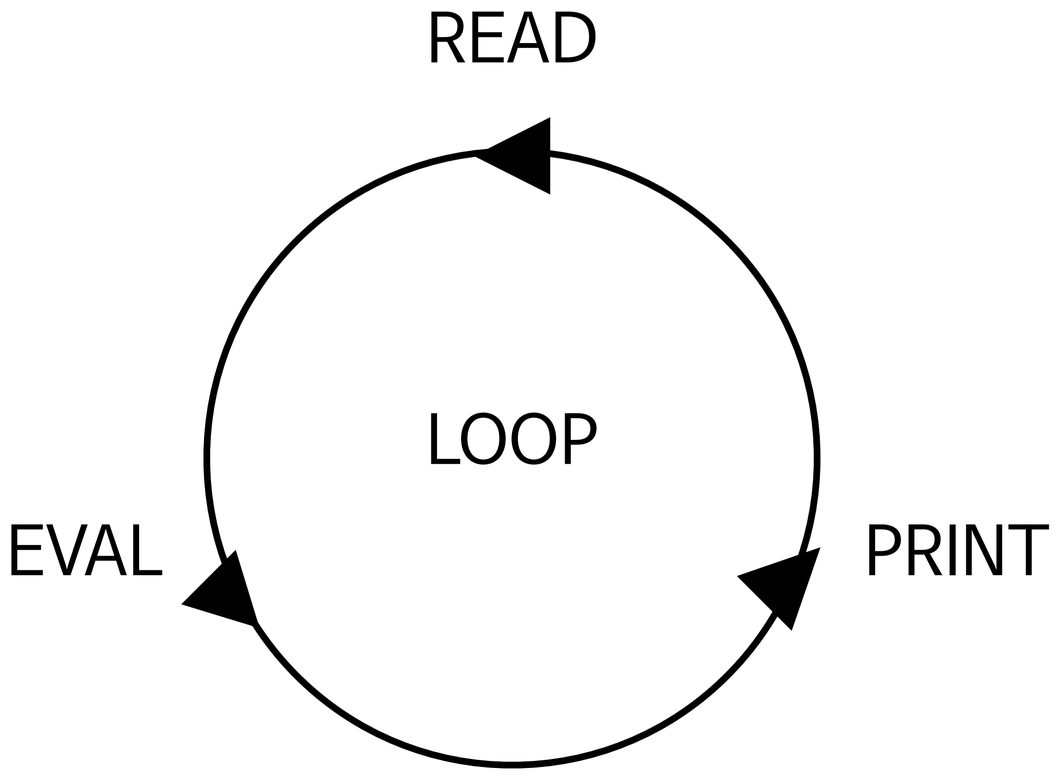
REPL, Clojure 의 슈퍼 파워

Clojure가 제안하는 새로운 개발 패러다임과 철학

김상현 그린랩스

REPL 이란?



