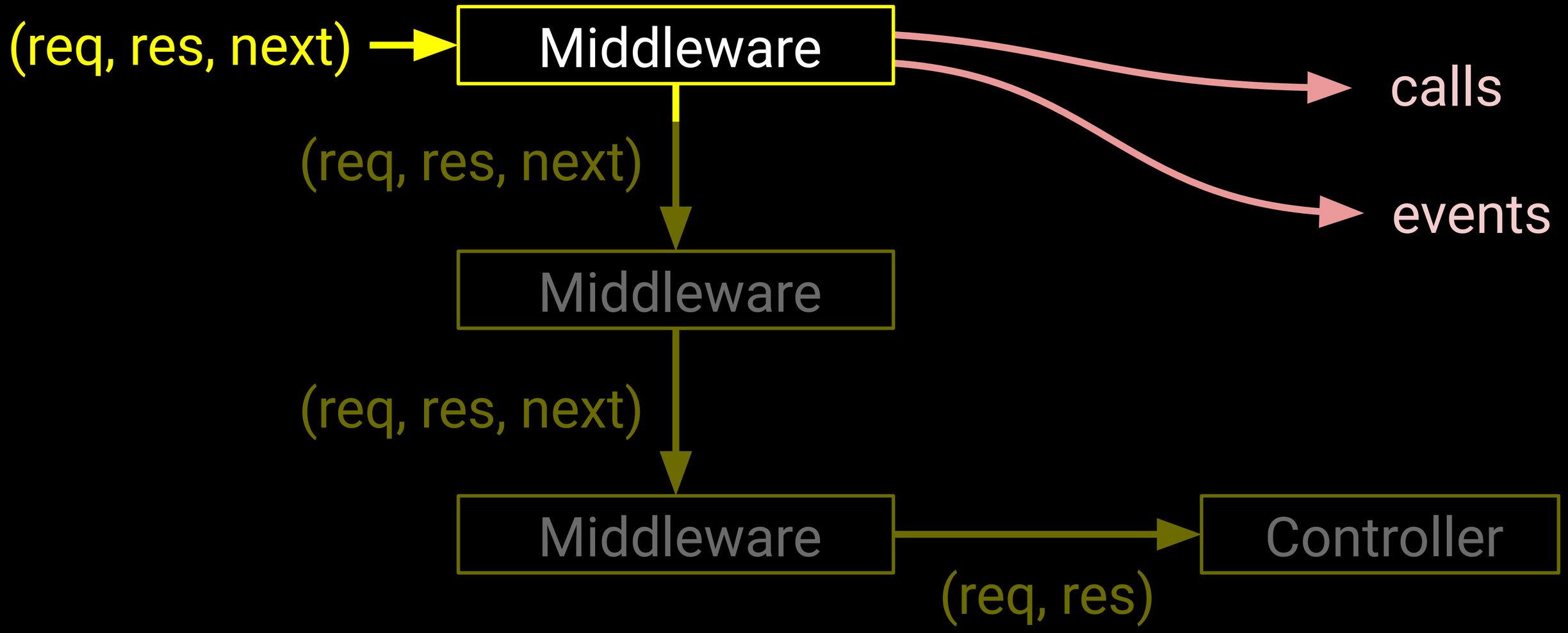
Middleware: Reference pollution

JavaScript fwdays



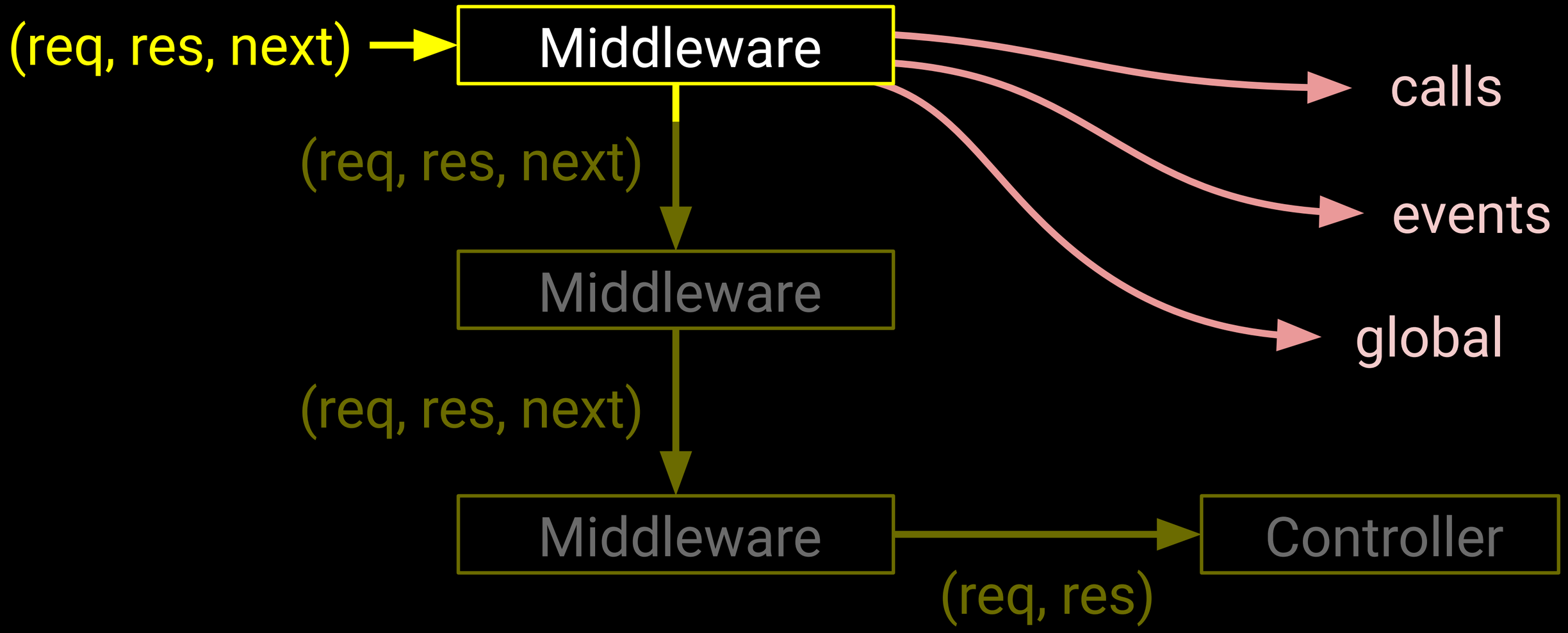
Middleware: Reference pollution

JavaScript fwdays



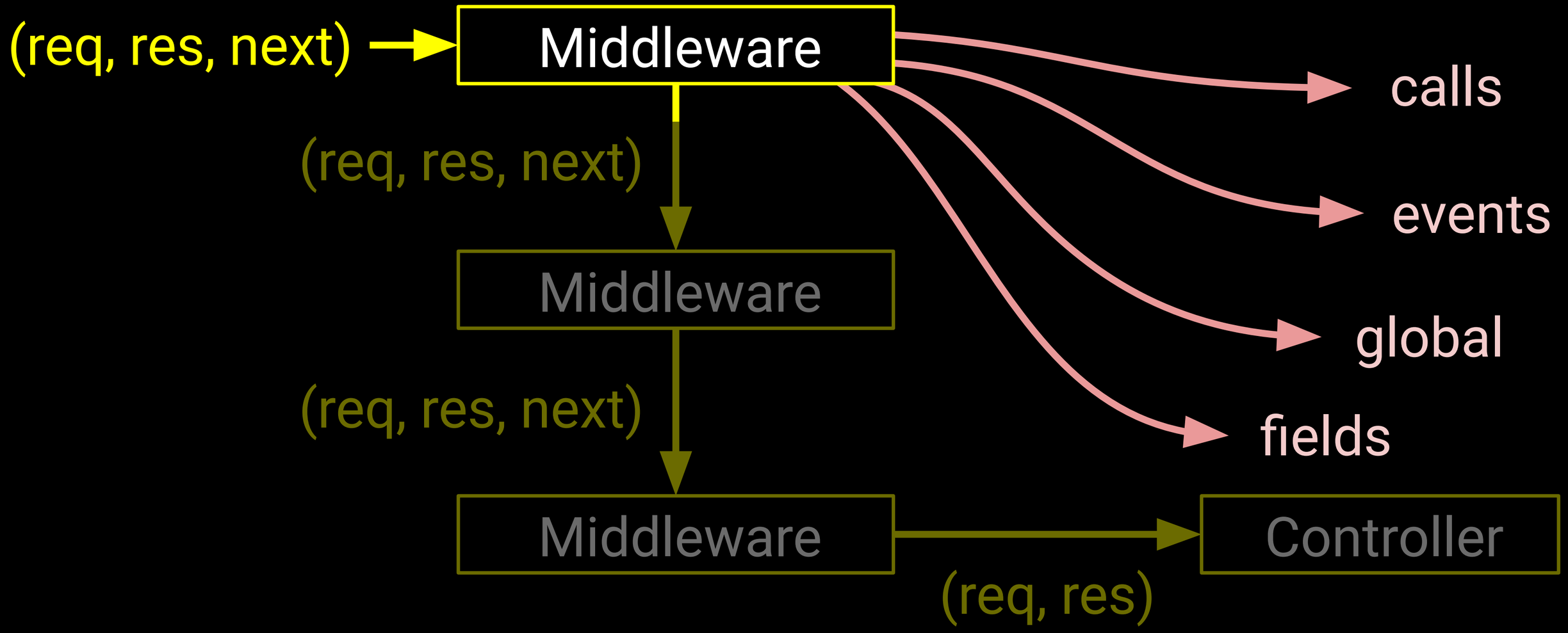
