Inside Vitest 

Test Framework Architecture Deep Dive

About Me

- Hiroshi Ogawa @hi-ogawa
- Vite and Vitest core team member
- Working at VoidZero
- SSR meta-framework fanatic
- Vite RSC support `@vitejs/plugin-rsc`

What is Vitest?

Testing framework

```ts
// packages/vite/src/node/__tests__/scan.spec.ts
import path from 'node:path'
import { describe, test, expect } from 'vitest'
import { commentRE, } from '../optimizer/scan'

describe('optimizer-scan:script-test', () => {
  /* ... */

  test('component return value test', () => {
    scriptRE.lastIndex = 0
    const [, tsOpenTag, tsContent] = scriptRE.exec(
      `<script lang="ts">${scriptContent}</script>`,
    )!
    expect(tsOpenTag).toEqual('<script lang="ts">')
    expect(tsContent).toEqual(scriptContent)

    scriptRE.lastIndex = 0
    const [, openTag, content] = scriptRE.exec( ... )!
    expect(openTag).toEqual('<script>')
    expect(content).toEqual(scriptContent)
```

```
> vitest run --reporter=verbose

 RUN  v3.2.4 /xxx/vite

...
     ✓ should throw error when warning contains UNRESOLVED_IMPORT 2ms
     ✓ should ignore dynamic import warnings (Unsupported expression) 1ms
     ✓ should ignore dynamic import warnings (statically analyzed) 1ms
     ✓ should ignore some warnings (CIRCULAR_DEPENDENCY) 1ms
     ✓ should ignore some warnings (THIS_IS_UNDEFINED) 1ms
   ✓ watch rebuild manifest 25ms
 ✓ packages/vite/src/node/__tests__/plugins/index.spec.ts (6 tests) 71ms
   ✓ hook filter with plugin container (3)
     ✓ resolveId 2ms
     ✓ load 1ms
     ✓ transform 63ms
   ✓ hook filter with build (3)
     ✓ resolveId 0ms
     ✓ load 0ms
     ✓ transform 0ms
✓ packages/vite/src/node/__tests__/scan.spec.ts (7 tests) 1014ms
   ✓ optimizer-scan:script-test (6)
     ✓ component return value test 3ms
     ✓ include comments test 0ms
     ✓ components with script keyword test 0ms
     ✓ ordinary script tag test 0ms
     ✓ imports regex should work 1ms
     ✓ script comments test 1ms
   ✓ scan jsx-runtime  1007ms

 Test Files  46 passed (46)
      Tests  660 passed (660)
   Start at  12:58:28
   Duration  2.48s (transform 2.32s, setup 0ms, collect 17.44s, tests 9.23s, envir
```

What is Vitest?

Features

- Jest-compatible API and feature set
  - `describe`, `test`, `expect`, ...
  - mocking, snapshot, coverage, ...
- ESM and TypeScript support out of the box
  - Vite builtin features available
- Extensible via Vite plugin ecosystem
  - React, Vue, Svelte, ...
- Runtime agnostic
  - Node.js, Browser Mode, Cloudflare Workers
- Advanced API and Programmatic API
  - Powerful customization to serve ecosystem (Storybook, VSCode extension, Cloudflare, ...)

```ts
// [add.test.ts]
import { test, expect } from "vitest"
import { add } from "./add"

test('add', () => {
  expect(add(1, 2)).toBe(3)
})
```

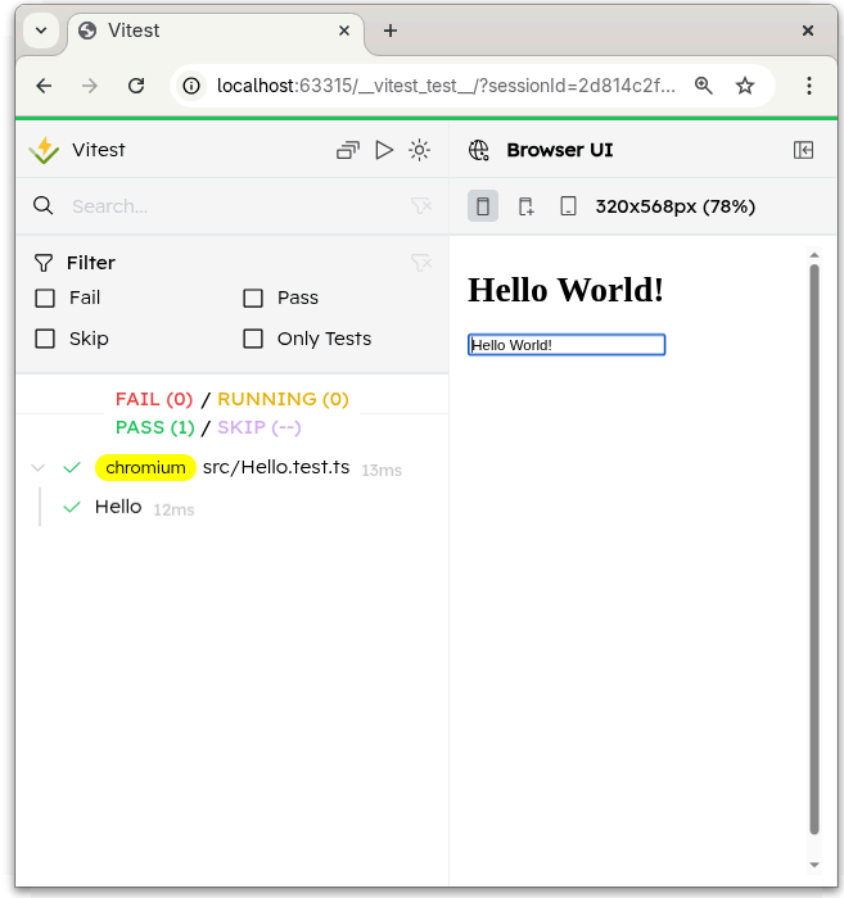
```ts
// [Hello.test.ts]
import { test, expect } from "vitest"
import { mount } from '@vue/test-utils'
import Hello from "./Hello.vue";

test('Hello', () => {
  const wrapper = mount(Hello, { attachTo: document.body })
  expect(wrapper.text()).toContain('Hello')
})
```

What is Vitest?