REPL 의 특징

빠른 즉시 평가

점진적인 로직 설계

실행 중인 프로그램과 상호작용 가능

Clojure + REPL

✓ 동적 바인딩

✓ 데이터 중심

✓ 디버깅

Clojure + REPL

✓ 동적 바인딩

✓ 데이터 중심

✓ 디버깅

Var

```
(def x 1)
```

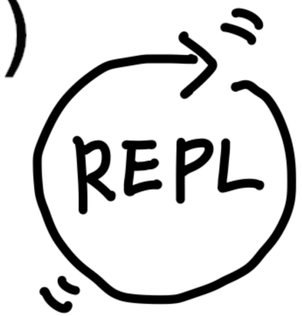
Var

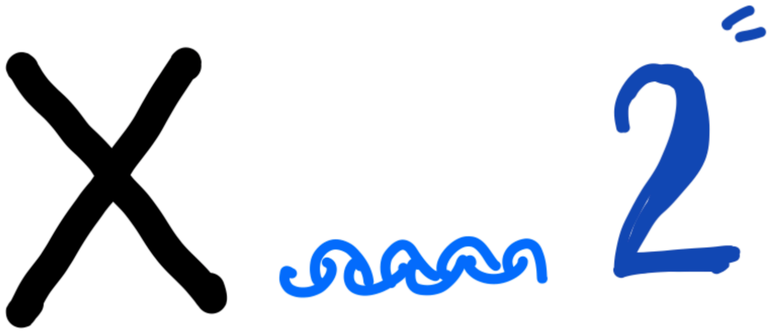
```
(+ x 1)  => 2
```

$$(X_{\sim 1}) + 1 = 2$$

Var

```
(def x 2)
```

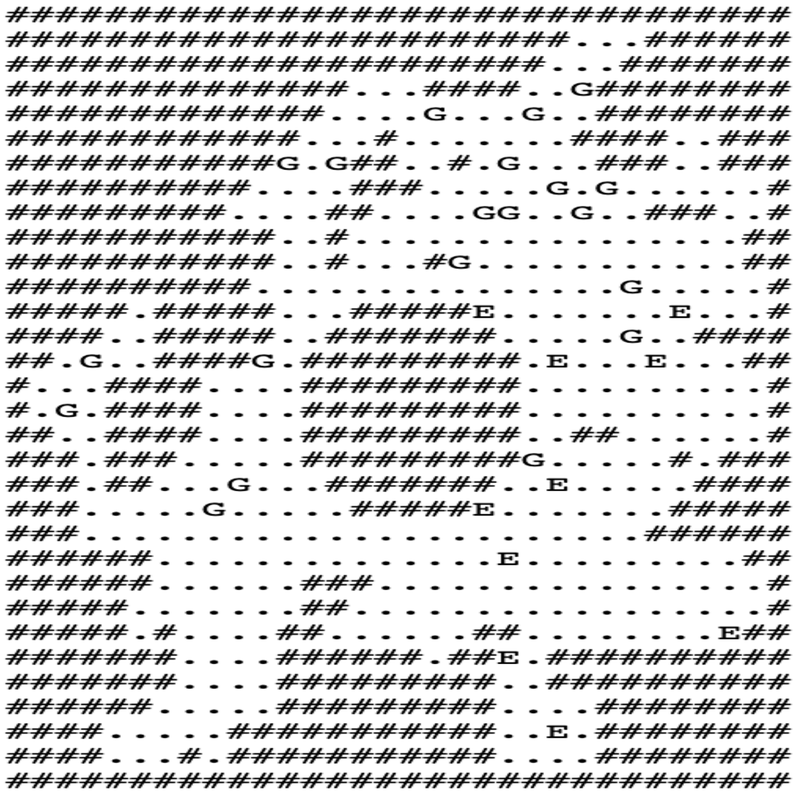




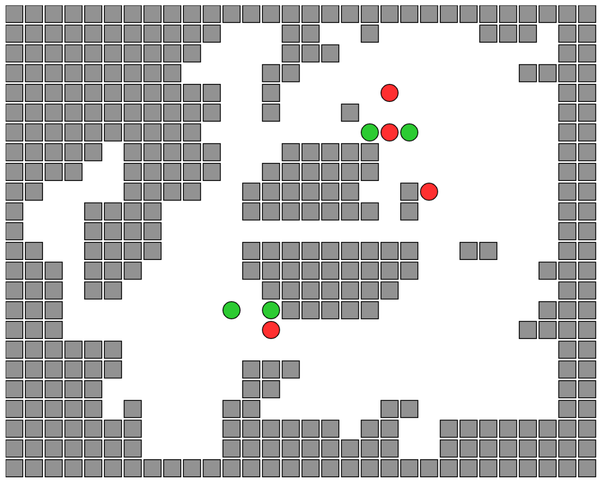
동적 바인딩 – 시연

고블린 vs. 엘프 전투 시뮬레이션

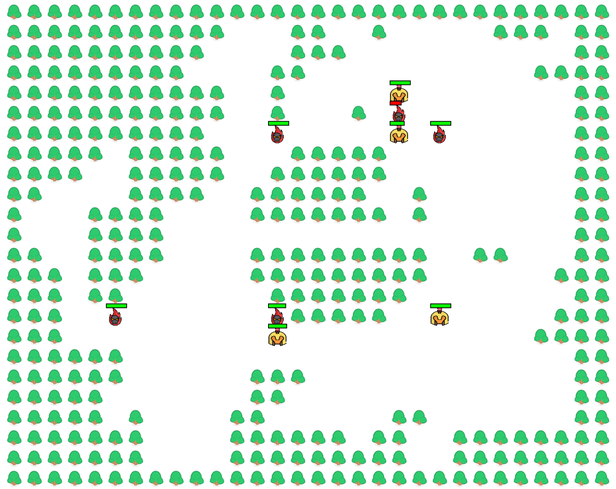
한쪽팀이 전부 없어지면 승리



동적 바인딩 – 시연



(draw-map)



