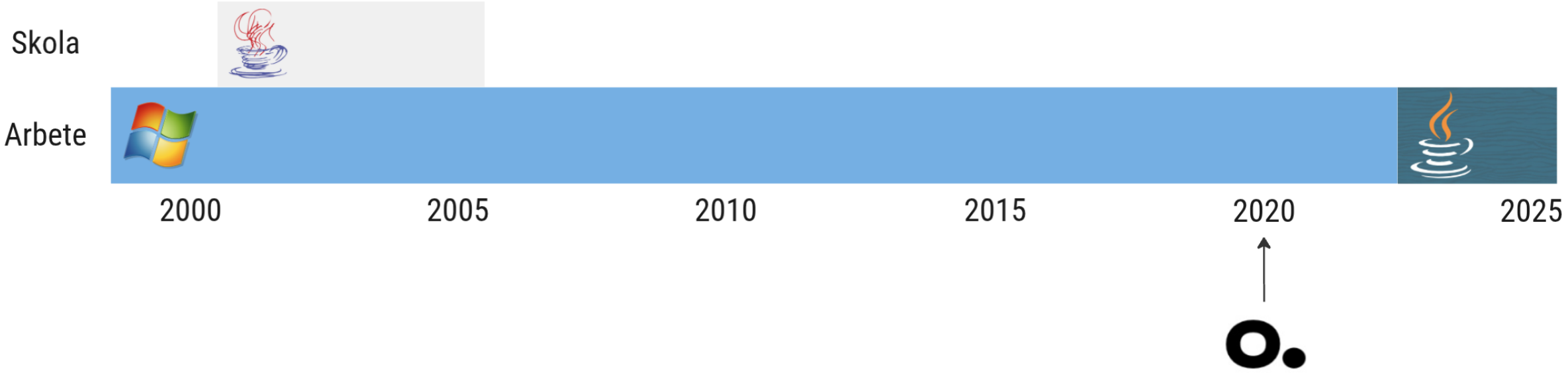

omega
point.

JEP-453

Snyggare samtidighet i Java

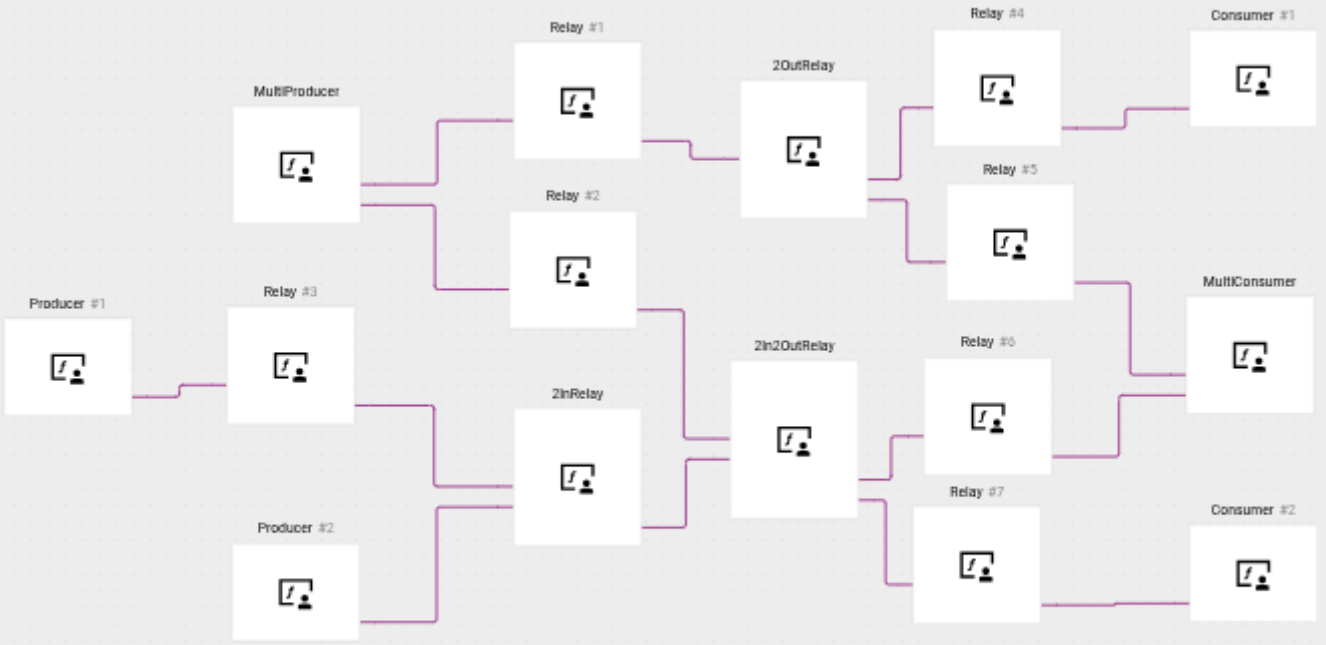
psst... JEP-505...

Min bakgrund



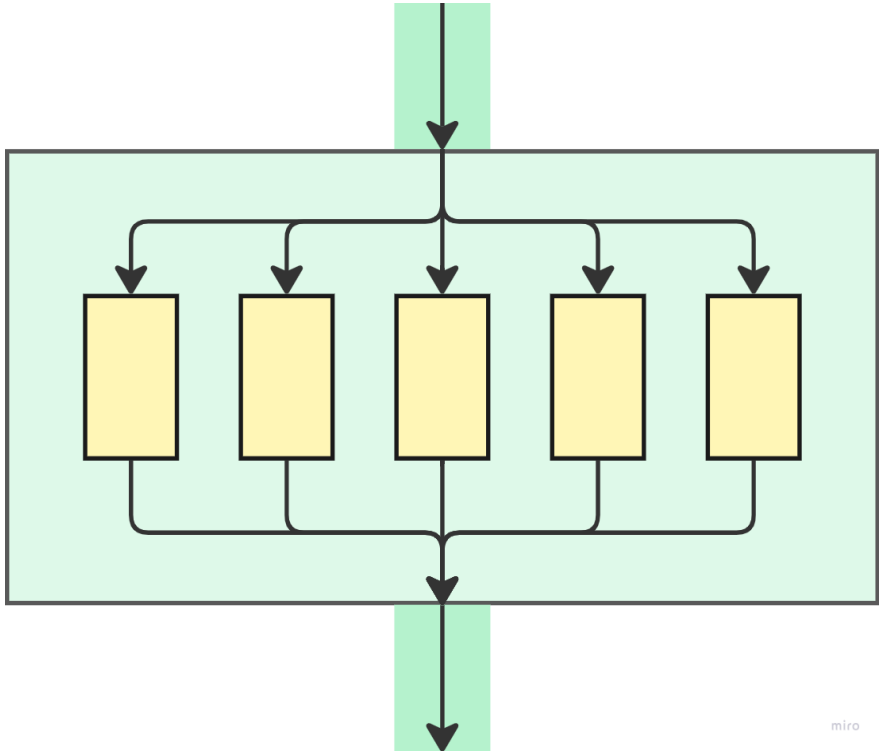
INTRO

Uppdrag Java (Scala)