Runtime agnostic → Browser Mode

```ts
// [Hello.test.ts]
import { test, expect } from "vitest"
import { page } from "vitest/browser";
import { mount } from '@vue/test-utils'
import Hello from "./Hello.vue";

test('Hello', () => {
  mount(Hello, { attachTo: document.body })
  await expect.element(page.getByText('Hello')).toBeVisible(
})
```

```ts
// [vitest.config.ts]
import { defineConfig } from "vitest/config"
import vue from '@vitejs/plugin-vue';
import { playwright } from '@vitest/browser-playwright'

export default defineConfig({
  plugins: [vue()],
  test: {
    browser: {
      enabled: true,
      provider: playwright(),
      instances: [{ browser: 'chromium' }],
    },
```



Vitest 4 is out!

- Announcement Blog
- State of Vitest (ViteConf 2025) by Vladimir @sheremet-va

Overview

This talk follows the **test lifecycle** to explore Vitest architecture:

- Orchestration → Collection → Execution → Reporting

Along the way, we'll see how each package divide responsibilities:

- `vitest`, `@vitest/runner`, `@vitest/browser`, `@vitest/expect`, ...

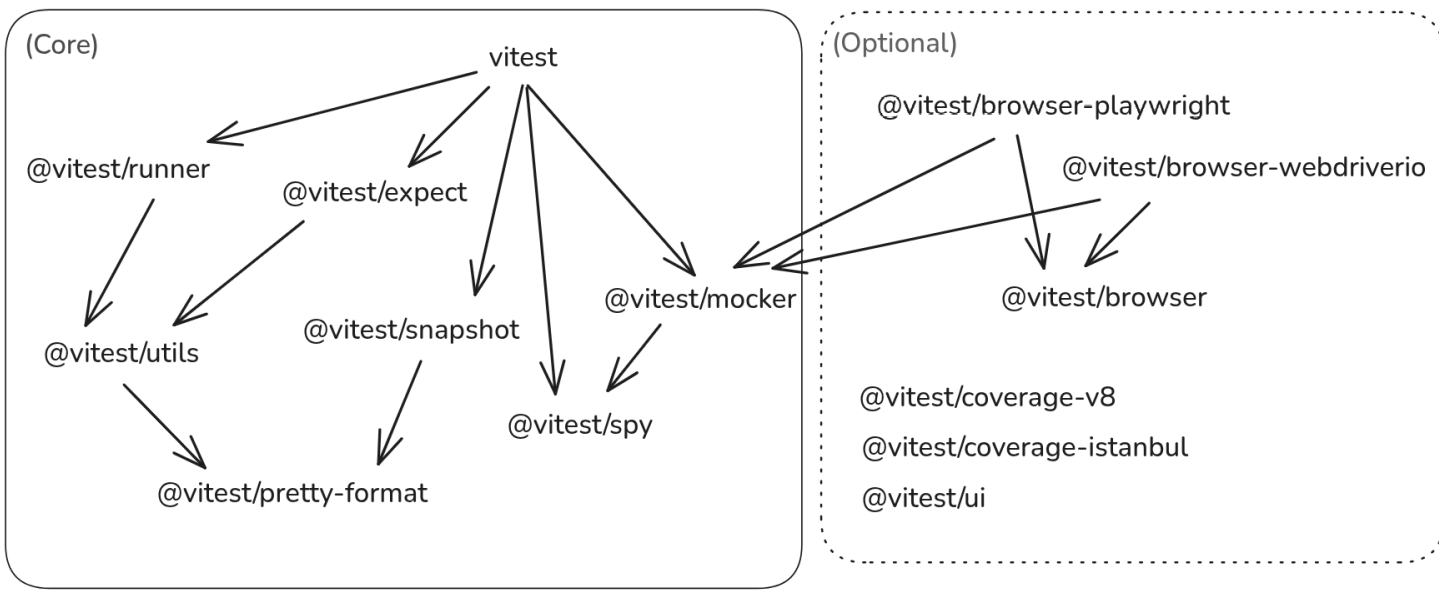
And finally we learn how **Vite** powers Vitest as a foundation:

- Environment API, Custom transform pipeline, ...

Vitest Monorepo Packages Dependencies

- **vitest** - cli entry, test orchestration, reporter, re-exports other packages
- **@vitest/runner** - `describe/suite`, `it/test`
- **@vitest/expect, @vitest/snapshot, @vitest/spy** -

  standalone libraries for `expect`, `toMatchSnapshot`, `vi.fn()`



Test Lifecycle

Example test run

```
// [add.test.ts]
import { test, expect, describe } from "vitest"
import { add } from "./add"

describe("add", () => {
  test('first', () => {
    expect(add(1, 2)).toBe(3)
  })
  test('second', () => {
    expect(add(2, 3)).toBe(4)
  })
})
```

```
// [mul.test.ts]
import { expect, test } from "vitest"
import { mul } from "./mul"

test("mul", () => {
  expect(mul(2, 3)).toBe(6)
})
```

```
> vitest --reporter tree

 DEV  v4.0.0-beta.18 /xxx/talks/2025-10-25/examples/lifecycle

 ❯ src/add.test.ts (2 tests | 1 failed) 5ms
   ❯ add (2)
     ✓ first 1ms
     × second 3ms
 ✓ src/mul.test.ts (1 test) 2ms
   ✓ mul 1ms

─────────────────── Failed Tests 1 ───────────────────

 FAIL  src/add.test.ts > add > second
AssertionError: expected 5 to be 4 // Object.is equality

- Expected
+ Received

- 4
+ 5

 ❯ src/add.test.ts:9:23
      7|   })
      8|   test('second', () => {
      9|     expect(add(2, 3)).toBe(4)
       |                       ^
     10|   })
     11| })

──────────────────────────────────────────────────[1/1]─

 Test Files  1 failed | 1 passed (2)
      Tests  1 failed | 2 passed (3)
   Start at  23:39:27
   Duration  154ms (transform 27ms, setup 0ms, collect 46ms, tests 6ms, environment
```