Middleware: Reference pollution

JavaScript fwdays



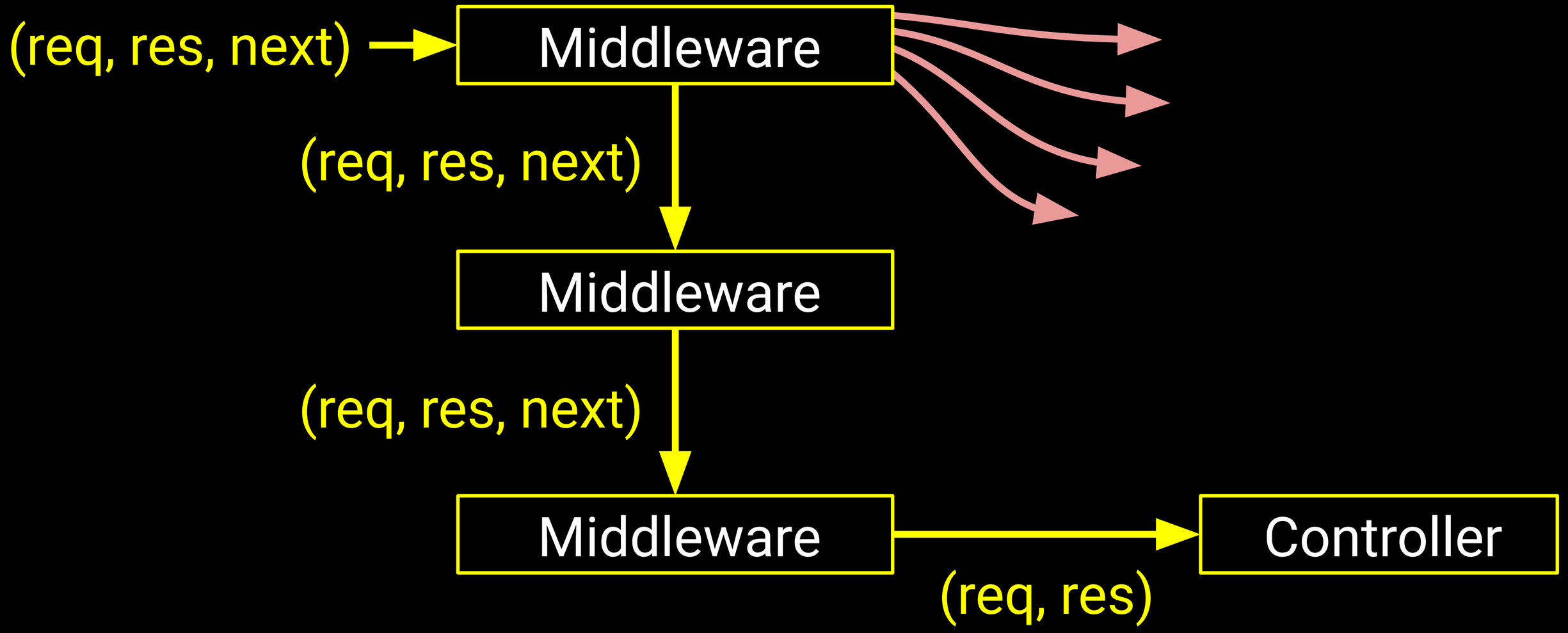
Middleware: Reference pollution

JavaScript fwdays



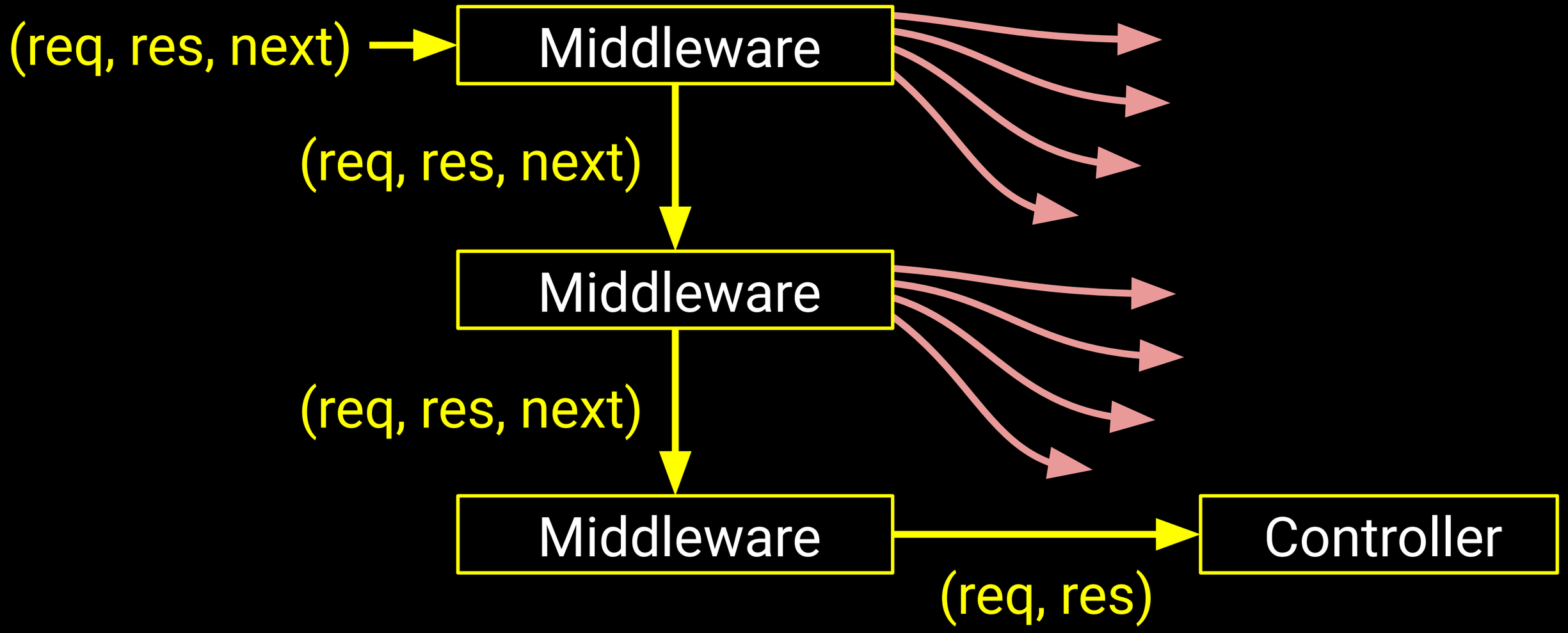