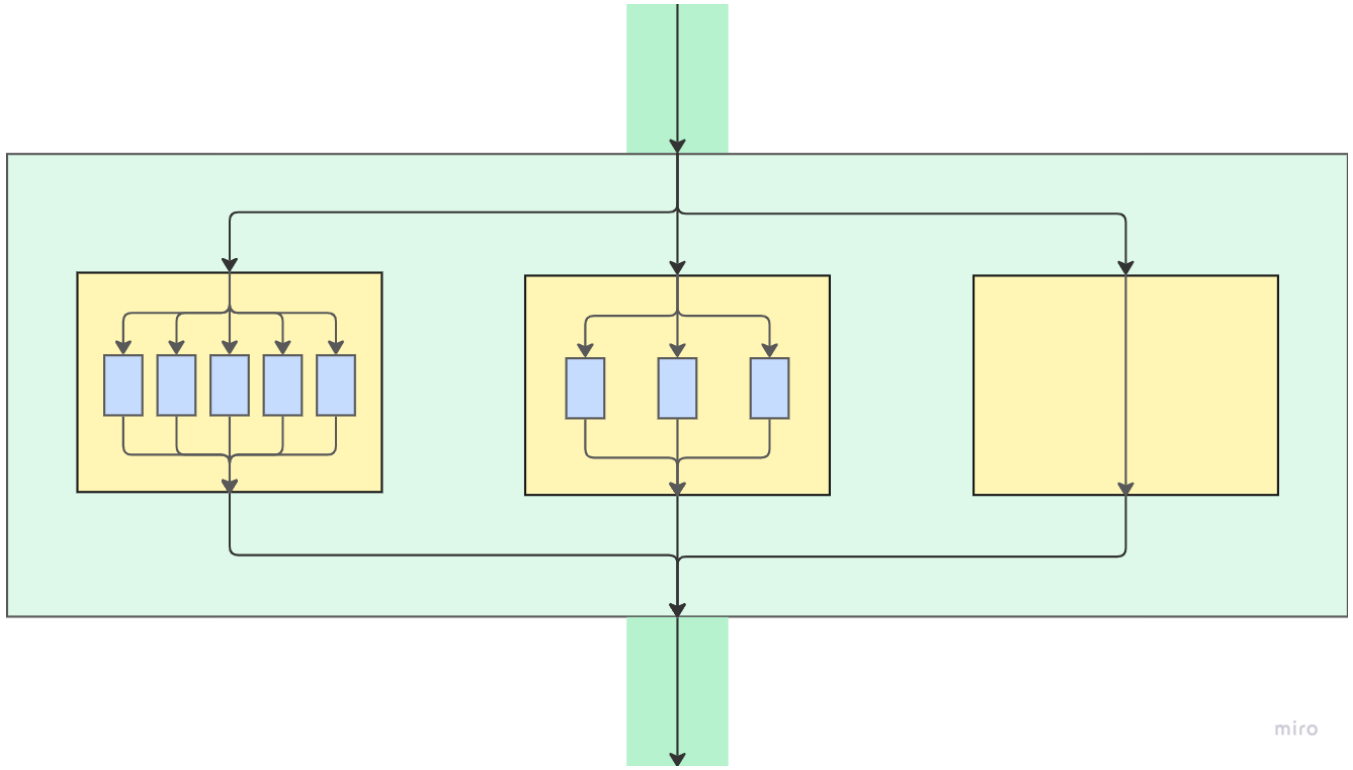
Structured Concurrency - hisspitchen

- Om man har en uppgift (*task*)
  som delas upp i samtidigt körande deluppgifter (*subtasks*)
  garanteras att alla deluppgifterna kört klart
  innan huvuduppgiften kan fortsätta
- Resultatet av deluppgifterna finns endast tillgängligt
  i det kodblock som skapade dem



Structured Concurrency - hisspitchen

- Deluppgifter kan i sin tur dela upp sig i ytterligare deluppgifter och bildar då en hierarki



OpenJDK - Project Loom

- Virtual Threads
  - ✅ Java 21
- Scoped values
  - *Out of scope idag*
- Structured Concurrency...

JEP historik – Structured Concurrency

- JEP = *JDK Enhancement Proposal*
- JEP 453 (Preview) Java 21 september 2023
- JEP 462 (Second Preview) Java 22 *(no API changes)*
- JEP 480 (Third Preview) Java 23 *(no API changes)*
- JEP 499 (Fourth Preview) Java 24 *(no API changes)*
- Ny LTS i höst!?! Java 25 🤞
- JEP 505 (**Fifth Preview**) API changes! 😫
- *Bra API changes dock!*

OpenJDK

JEP Goals and non-goals

- Goals
  - Främja en programmeringsstil för samtidiga uppgifter som kan eliminera vanliga risker rörande avbrott och nedstängning såsom trådläckor och fördröjningar *(nästa slide)*
  - Ge bättre "observability"
- Non-goals
  - Ska inte ersätta något i java.util.concurrency
  - Inte låta subtasks/trådar prata med varandra
  - Inte ändra avbrottshantering, arbetar med existerande interrupt-mekanismer
  - Inte vara det definitiva/slutgiltiga "structured concurrency" api:t
- Resultat: litet api med stort design space

Problem #1 – trådläcka

```
// (1) submit two tasks
Future<String> futureHello = executorService.submit(ProblemThreadLeak::getHelloThrows);
Future<String> futureWorld = executorService.submit(ProblemThreadLeak::getWorld);

try {
    var helloResult = futureHello.get(); // (2) blocking on result from task getHello
    var worldResult = futureWorld.get();

    log("Main thread result: %s %s".formatted(helloResult, worldResult));
} catch (ExecutionException e) {
    log("Main thread handles exception: " + e.getCause()); // (3) handle getHello failure
}
// (4) getWorld() will still complete a while later
```

Problem #2 – fördröjda fel

```java
try {
    // (1) submit two tasks
    Future<String> futureHello = executorService.submit(ProblemCancellationDelay::getHello);
    Future<String> futureWorld = executorService.submit(ProblemCancellationDelay::getWorldThrows);

    var helloResult = futureHello.get(); // (2) blocking (while getWorld fails in background)
    var worldResult = futureWorld.get(); // (3) a while later we query for the result of getWorld
                                         //     and only now find the Exception

    log("Main thread result: %s %s".formatted(helloResult, worldResult));
} catch (ExecutionException e) {
    log("Main thread failed: " + e.getCause()); // (4) delayed exception handling
}
```