Test Lifecycle

👉 **Orchestration** → Collection → Execution → Reporting

- Test files scheduling

Test orchestration

Find test files to run

- Configuration
  - `vite.config.ts`, `vitest.config.ts`

vitest.config.ts

```ts
export default defineConfig({
  test: {
    include: ["**/*.test.ts"],
    exclude: ["**/e2e/**"],
    projects: [
      {
        name: "unit",
        test: {
          include: ["**/*.unit.ts"],
        }
      },
      ...
    ]
  },
})
```

- CLI arguments to overrides

```
vitest src/add.test.ts src/dir/ # glob file pattern
vitest --project=unit # filter projects
vitest --shard=1/3 # parallelize across multiple machines
```

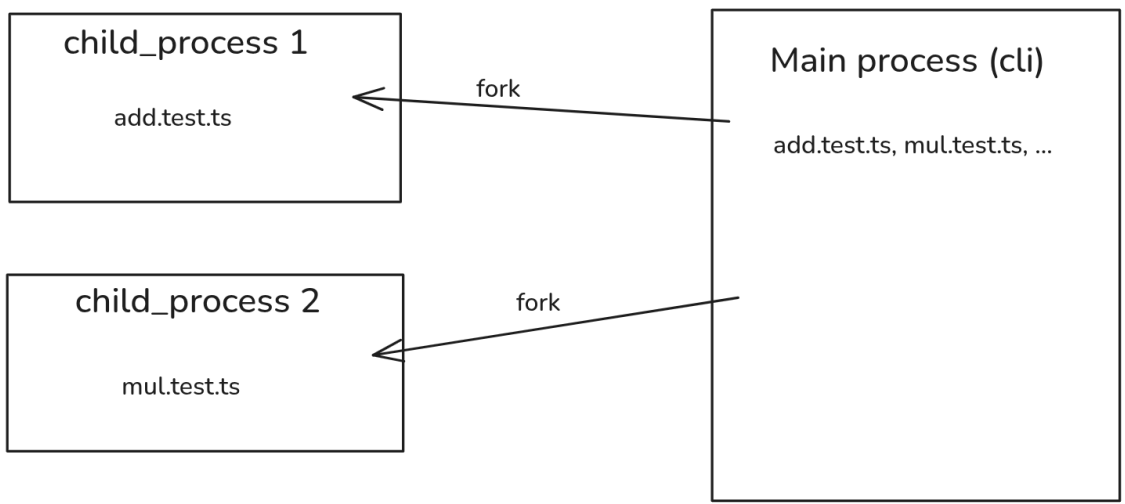
```
pool.runTests([
  {
    project: ...,
    moduleId: "/xxx/add.test.ts"
  },
  {
    project: ...,
    moduleId: "/xxx/mul.test.ts"
  }
])
```

Test orchestration / Pool

packages: `vitest`, ~~`tinypool`~~

- Spawn isolated runtime from main process and assign test files
- `pool: "forks"`, `"threads"`, `"vmThreads"`, ... + Browser mode
- The default is `pool: "forks"`

```
import { fork } from "node:child_process"
```