Middleware: Reference pollution

JavaScript fwdays



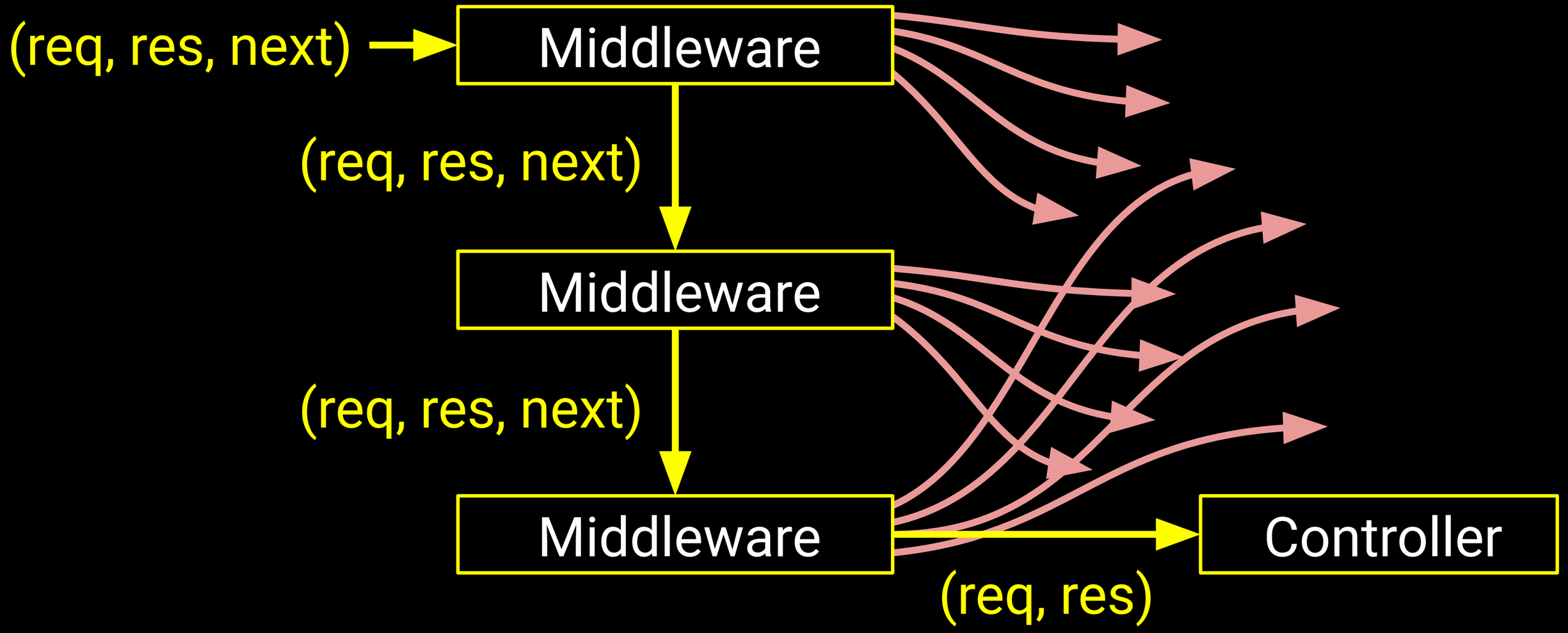
Middleware: Reference pollution

JavaScript fwdays



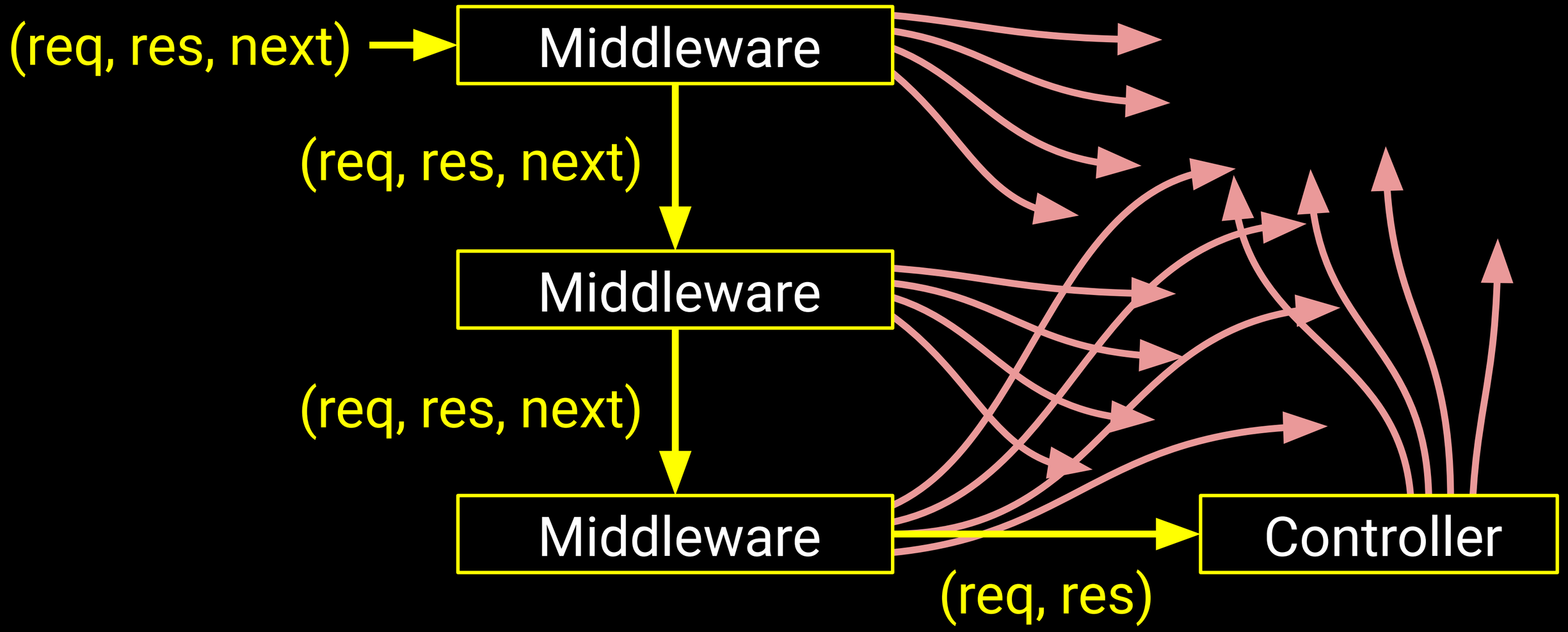
Middleware: Reference pollution