(draw-map)

Clojure + REPL

✓동적 바인딩 ✓데이터 중심 ✓디버깅

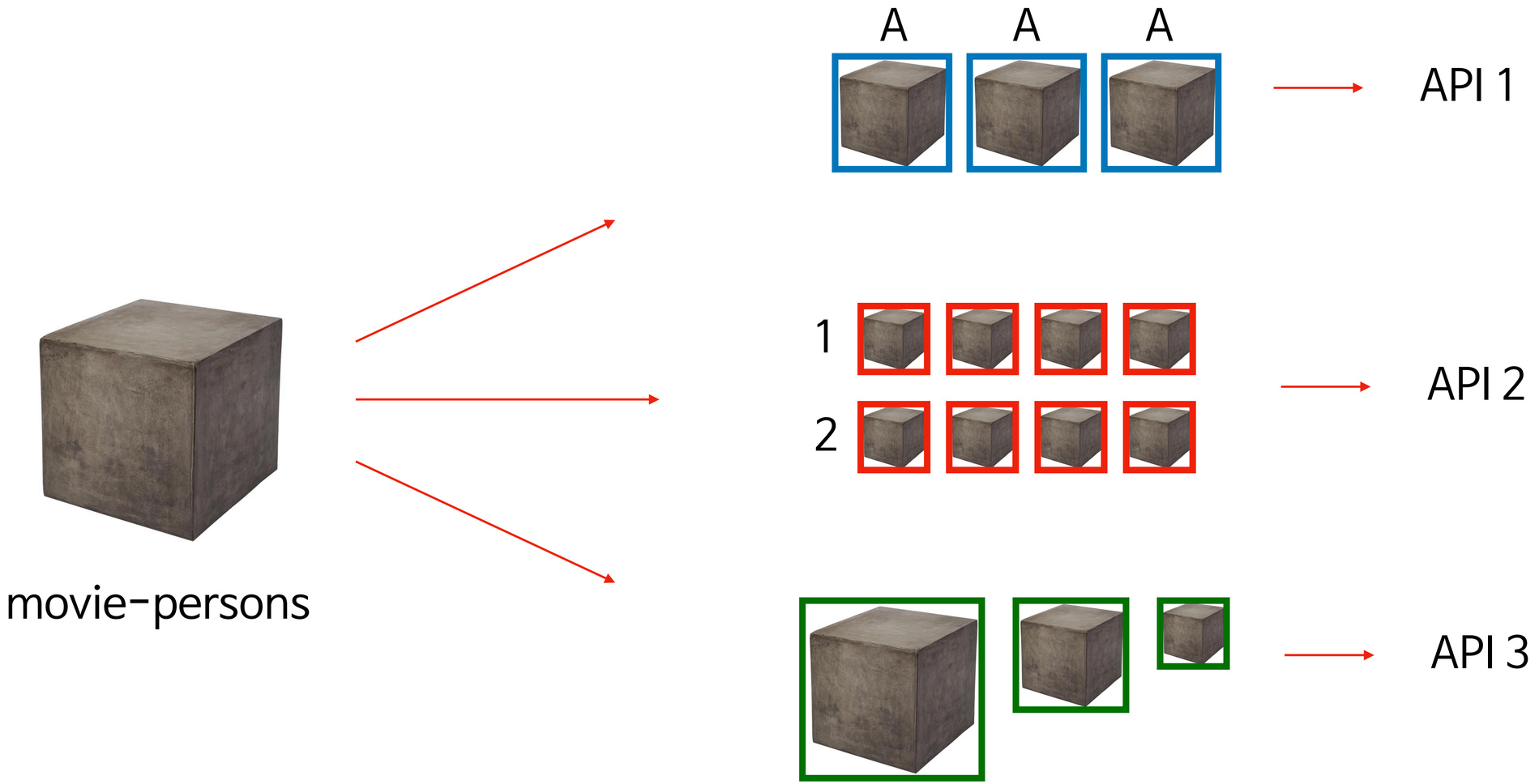
데이터 중심

객체가 아닌 **데이터**를 다룸

데이터 중심 개발 - 영화 API 다루기

영화인 (배우, 감독, 스텝) 리스트

데이터 중심 개발 - 영화 API 다루기



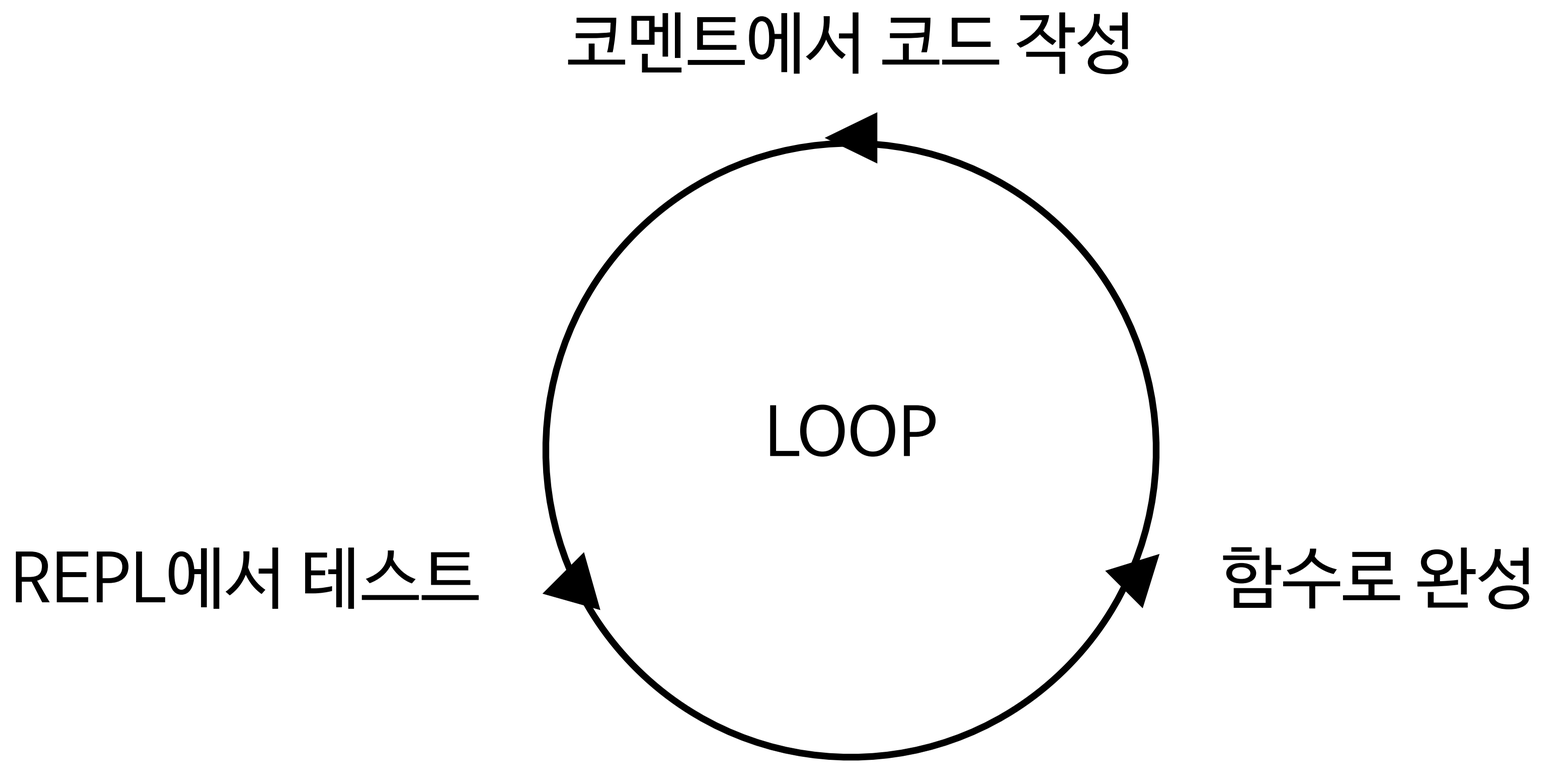
Clojure + REPL

✔ 동적 바인딩

✔ 데이터 중심