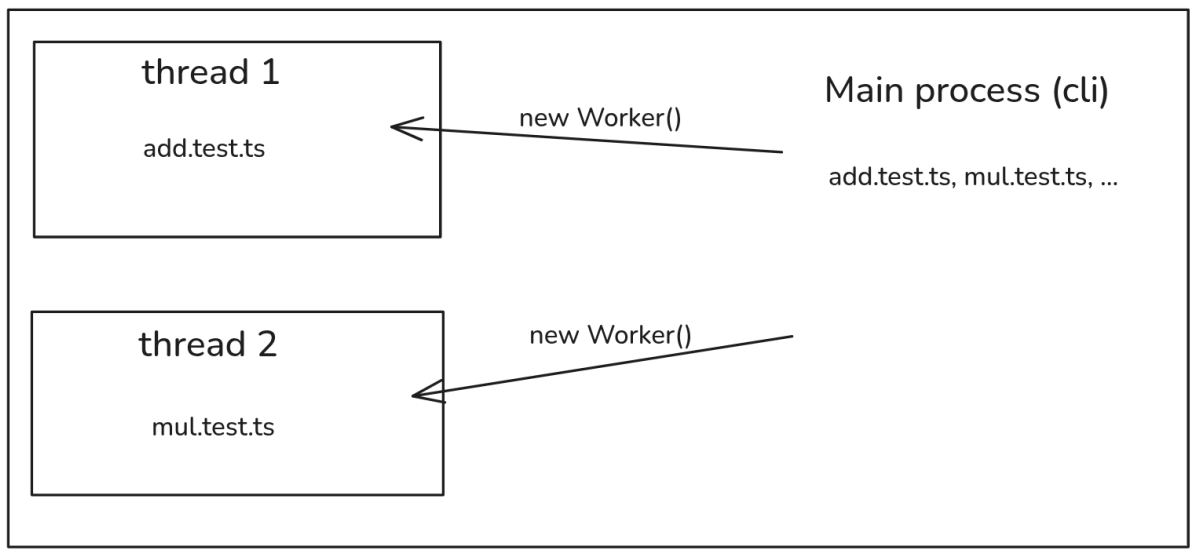


Test orchestration

- `pool: "threads"`

```
import { Worker } from 'node:worker_threads'
```

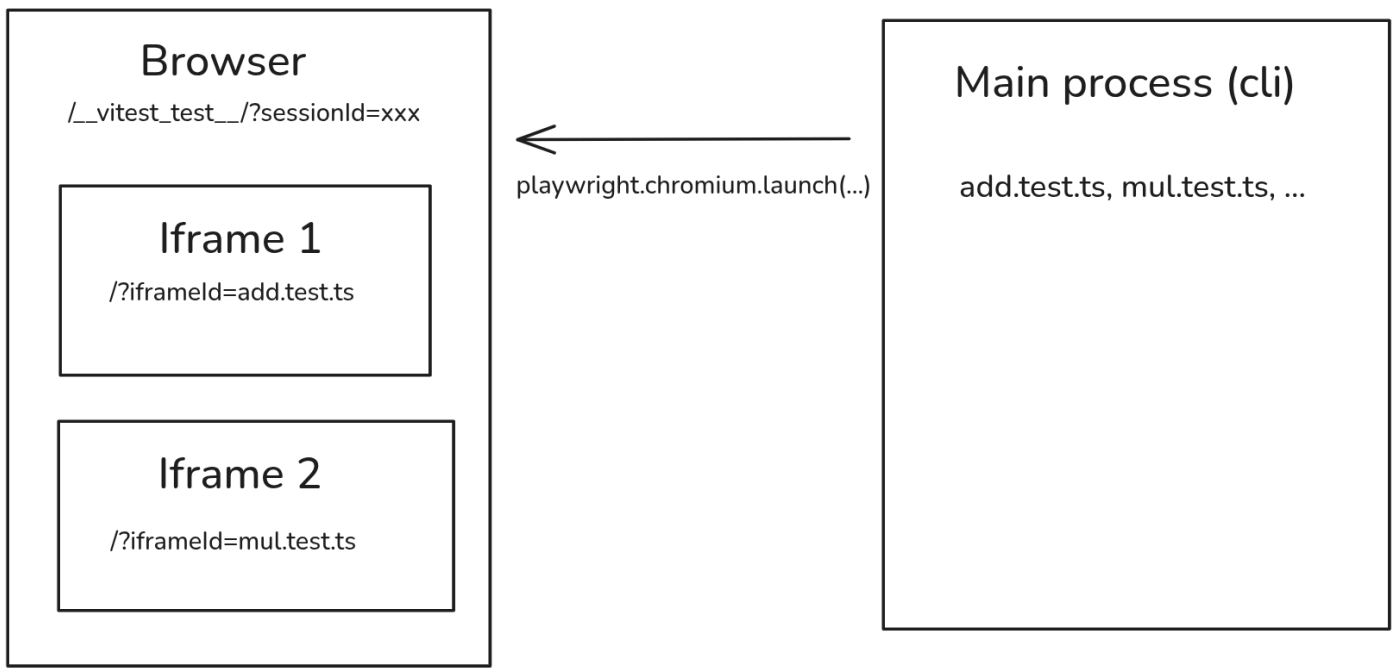
- Light weight compared to child process, however:
  - `process.chdir( ... )` is not available.
  - "less stable" (e.g. Native module / Node-API library compatibility - Common Errors)



Test orchestration

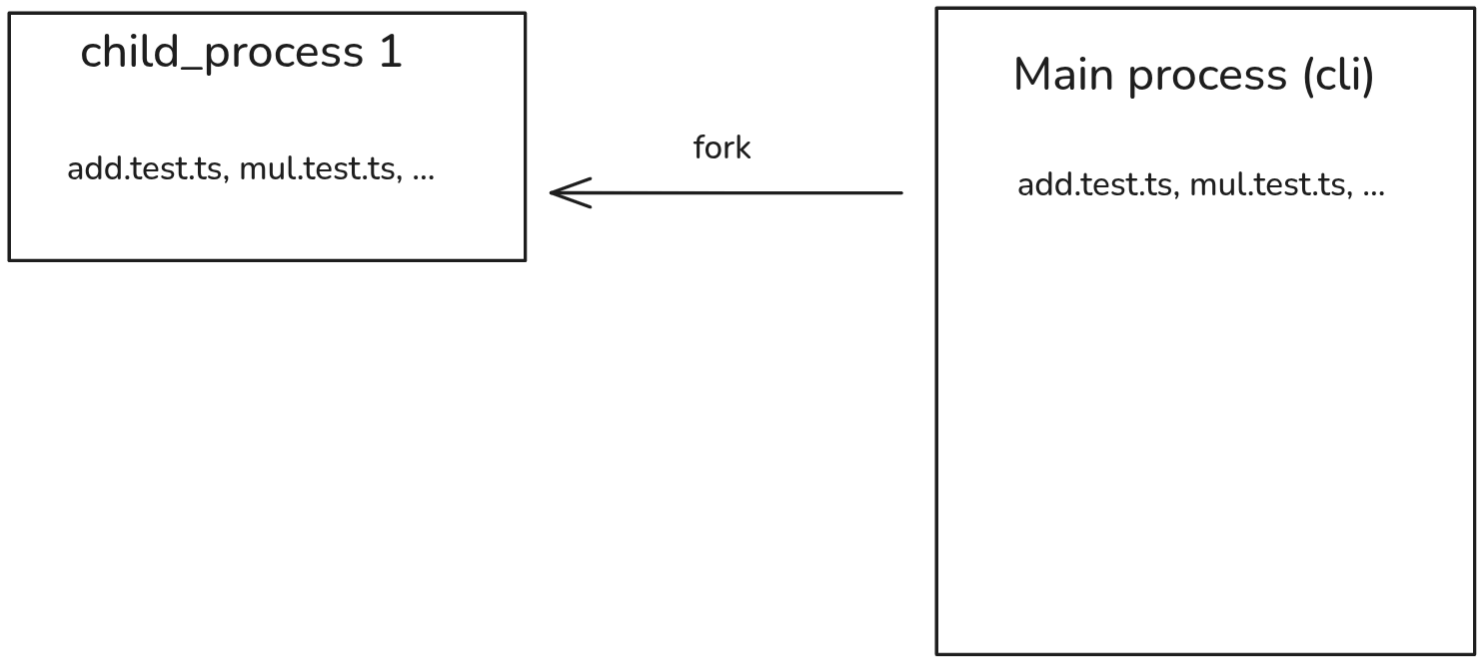
packages: `@vitest/browser-playwright`, `@vitest/browser-webdriverio`