JavaScript fwdays



Middleware: Reference pollution

JavaScript fwdays

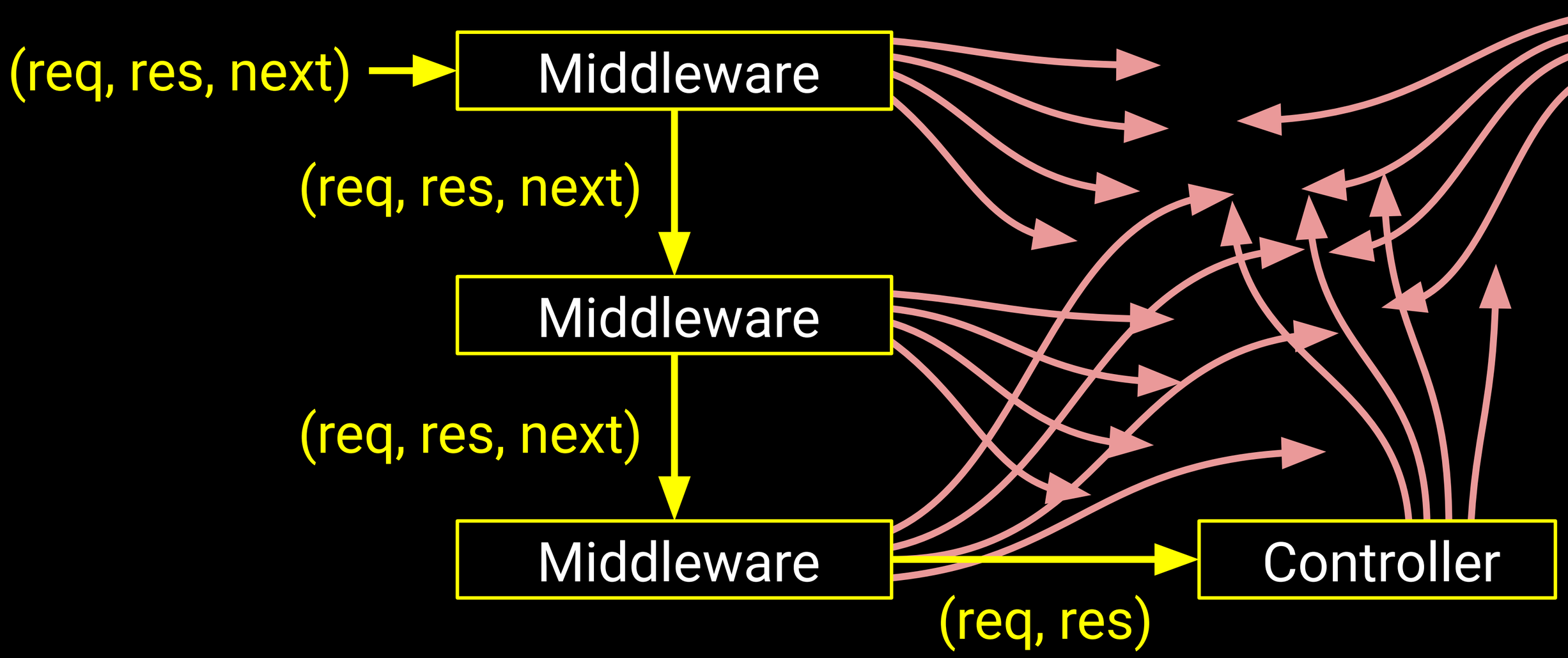


Middleware: Pass ref everywhere

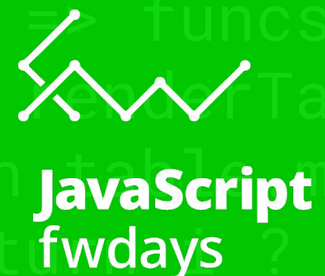

```
const ee = new EventEmitter();

app.use((req, res, next) => {
  ee.emit('timeout', res);
  next();
});

ee.on('timeout', res) => {
  setTimeout(() => {
    if (!res.writableEnded) res.end('timeout');
  }, 5000);
});
```