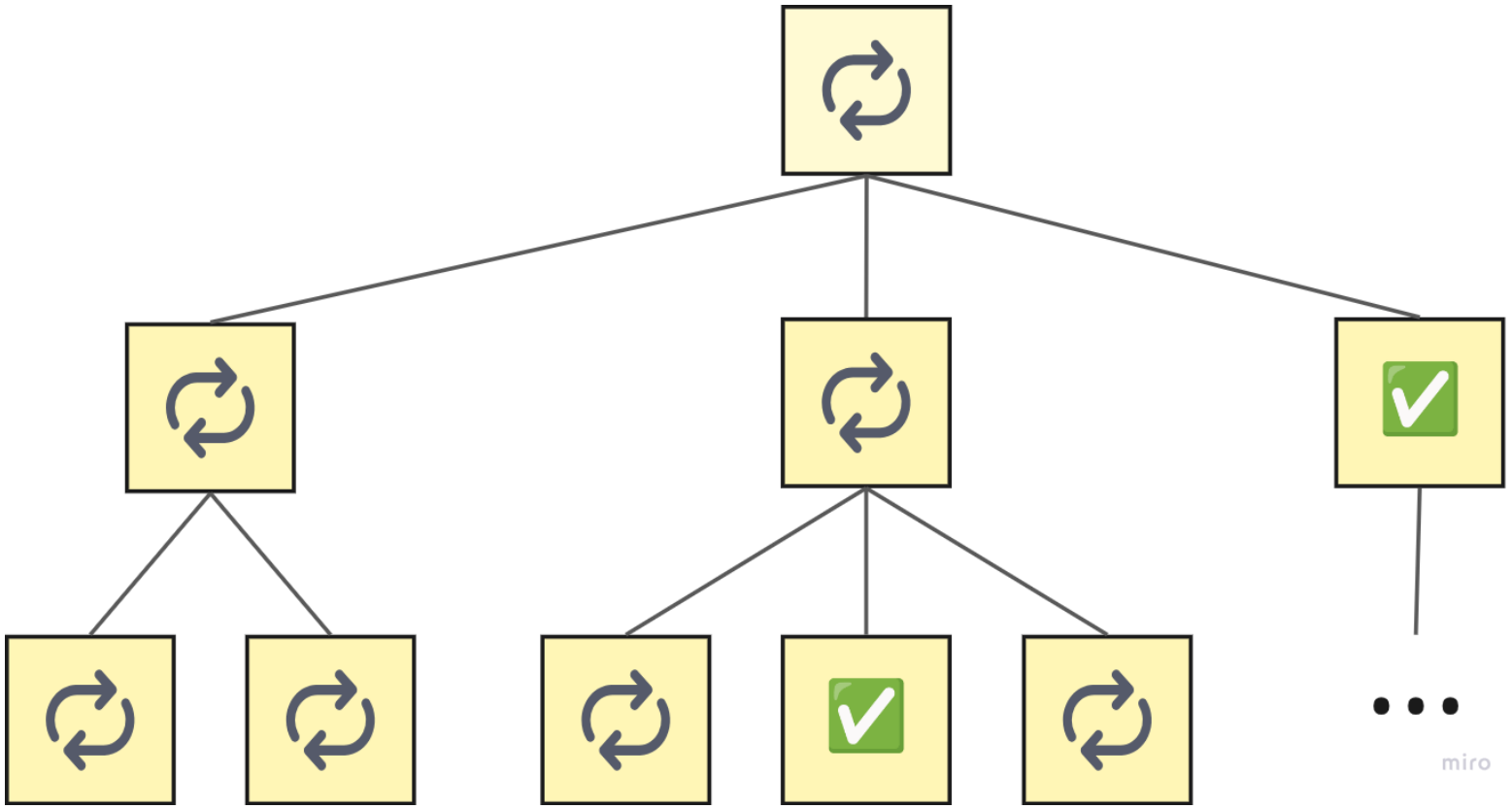
JEP-505

Hur ser API:t ut då!?

```
try (var scope = StructuredTaskScope.open()) {

    Subtask subtask1 = scope.fork(() -> query(left));
    Subtask subtask2 = scope.fork(() -> query(right));

    // throws if either subtask fails
    scope.join();

    // both subtasks completed successfully
    return new MyResult(subtask1.get(), subtask2.get());

} // close
```