- Browser Mode
- Using a single browser instance and a single page to reduce overhead



Test orchestration

- No isolation (`vitest --no-isolate` or `isolate: false`)



Module graph

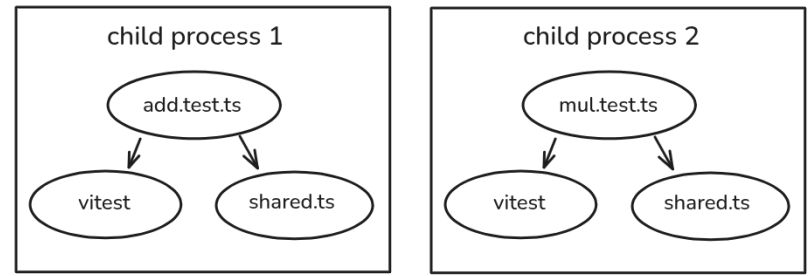
- Runtime's module graph is also reused, so it avoids evaluating same modules multiple times when shared by multiple test files.

```ts
// [add.test.ts]
import { test } from "vitest"
import { shared } from "./shared"

test("add", ...)
```

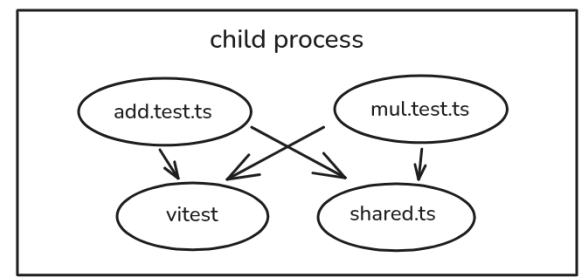
```ts
// [mul.test.ts]
import { test } from "vitest"
import { shared } from "./shared"

test("mul", ...)
```

```ts
// [shared.ts]
console.log("[shared.ts evaluated]")
export const shared = "shared";
```

isolate: true (default)



isolate: false



About isolation and pool

- Trade-off between `pool: "forks"`, `"threads"`, `"vmThreads"`
  - `forks` as default as it's closest to how the code is actually used.
- `isolate: false` to opt-out from isolation
  - Reusing existing child process / worker thread can save time to spawn for each test file.
  - This mode still allows splitting multiple test files into multiple pools for parallelization to benefit multiple CPUs.
  - Cons: Each test file can affect each other and non deterministic behavior is easier to manifest.
- Docs Improving Performance

```
export default defineConfig({
  test: {
    pool: 'threads', // default is 'forks'
    isolate: false, // default is true
  },
})
```

Test Lifecycle

Orchestration → 👉 **Collection** → Execution → Reporting

Collecting tests

- Main process only knows about test files.
- Execute **test files** to discover **test cases**

```
$ vitest list --json
[
  {
    "name": "named exports overwrite export all",
    "file": "/xxx/vite/packages/vite/src/node/ssr/__tests__/ssrLoadModule.spec.ts"
  },
  {
    "name": "buildStart before transform",
    "file": "/xxx/vite/packages/vite/src/node/ssr/__tests__/ssrLoadModule.spec.ts"
  },
  {
    "name": "module runner initialization > correctly runs ssr code",
    "file": "/xxx/vite/packages/vite/src/node/ssr/runtime/__tests__/server-runtime.spec.ts"
  },
  {
    "name": "mergeConfig > handles configs with different alias schemas",
    "file": "/xxx/vite/packages/vite/src/node/__tests__/config.spec.ts"
  },
  ...
```

Creating `Task` tree

package: `@vitest/runner`

- `describe()` → `Suite`, `test()` → `Test`

```ts
// [add.test.ts]
describe("add", () => {
  test('first', () => {
    expect(add(1, 2)).toBe(3)
  })
  test('second', () => {
    expect(add(2, 3)).toBe(4)
  })
})
```

```ts
type Task = File | Suite | Test
```

```
File(name: add.test.ts)
  Suite(name: add)
    Test(name: first)
      fn: () => { expect(add(1, 2)).toBe(3) }
      result: undefined
    Test(name: second)
      fn: () => { expect(add(2, 3)).toBe(4) }
      result: undefined
```

This phase often takes time since it involves executing import statements and evaluating dependency modules.

```
...
 Test Files  2 passed (2)
      Tests  3 passed (3)
   Start at  16:51:13
   Duration  130ms (transform 33ms, setup 0ms, collect 46ms, tests 3ms, environment 0ms, prepare 7ms)
                                          ^^^^^^^^^^^^^ 👆
```

Test Lifecycle

Orchestration → Collection → 👉 **Execution** → Reporting

Executing `Test`

packages: `@vitest/runner`, `@vitest/expect`, `@vitest/snapshot`, `@vitest/pretty-format`

- Finally execute each `Test` functions and record results.

```
File(name: add.test.ts)
  Suite(name: add)
    Test(name: first)
      fn: () ⇒ { expect(add(1, 2)).toBe(3) }
      result: { status: 'pass' }
    Test(name: second)
      fn: () ⇒ { expect(add(2, 3)).toBe(4) }
      result: { status: 'fail', errors: [Error('Expected 5 to be 4', diff="...")] }
```

- Depending on the nature of tests, a test execution time can be shorter than collection time.

