


Inside Vitest

Test Framework Architecture Deep Dive

About Me

- Hiroshi Ogawa [@hi-ogawa](#) 
- [Vite](#)  and [Vitest](#)  core team member
- Open Source Developer at [VoidZero](#) ()
- SSR meta-framework fanatic
- [Vite RSC support](#) [@vitejs/plugin-rsc](#) 



What is Vitest?

Unit testing framework

```
// packages/vite/src/node/__tests__/scan.spec.ts
import path from 'node:path'
import { describe, expect, test } 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 /home/hiroshi/code/others/vite
```

```
...
```

```
✓ Rollup logs of warn should be handled by vite 7ms
✓ onLog passed by user is called 9ms
✓ onwarn passed by user is called 9ms
✓ 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 agnostics
 - Node.js, Browser Mode, Cloudflare Workers

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

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

```
// [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?

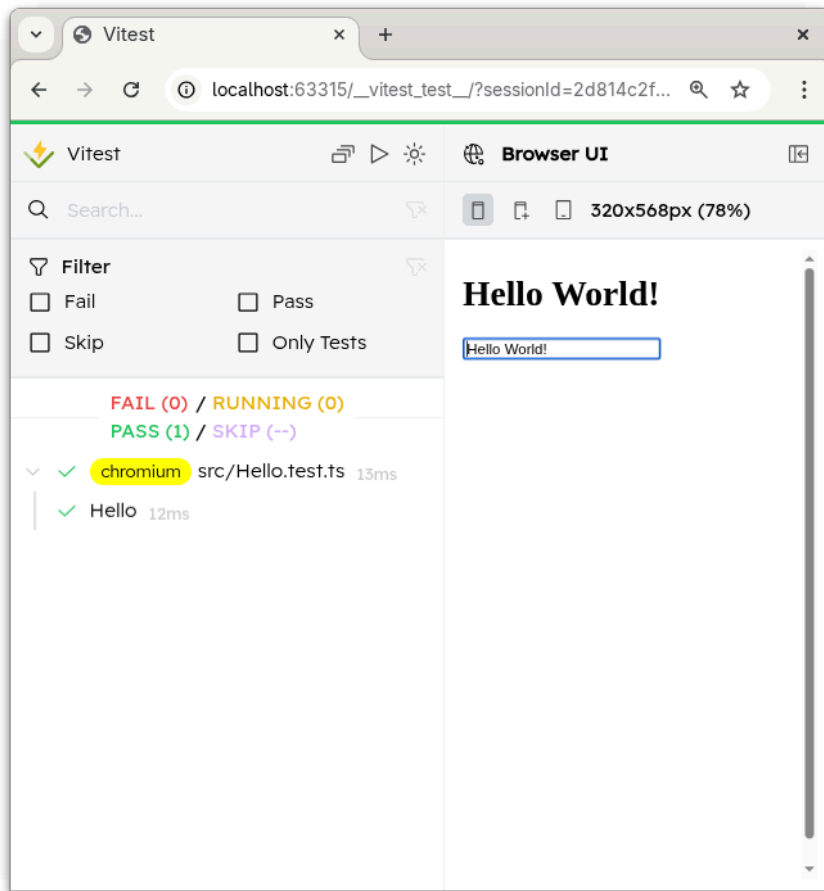
Runtime agnostic → Browser Mode

```
// [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()
})
```

```
// [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' }],
    },
  },
})
```