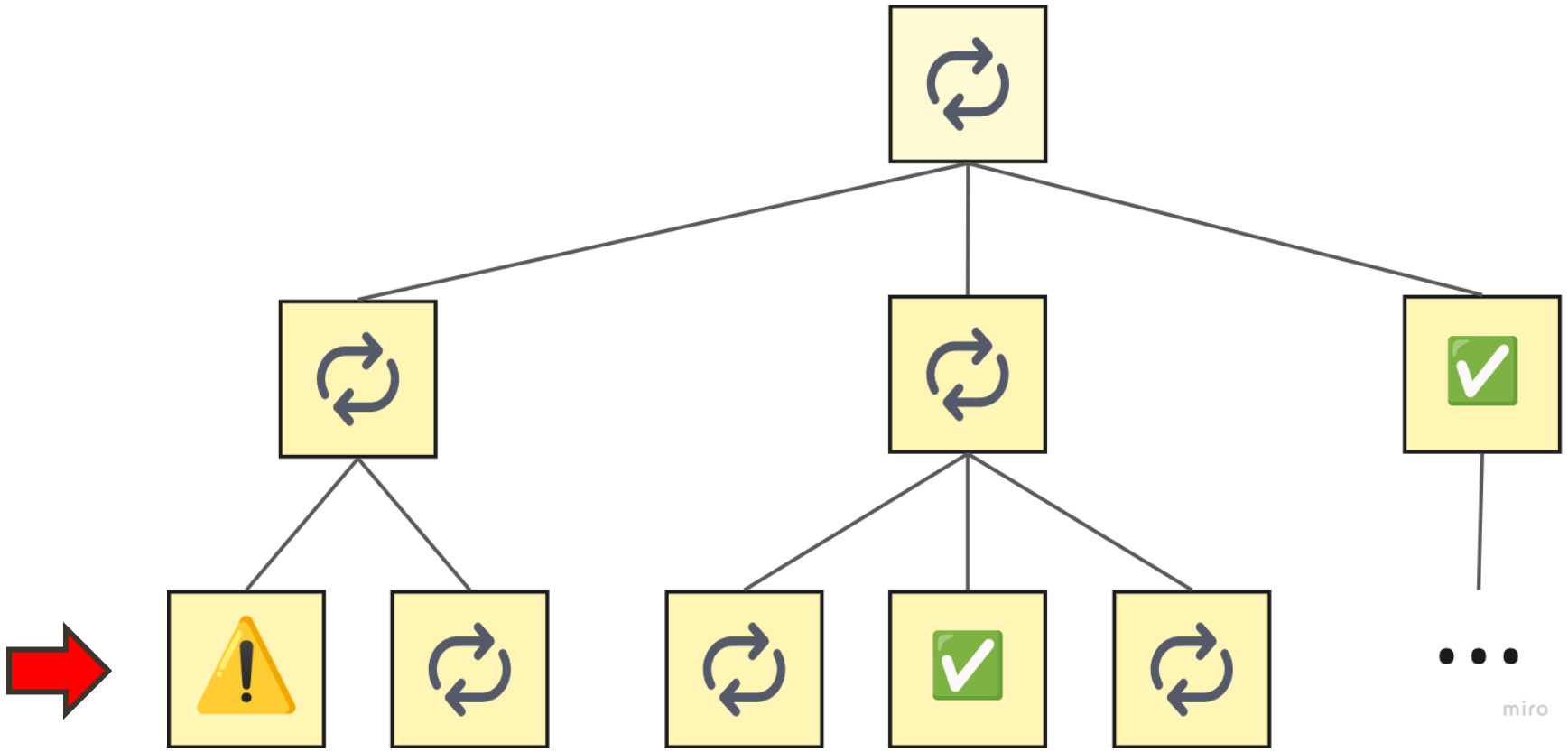
JEP-505

Felhantering



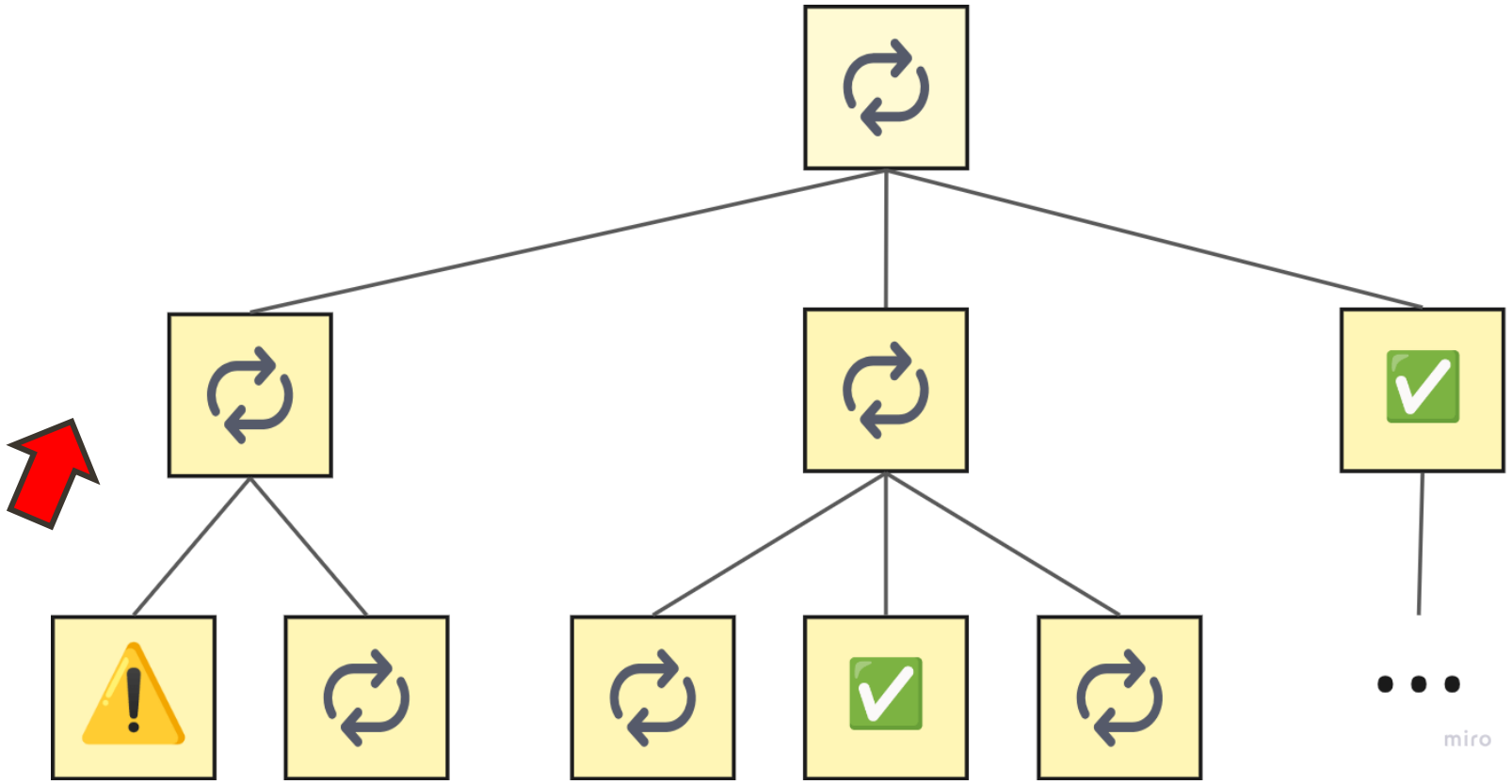
JEP-505

Felhantering



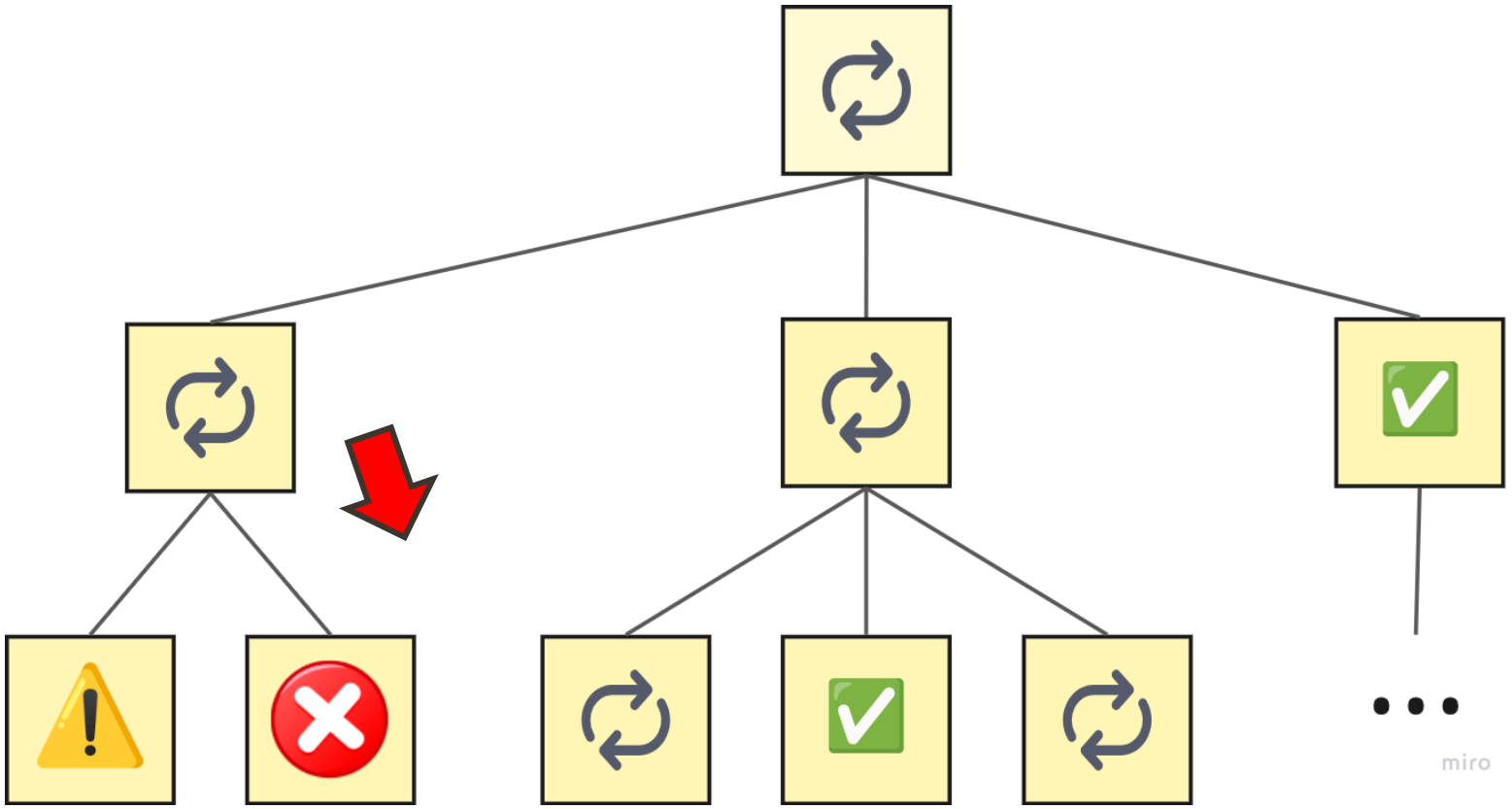
JEP-505

Felhantering



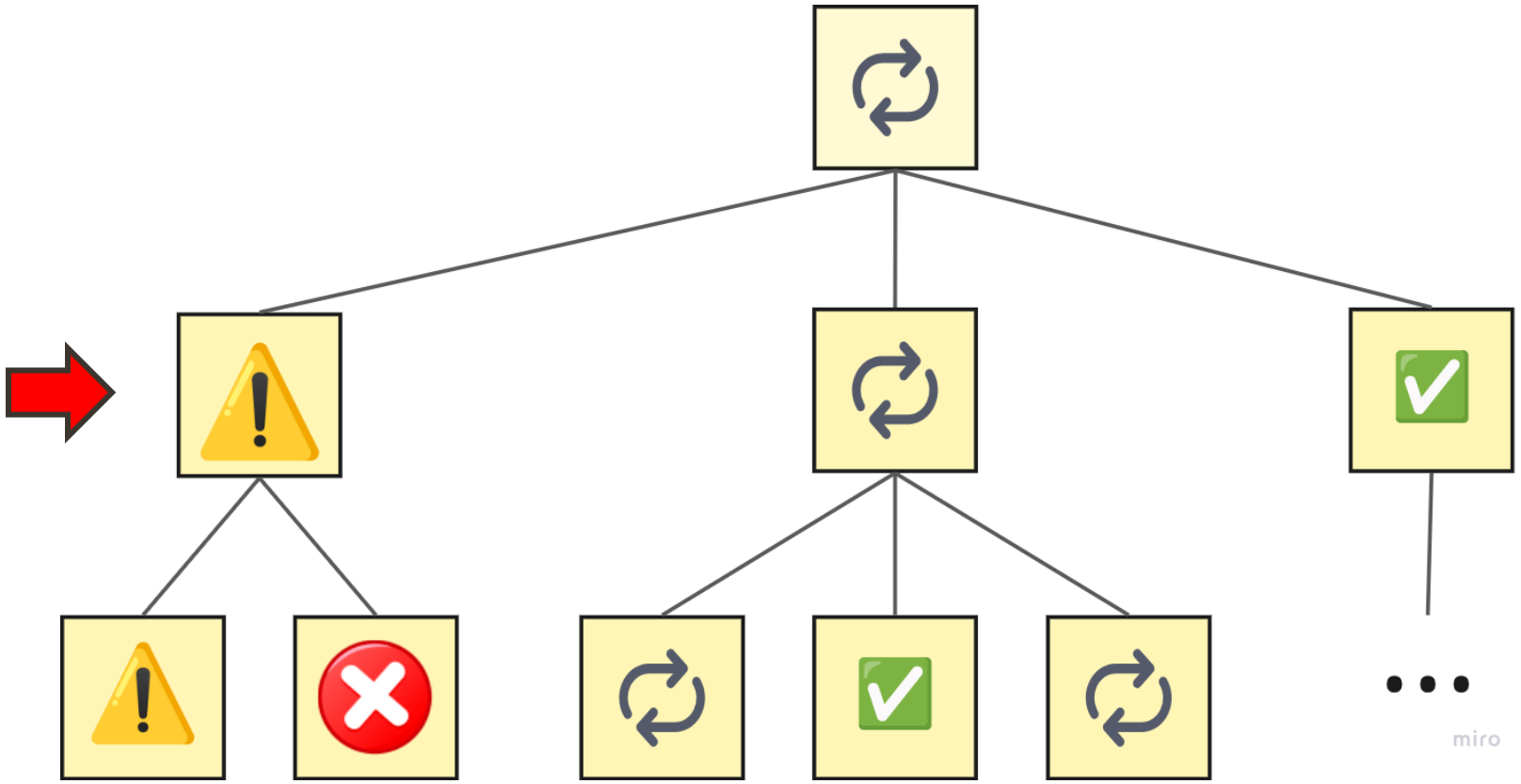
JEP-505

Felhantering



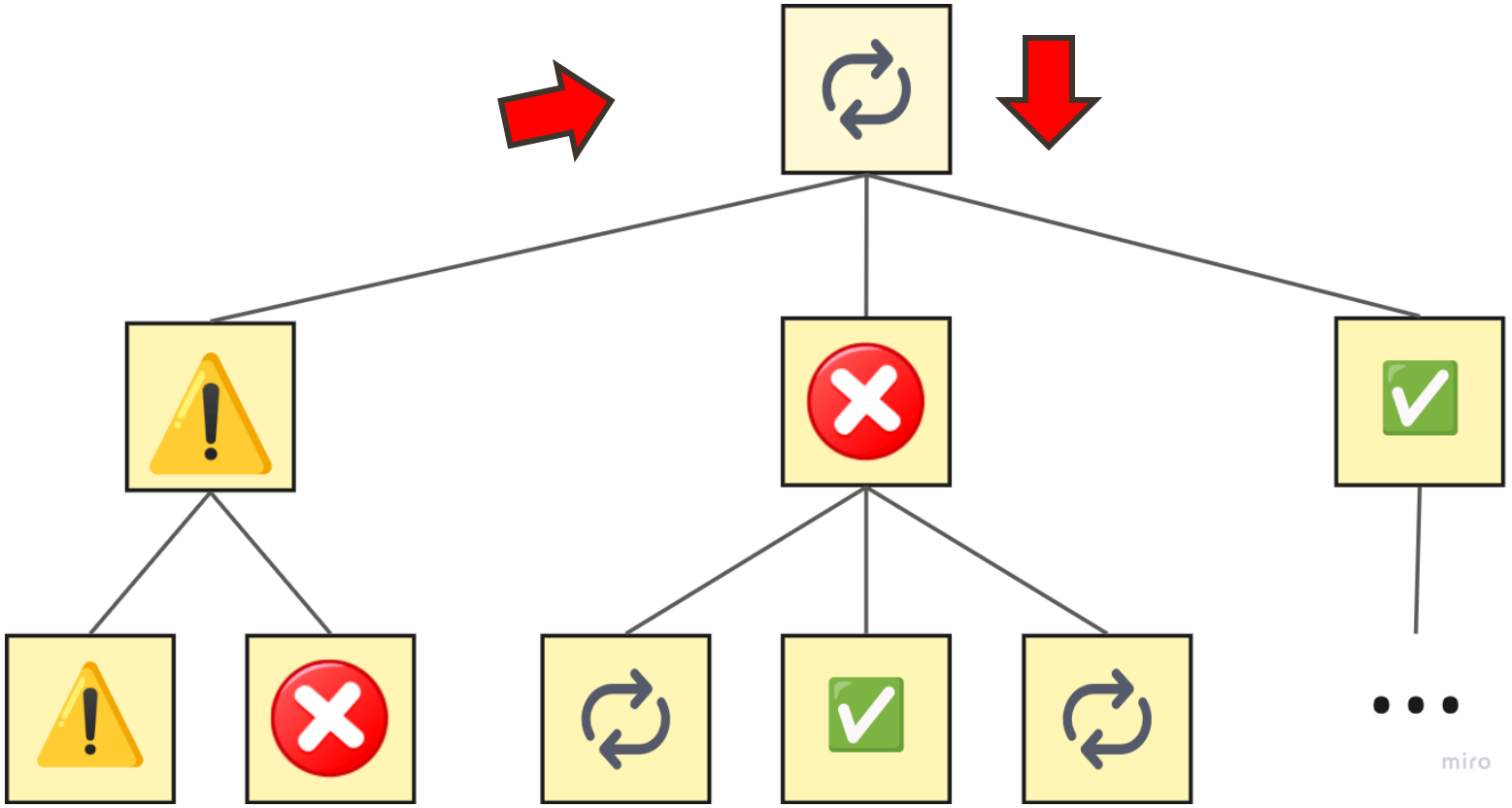