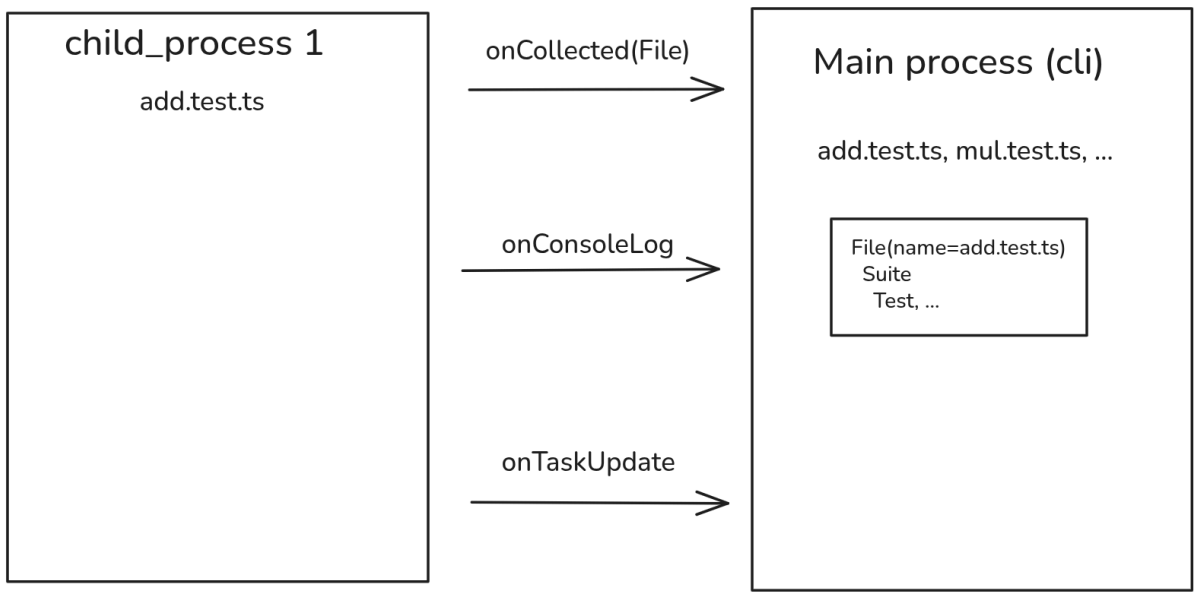
```
...
 Test Files  2 passed (2)
      Tests  3 passed (3)
   Start at  16:51:13
   Duration  130ms (transform 33ms, setup 0ms, collect 46ms, tests 3ms, environment 0ms, prepare 7ms)
                                                          ^^^^^^^^^^^ 👆
```

Test Lifecycle

Orchestration → Collection → Execution → 👉 **Reporting**

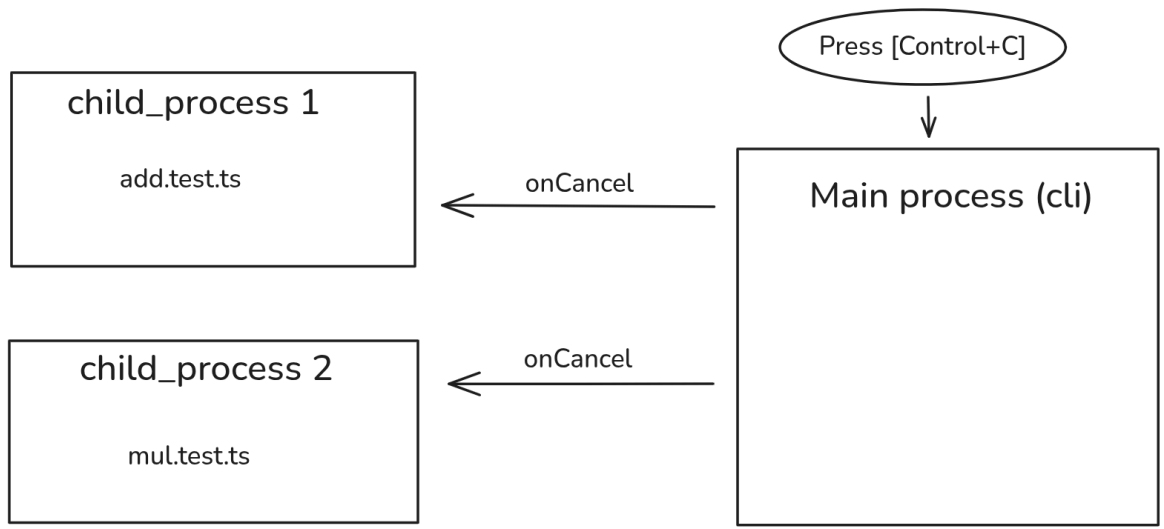
Reporting results

- `onCollected(files: File[])` notify collected `Task` tree
- `onTaskUpdate(pack: { id, result }[], ...)` notify test status incrementally in batch
- `onConsoleLog(log: ConsoleLog)` notify captured console logs during test run



Runtime communication

- birpc - Protocol agnostic typed two-way rpc library
- `node:child_process / fork` : `process.send` , `process.on("message")`
- `node:worker_threads / Worker` : `parentPort.postMessage` , `parentPort.on("message")`
- Browser mode: `Websocket`
- Support sending events from main process to test runner (e.g. gracefully abort tests on Control+C)