Middleware: Reference pollution

JavaScript fwdays



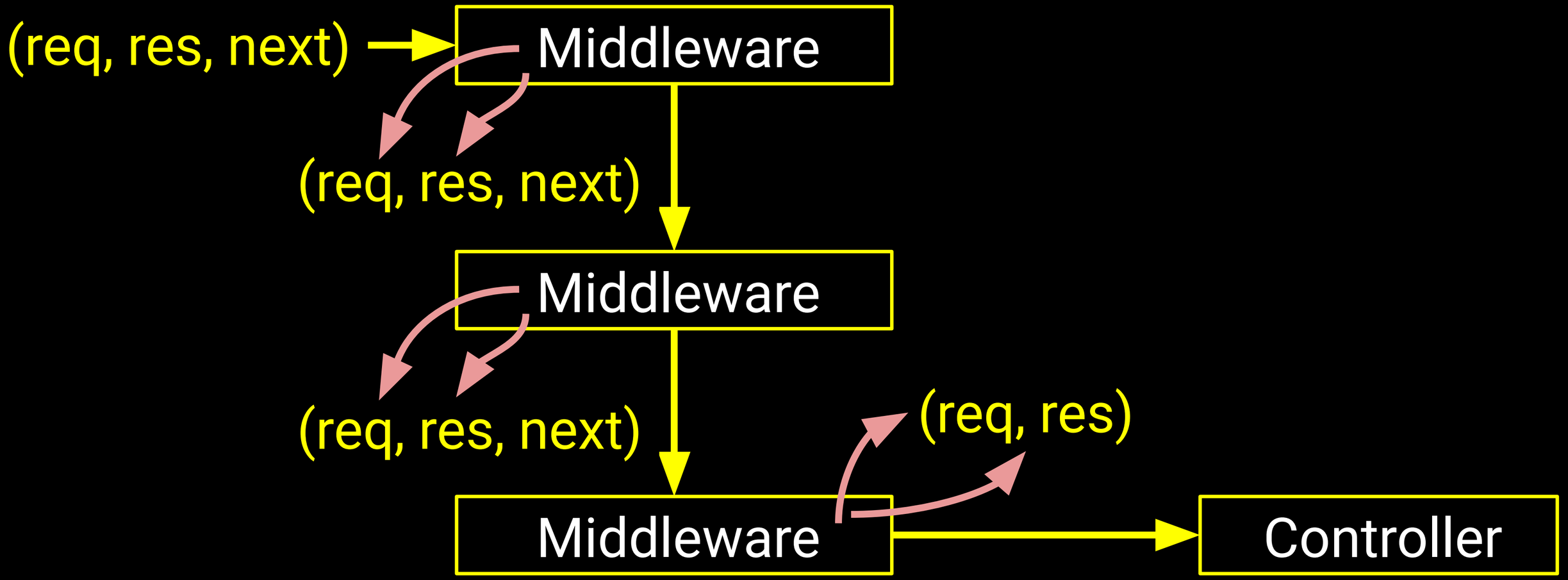
Middleware: Race, data corruption




```
let userId;

app.use((req, res, next) => {
  userId = ...;
  next();
});

app.get('/resource', (req, res) => {
  if (checkAccess('/resource', userId) === GRANTED) {
    res.end('You have an access');
  }
});
```

Middleware: Mixin pollutions

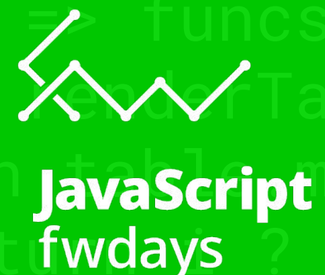
JavaScript fwdays



Middleware: mixin to req, res, ctx


```
app.use((req, res, next) => {
  res.groupName = 'idiots';
  next();
});

app.get('/user', (req, res) => {
  if (res.groupName === 'idiots') {
    res.end('Welcome, my dear friend!');
  }
});
```

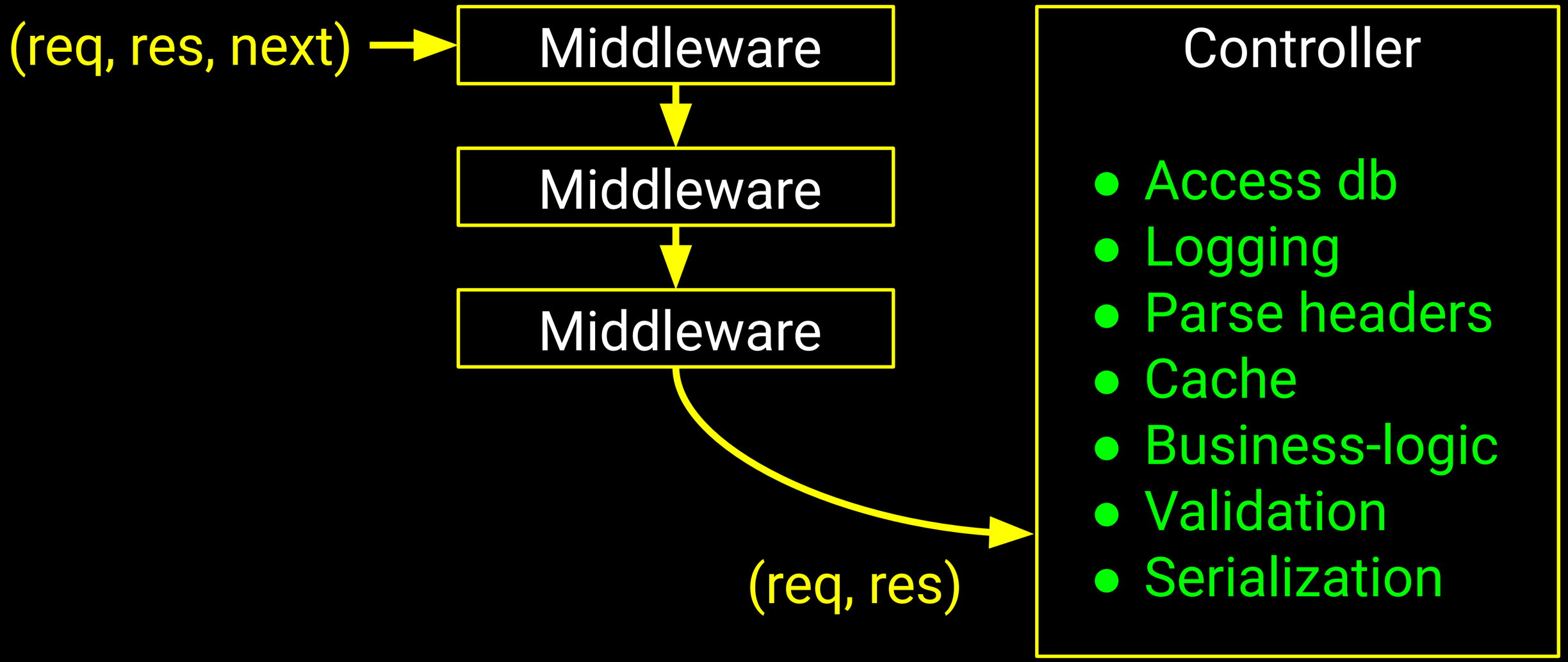
Middleware: mixin to req, res, ctx




```
app.use((req, res, next) => {
  res.locals.groupName = 'idiots';
  next();
});

app.get('/user', (req, res) => {
  if (res.locals.groupName === 'idiots') {
    res.end('Welcome, my dear friend!');
  }
});
```