✔ 디버깅

디버깅 시연



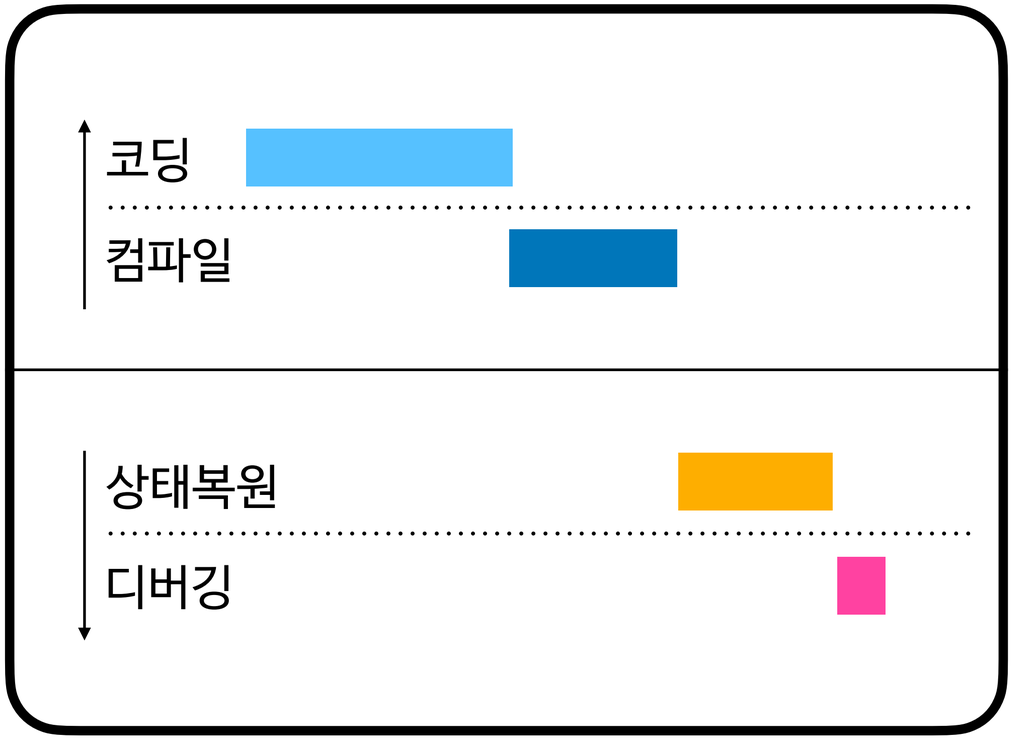
Comment

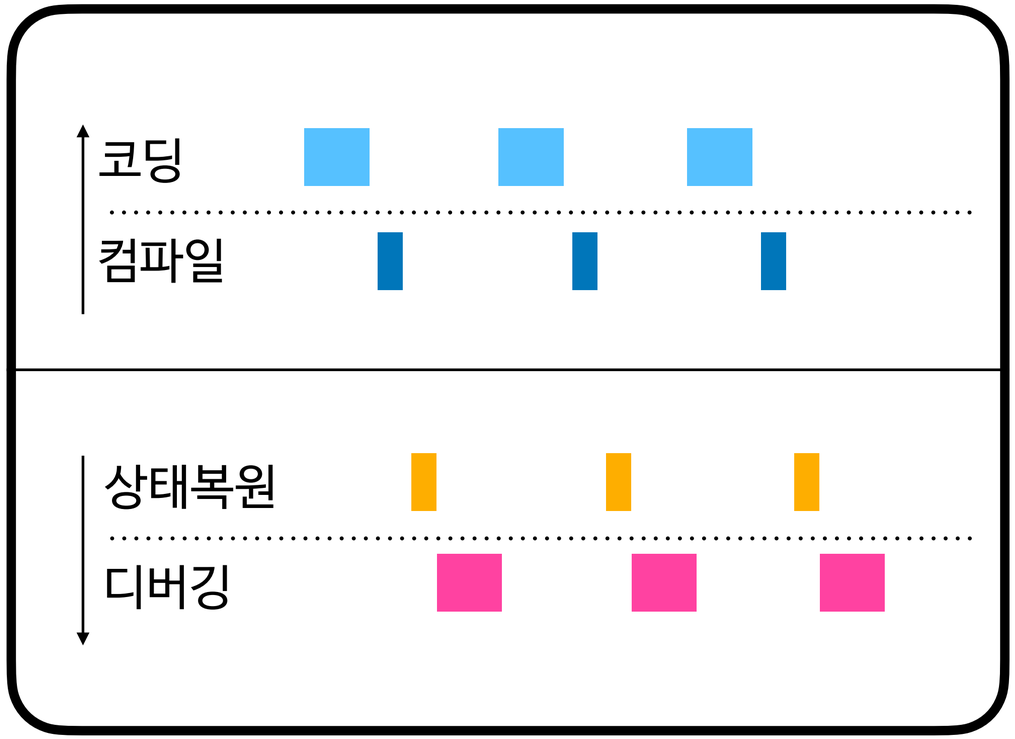
프로덕션 **코드의 재료**

즉각 테스트가 가능한 **로직의 조각**

소스의 일부

REPL 기반 개발의 특징





REPL 친화적 언어의 특징

✔ 불변성

✔ 표현력

불변 자료의 이점

객체의 운명이 정해짐 ⟶ 상태 예측 할 **필요가 없음**

불변 자료의 이점