Overview

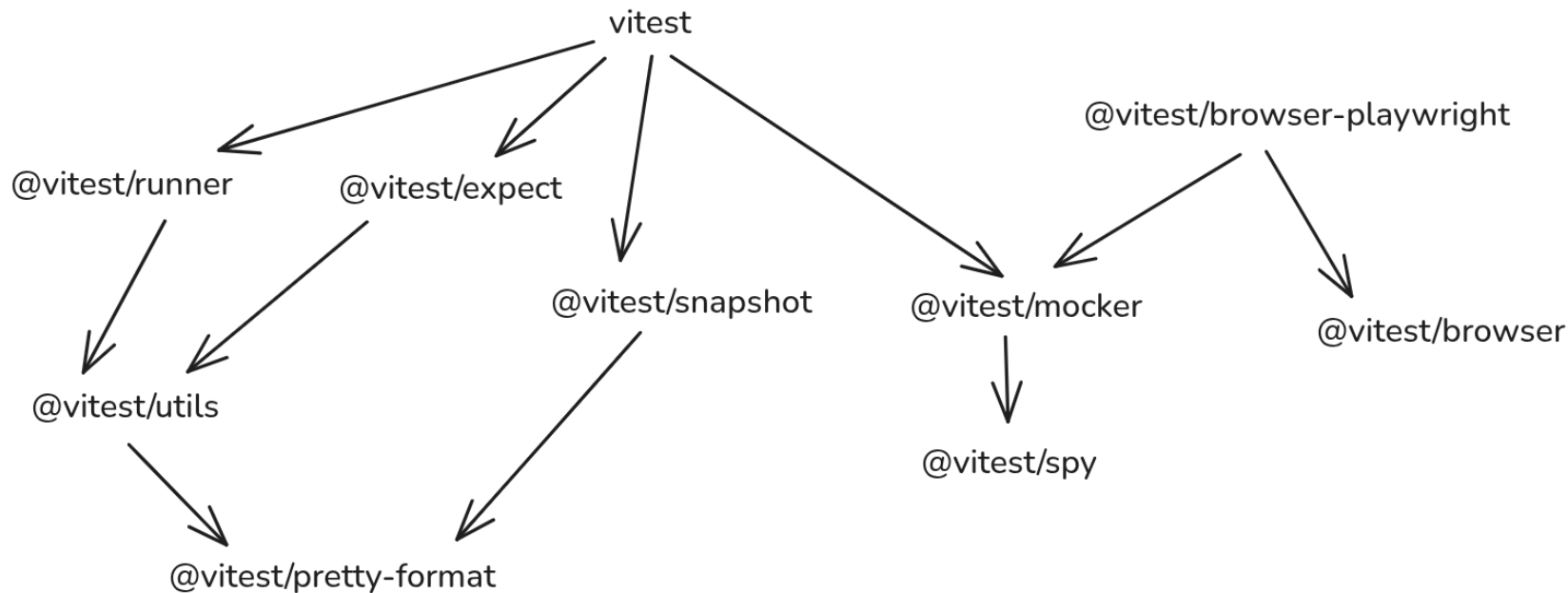
This talk follows the **test lifecycle** to explore Vitest architecture:

- Orchestration → Collection → Execution → Reporting

Along the way, we'll explore:

- Which parts are **general test framework** implementation
- How **Vite** powers test runtime (ModuleRunner, transform, etc.)
- How **monorepo packages** divide responsibilities