Reporter API

- Conveniently normalized data structure `TestModule` is provided instead of raw `Task` tree structure.
- Buildin reporters: `default`, `tree`, `dot`, `github-actions`, `json`, `junit`, `tap`, ...
- https://vitest.dev/advanced/api/reporters.html

```ts
// [custom-reporter.ts]
import { BaseReporter } from 'vitest/reporters'

export default class CustomReporter extends BaseReporter {
  onTestRunEnd(
    testModules: TestModule[],
    unhandledErrors: SerializedError[],
  ) {
    console.log(testModules.length, 'tests finished running')
    super.onTestRunEnd(testModules, unhandledErrors)
  }
}
```

```ts
// [vitest.config.ts]
import { defineConfig } from 'vitest/config'

export default defineConfig({
  test: {
    reporters: ['./custom-reporter.ts'],
  },
})
```

Example: Default reporter

```
——————————————— Failed Tests 1 ———————————————————

 FAIL  src/add.test.ts > add > second
AssertionError: expected 5 to be 4 // Object.is equality

- Expected
+ Received

- 4
+ 5

 ❯ src/add.test.ts:9:23
      7|   })
      8|   test('second', () => {
      9|     expect(add(2, 3)).toBe(4)
       |                      ^
     10|   })
     11| })

————————————————————————————————————————————[1/1]—

 Test Files  1 failed | 1 passed (2)
      Tests  1 failed | 2 passed (3)
   Start at  23:39:27
   Duration  154ms (transform 27ms, setup 0ms, collect 46ms, tests 6ms, environment 0ms, prepare 8ms)
```

Example: Github Actions Reporter

```
::error file={/xxx/add.test.ts},line={9},... ::AssertionError: expected 5 to be 4 ...
```

Test Lifecycle

Orchestration → Collection → Execution → Reporting

Done! ✅

Where is Vite?

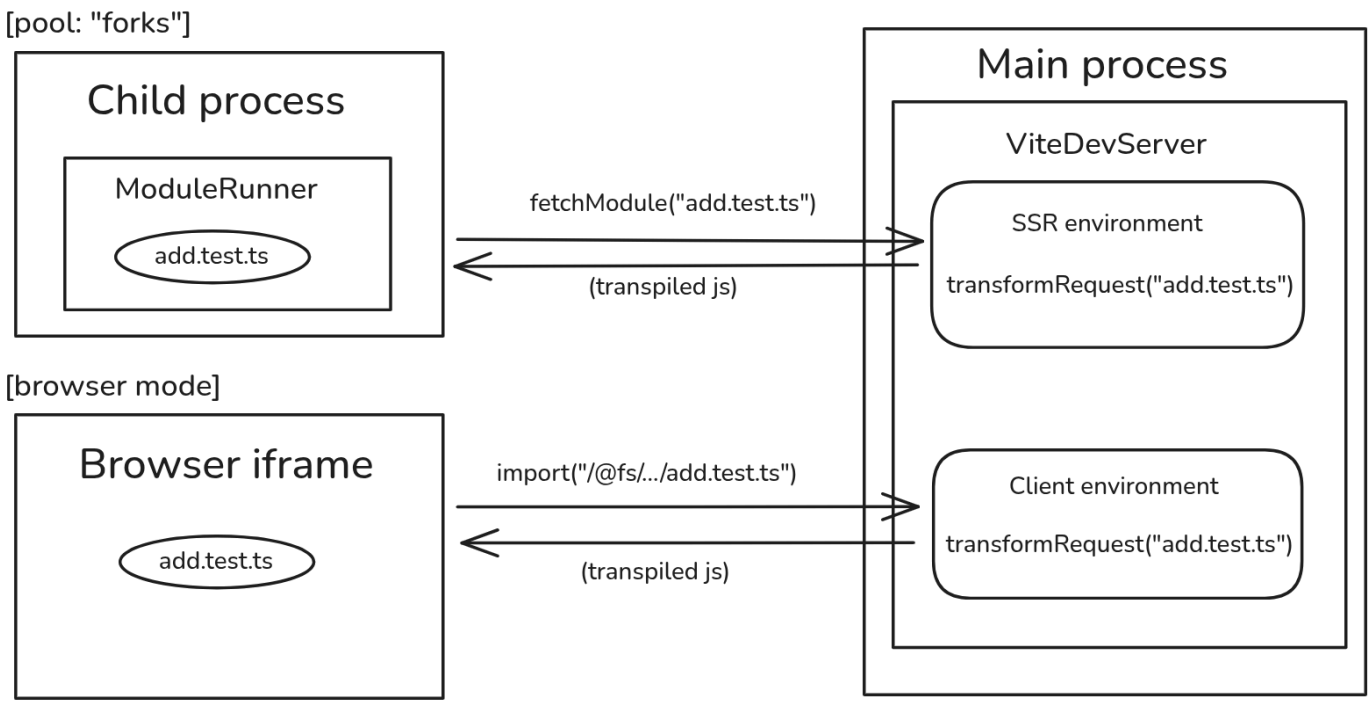
- We followed the entire lifecycle of test run, but how does Vite come into play?

```
...
 Test Files  2 passed (2)
      Tests  3 passed (3)
   Start at  16:51:13
   Duration  130ms (transform 33ms, setup 0ms, collect 46ms, tests 3ms, environment 0ms, prepare 7ms)
                  ^^^^^^^^^^^^^^^ 👆
```