Middleware: Fat controller

JavaScript fwdays



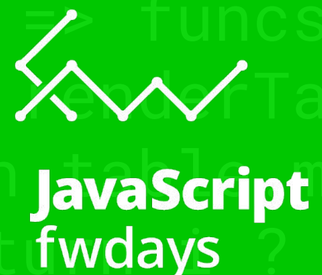
Don’t mix in a single function

- Data access (database connection)
- Business-logic and domain model
- Routing, logging, configuration
- Health and server state reporting
- Working with sockets, headers, cookies, etc.
- Serialization and deserialization
- Templating, caching, cryptography, sessions

Middleware: Fat controller


```
router.get('/user/:id', (req, res) => {
  if (blacklist.has(res.socket.remoteAddress)) {...}
  const id = parseInt(req.params.id);
  if (!isValidateUserId(id)) return res.status(500);
  const query = 'SELECT * FROM users WHERE id = $1';
  pool.query(query, [id], (err, data) => {
    if (err) {...}
    logger.write(`access user: ${id}`);
    res.status(200).json(data.rows);
  });
});
```

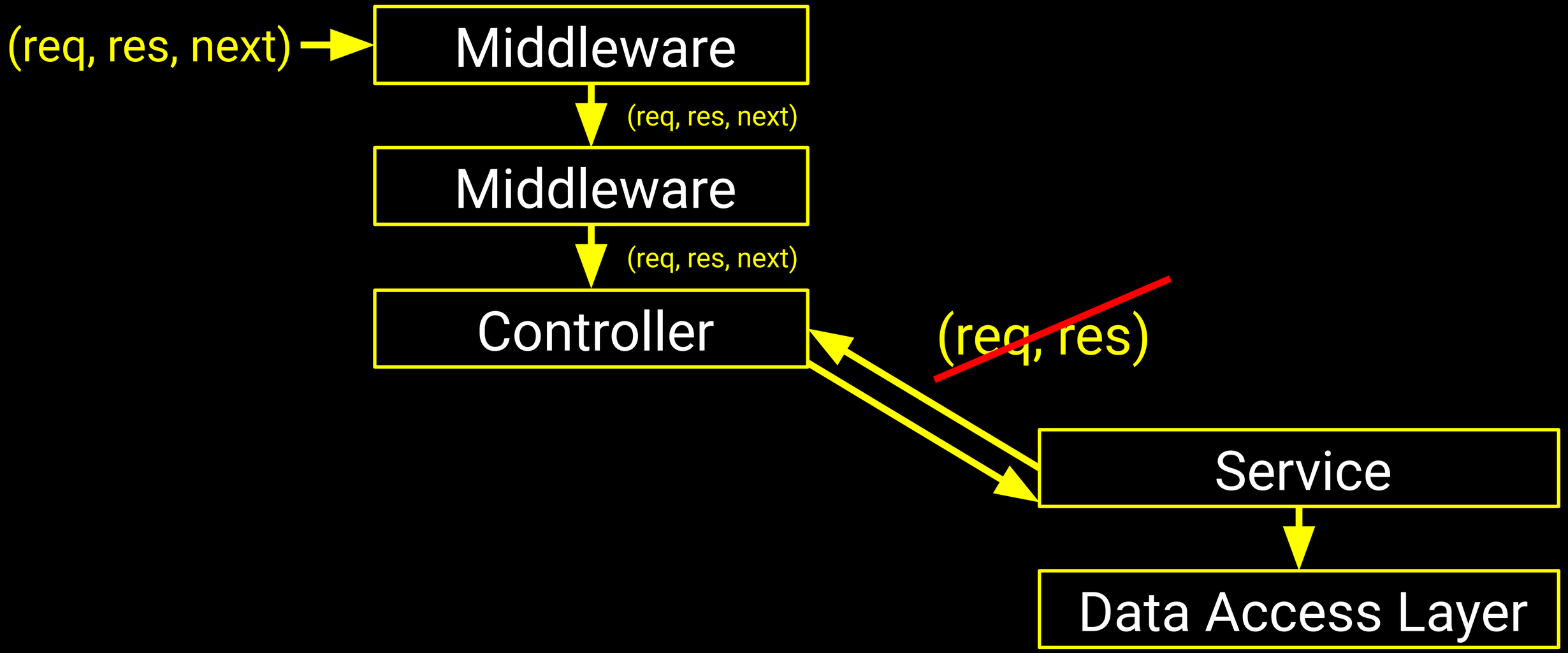
Middleware provocates antipatterns

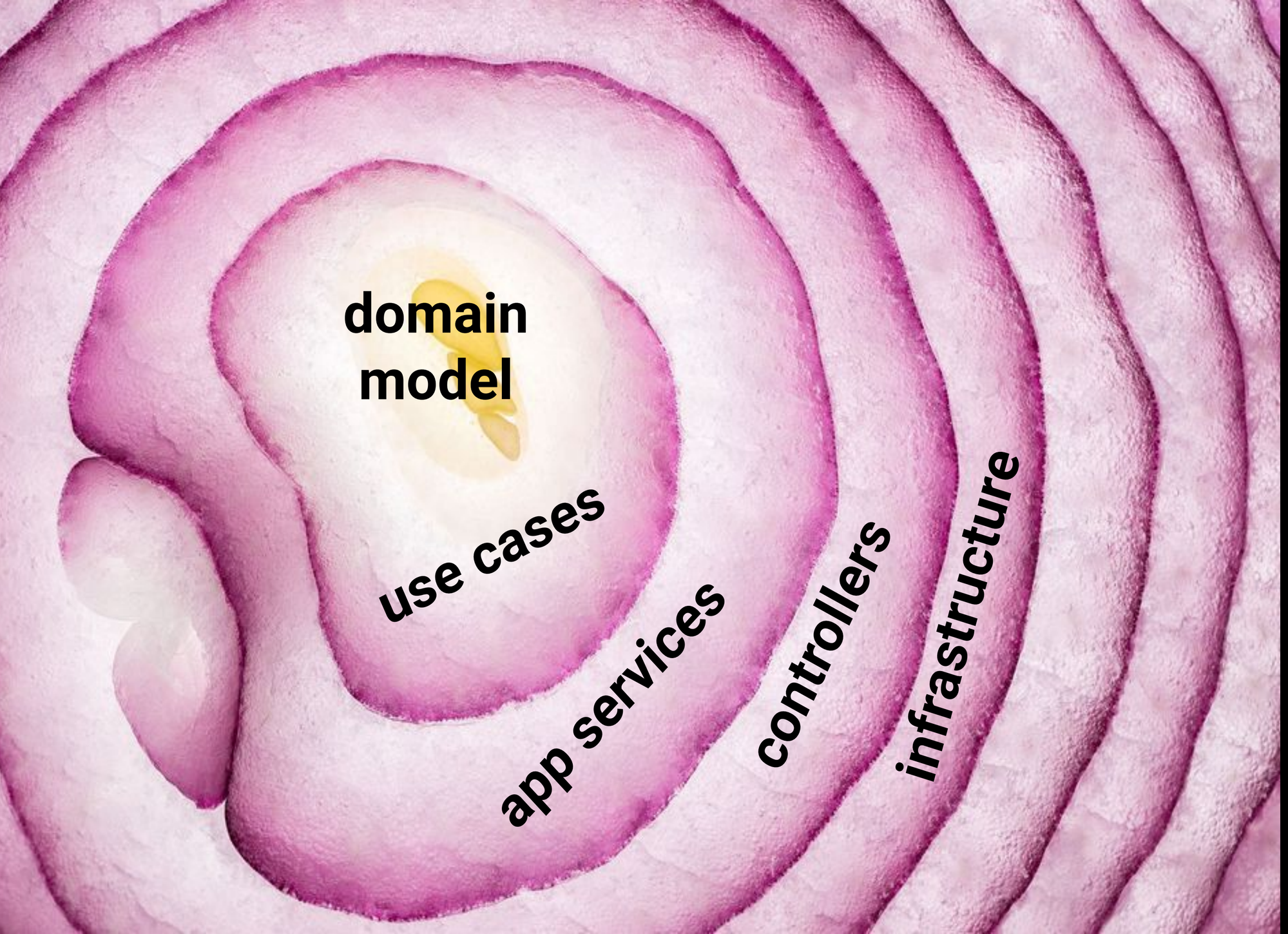


- Pass-through parameters, Too many parameters, Data clump
- Accumulate and fire
- Temporary field, State mixins, Shared state, Global state, State in outer contexts
- Inappropriate intimacy
- High coupling (also see high cohesion pattern)

Layered (onion) architecture

JavaScript fwdays





JavaScript
fwdays

We need patterns and principles

- DIP (dependency inversion principle)
  - IoC (Inversion of control)
  - DI (dependency injection)
- Chain of responsibility (GoF)
- Law of demeter
- SRP (Single responsibility, SOLID)
- Low coupling (GRASP)

Keep attention on

- Domain in the middle
- Context isolation
- Layered (onion) architecture
- Don’t depend on frameworks
- Don’t move logic between model and controller
- Always work on abstraction leaking
- Protect data with parallel primitives

github.com/tshemsedinov
youtube.com/TimurShemsedinov
github.com/HowProgrammingWorks/Index

Весь курс по ноде и JS (186 лекций)
https://habr.com/ru/post/485294/

t.me/HowProgrammingWorks
t.me/NodeUA

timur.shemsedinov@gmail.com