JEP-505

Felhantering



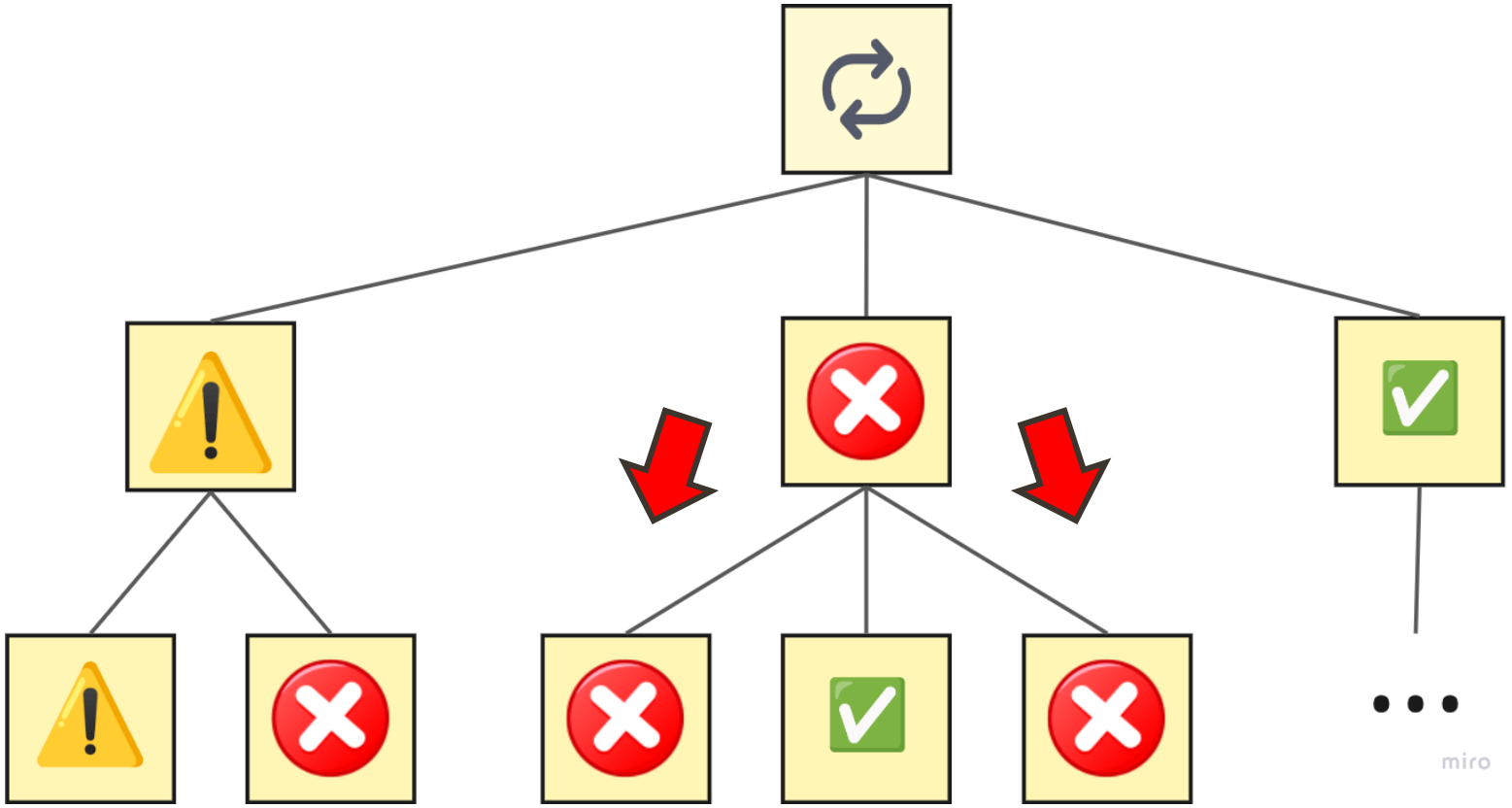
JEP-505

Felhantering



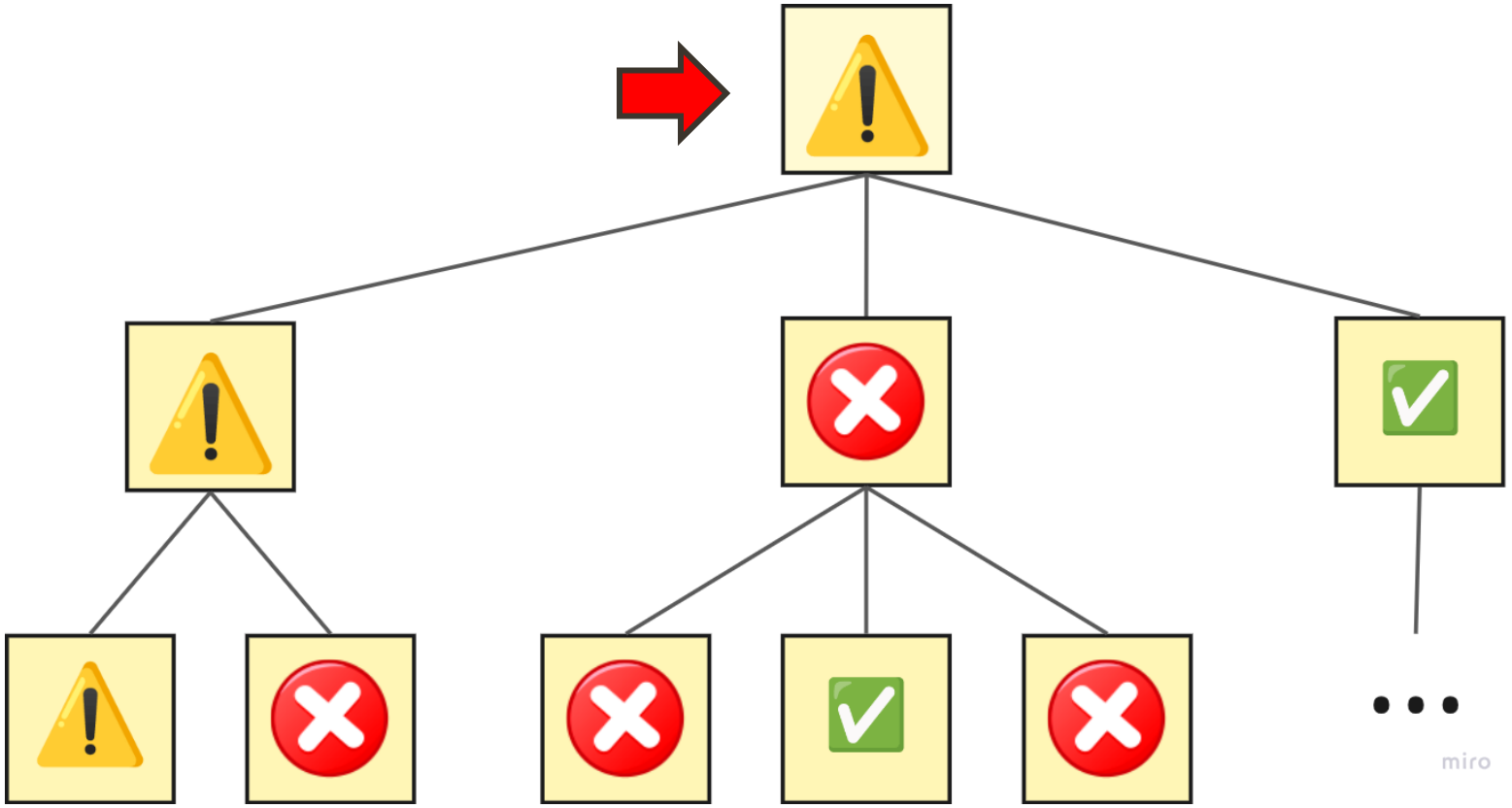
JEP-505

Felhantering



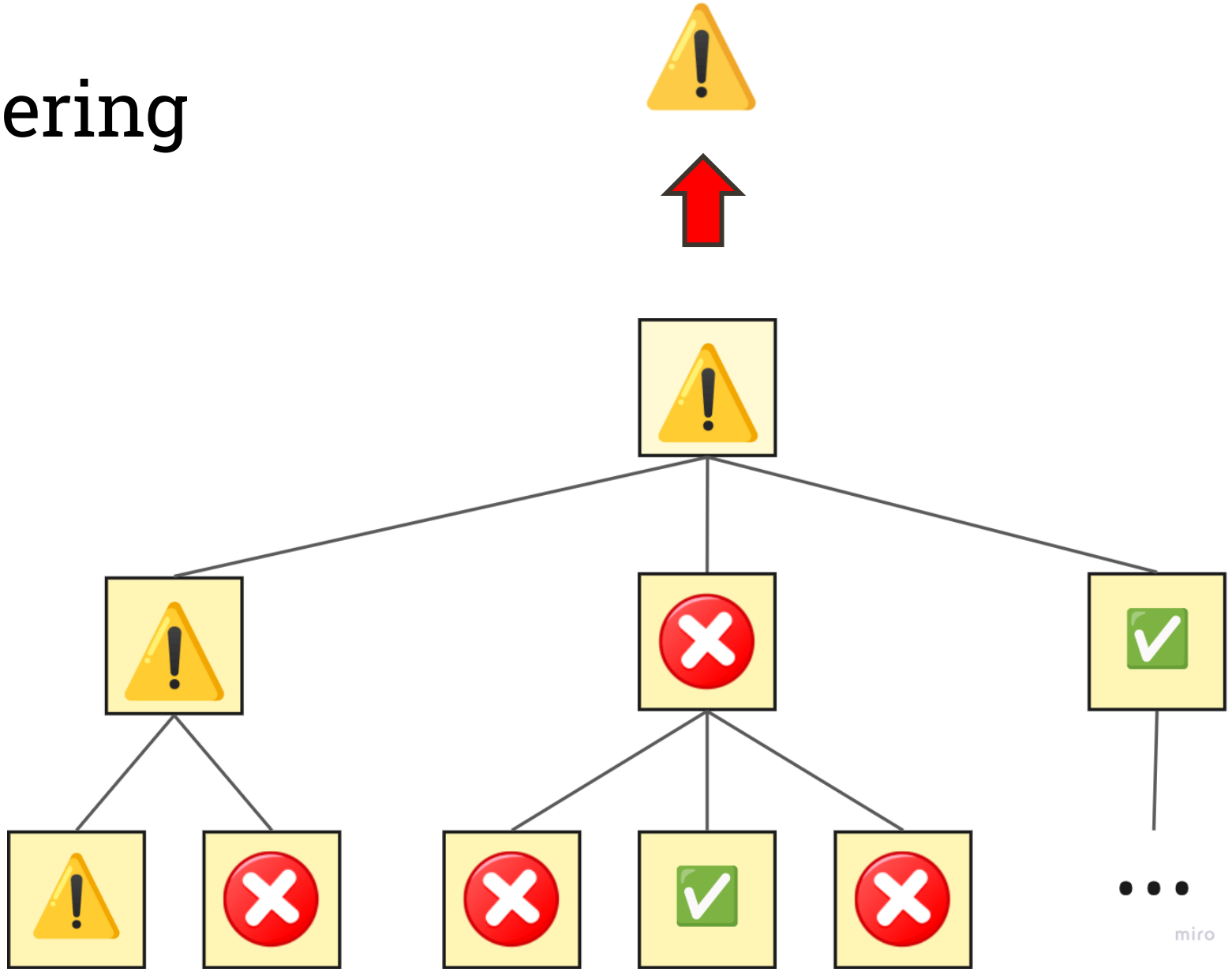
JEP-505

Felhantering



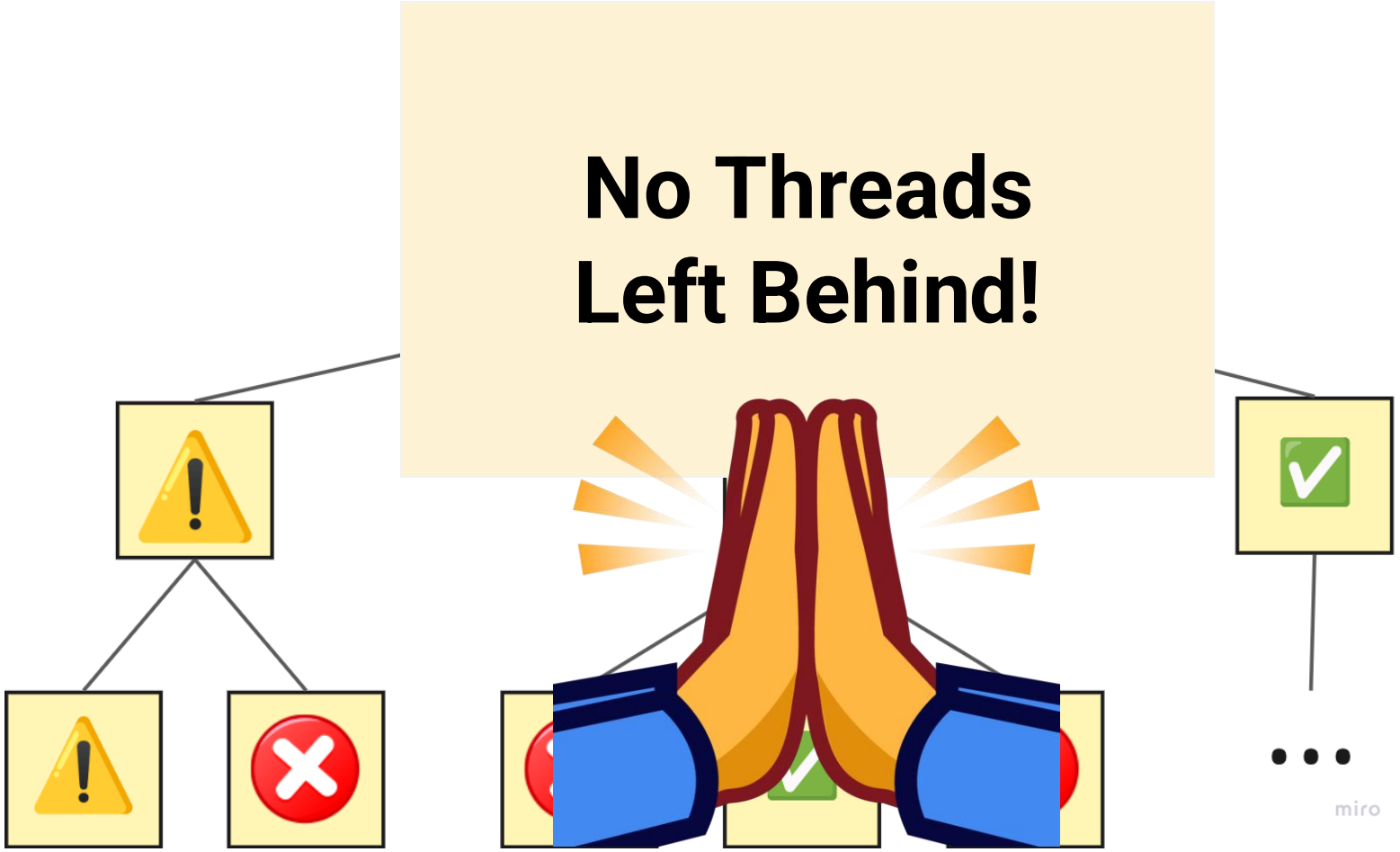
JEP-505

Felhantering



JEP-505

Felhantering



JEP-505

Slutmålet

- Imperativ kod...

```
private MyResult getResult(int left, int right) {

    var leftQuery = query(left);
    var rightQuery = query(right);



    return new MyResult(leftQuery, rightQuery);
}
```

- Imperativ samtidighet!

```
try (var scope = StructuredTaskScope.open()) {

    var leftQuery = scope.fork(() -> query(left));
    var rightQuery = scope.fork(() -> query(right));

    scope.join();

    return new MyResult(leftQuery.get(), rightQuery.get());
}
```

JEP-505

Konfiguration

- StructuredTaskScope.open()
- StructuredTaskScope.open(Joiner, Configuration)
- Joiner (Policy)
  - awaitAllSuccessfulOrThrow() - default
  - anySuccessfulResultOrThrow()
  - …
  - Implementera interfacet själv! (helst inte)
- Configuration
  - Namn
  - ThreadFactory
  - Timeout

Demo

THE END

Tack för idag!

https://github.com/mnsc/koko-251-structured-concurrency/