객체의 운명이 정해짐 → 상태 예측 할 **필요가 없음**

한번 동일하면 → 영원히 **동일함**

불변 자료의 이점

객체의 운명이 정해짐 → 상태 예측 할 **필요가 없음**

한번 동일하면 → 영원히 **동일함**

방어적 복사 X → 공유가 **쉬워짐**

불변 자료의 이점

객체의 운명이 정해짐 → 상태 예측 할 **필요가 없음**

한번 동일하면 → 영원히 **동일함**

방어적 복사 X → 공유가 **쉬워짐**

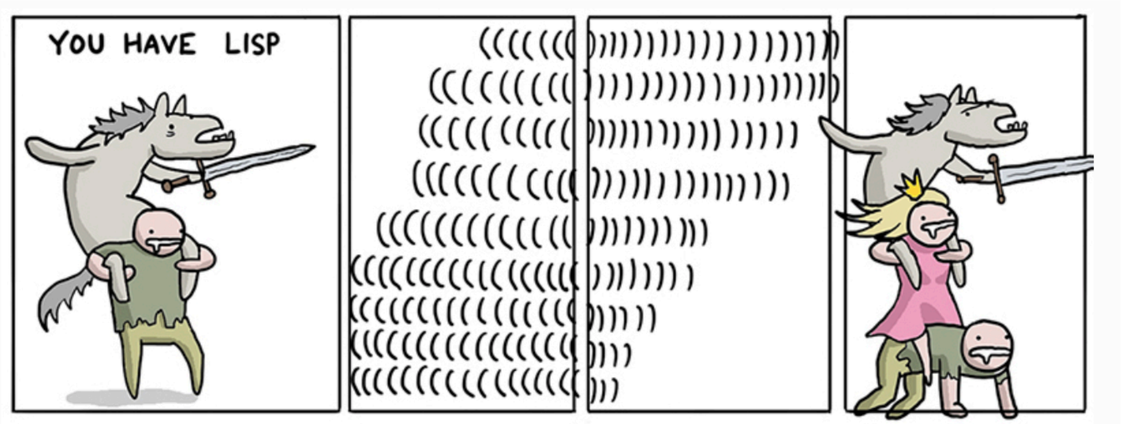
항상 thread-safe → 병렬 프로그래밍이 **간단해짐**

표현력

S-표현식 = 괄호

괄호는 불편하지 않습니다!

parinfer, paredit



표현력

S-표현식

[exp1] [exp2] [exp3]

[[exp4] […]]

일반 코드

```
[
  … (exp1)
  … (exp2)
  … (exp3)
]
```

Clojure + REPL은...

다이나믹하다.

데이터 중심적이다.

빠르고 재미있다!

감사합니다.