Test runner and Vite environment API

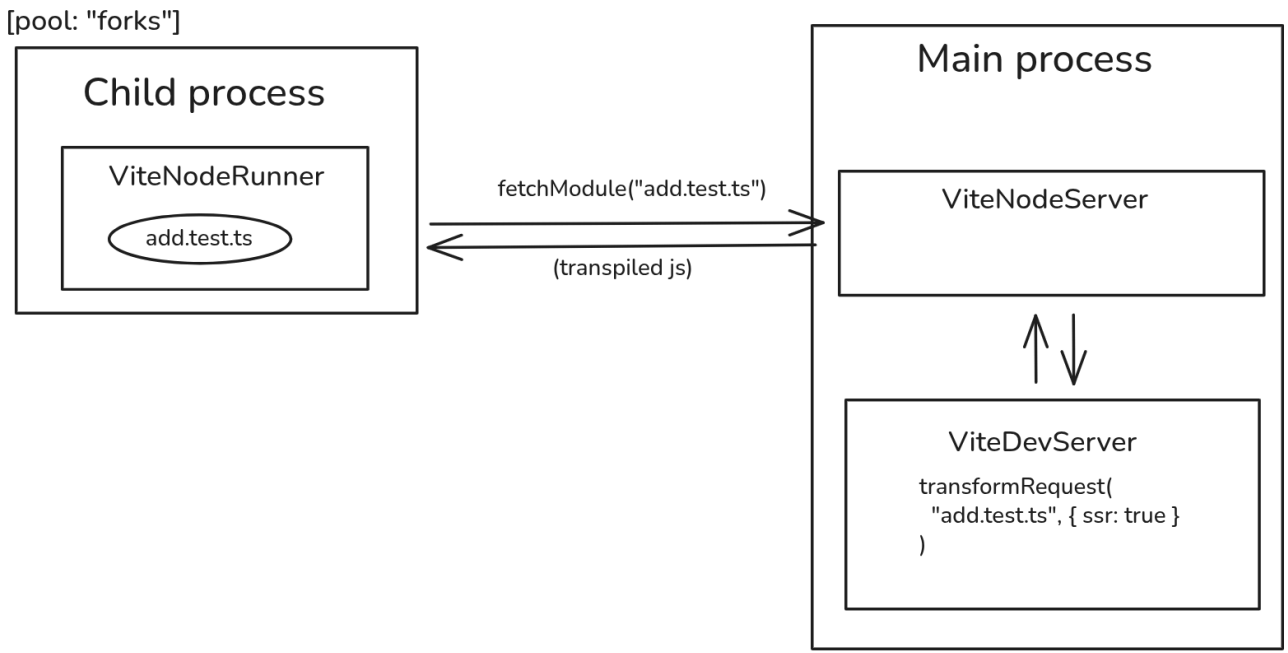
Client-server architecture

- Node test → SSR environment
- Browser mode → Client environment



`vite-node` → Vite environment API

- Historically, `vite-node` has been used to achieve the same architecture before Vitest 4.
- `import { ViteNodeRunner } from "vite-node/client"` on test runner
- `import { ViteNodeServer } from "vite-node/server"` on main process



Vite Module Runner

- "Vite module runner transform" rewrites original `import` and `export` into runtime functions.
  - `import` → `__vite_ssr_import__`, `__vite_ssr_dynamic_import__`
  - `export` → `__vite_ssr_exportName__`, `__vite_ssr_exportAll__`
- Run `VITE_NODE_DEBUG_DUMP=true vitest` (`VITEST_DEBUG_DUMP=.vitest-dump vitest` for Vitest 4)

```ts
// [src/add.test.ts]
import { test, expect } from "vitest"
import { add } from "./add"

test("add", () => {
  expect(add(1, 2)).toBe(3)
});
```

```ts
// [.vitest-dump/root/-src-add-test-ts]
const __vite_ssr_import_0__ = await __vite_ssr_import__("/xxx/node_modules/vitest/dist/index.js", ...);
const __vite_ssr_import_1__ = await __vite_ssr_import__("/src/add.ts", ...);

(0,__vite_ssr_import_0__.test)("add", () => {
  (0,__vite_ssr_import_0__.expect)((0,__vite_ssr_import_1__.add)(1, 2)).toBe(3);
});
```

Module mocking

packages: `@vitest/mocker`, `@vitest/spy`

- Auto-mocking `vi.mock("./add.js")`
  - import original module and deeply replace all exports with spies.
- Manual-mocking with factory `vi.mock("./add.js", () => ...)`
  - the original module is not imported but implementation is provided inline.

```
import { test, expect } from "vitest"
import { add } from "./add"
import { mul } from "./mul"

vi.mock("./add.js") // auto-mocking
vi.mock("./mul.js", () => ({ mul: vi.fn(() => 42) })) // manual-mocking

test("add", () => {
  expect(add(1, 2)).toBeUndefined()
  expect(add).toHaveBeenCalled()
  expect(mul(2, 3)).toBe(42)
  expect(mul).toHaveBeenCalled()
})
```

Module mocking with Module Runner

- Vitest transforms `vi.mock` to be at the top, so it's processed before `import`.

```
import { add } from "./add.js"

vi.mock("./add.js", () => ({ add: vi.fn(() => 42) }))

test("add", () => {
  expect(add(1, 2)).toBe(42)
})
```

```
// register mocking state before import
__vite_ssr_import_0__.vi.mock("./add.js", () => ({
  add: __vite_ssr_import_0__.vi.fn(() => 42)
}));
// import is intercepted by Vitest to implement mocking
const __vi_import_0__ = await __vite_ssr_dynamic_import__("/src/add.ts");

(0,__vite_ssr_import_0__.test)("add", () => {
  (0,__vite_ssr_import_0__.expect)(__vi_import_0__.add(1, 2)).toBe(42);
})
```

Key Takeaways

- Walked through test lifecycle to understand testing framework architecture
  - Orchestration → Collection → Execution → Reporting
- Reviewed how Vitest is powered Vite
  - Test runner leverages the same runtime mechanism as Vite's client and SSR application
  - Vite's transform pipeline and environment API provides a foundation