Vitest Monorepo Packages Dependencies



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, environmen
```

```
FAIL Tests failed. Watching for file changes ...
press h to show help, press q to quit
```


Finding test files to run

package: `vitest`

- CLI arguments (file pattern, overrides, etc.)

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

- Configuration

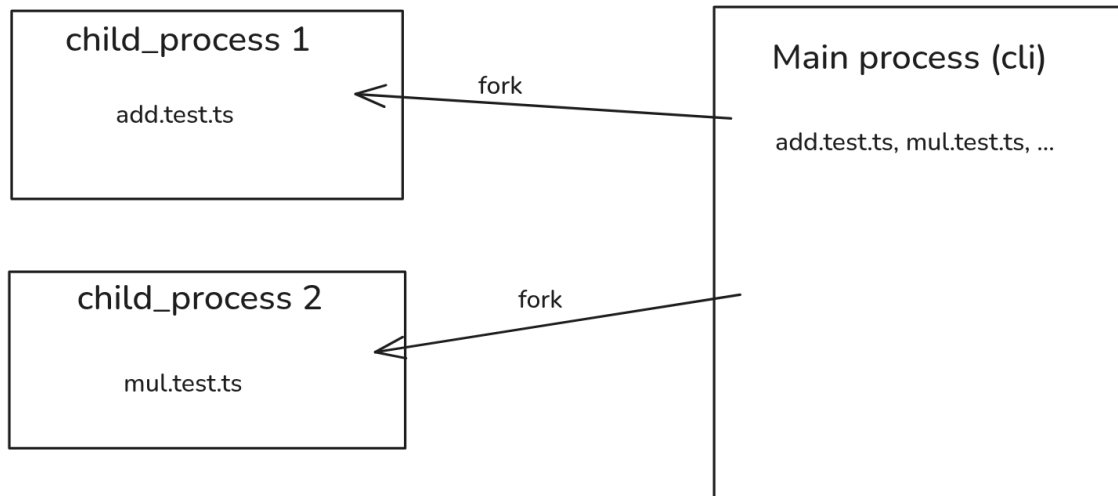
```
export default defineConfig({  
  test: {  
    dir: ... ,  
    include: ... ,  
    exclude: ... ,  
  },  
  projects: [  
    ...  
  ]  
})
```

Test runner orchestration

packages: `vitest`, `tinypool`

- Spawn isolated runtime from main process and assign test files
- The default is `pool: "forks"`

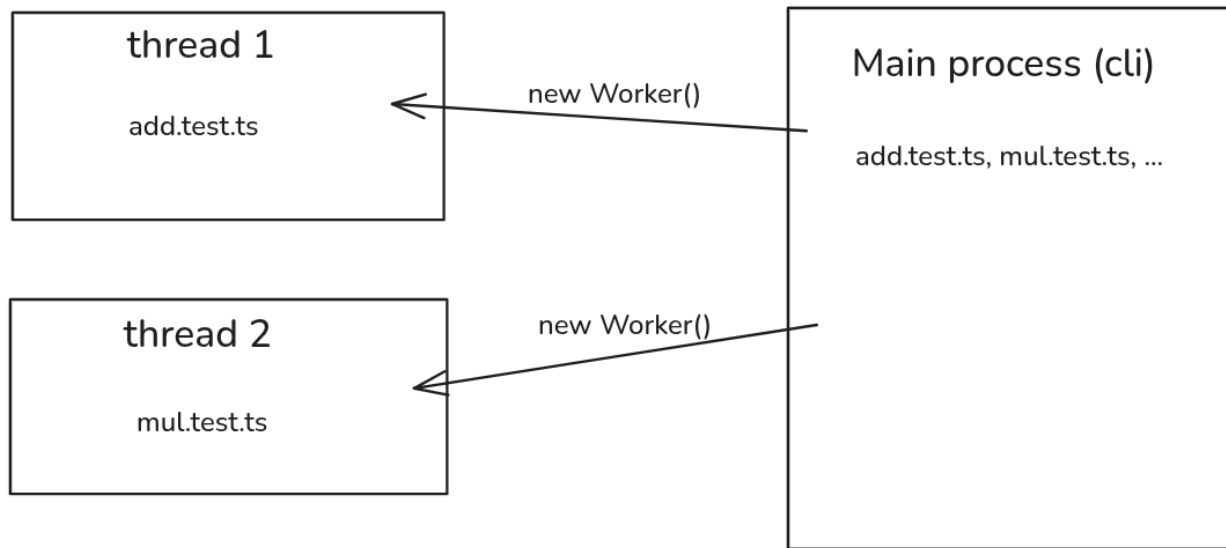
```
import { fork } from "node:child_process"
```



Test runner orchestration

- `pool: "threads"`

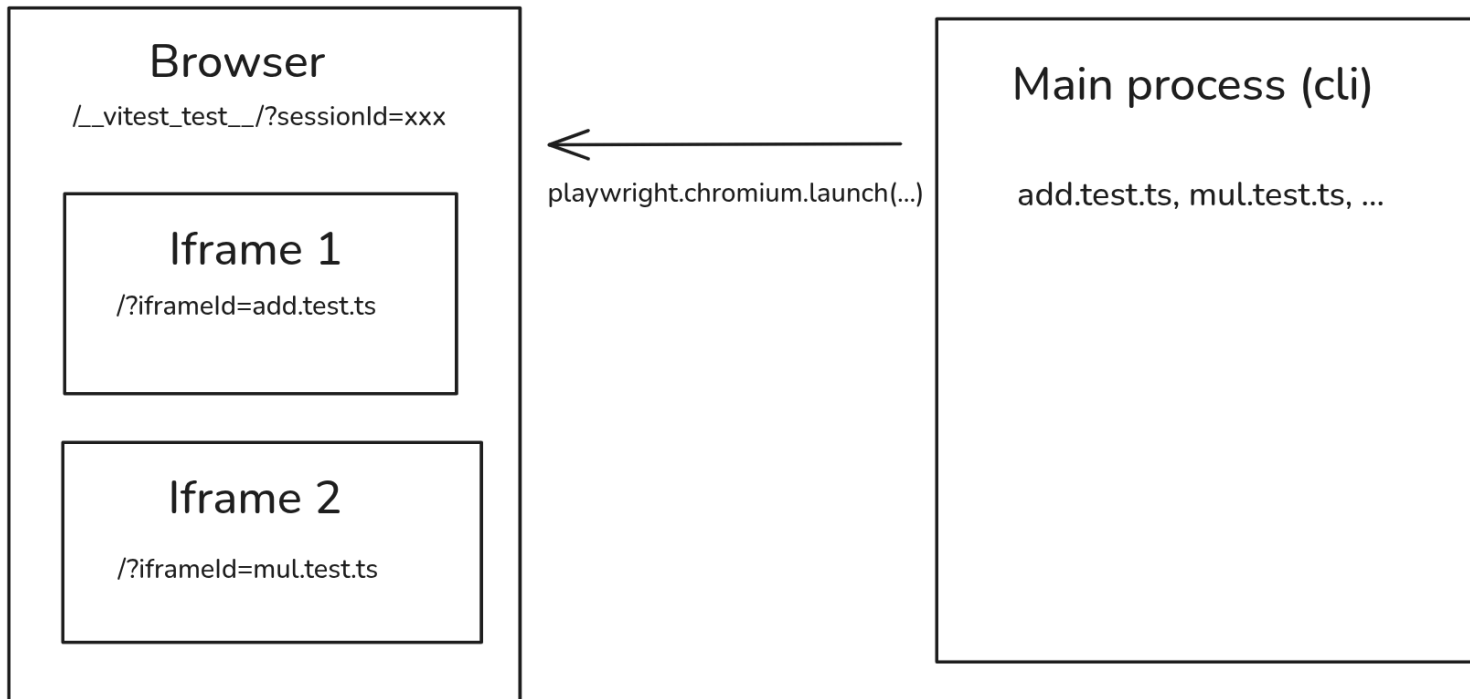
```
import { Worker } from 'node:worker_threads'
```



Test runner orchestration

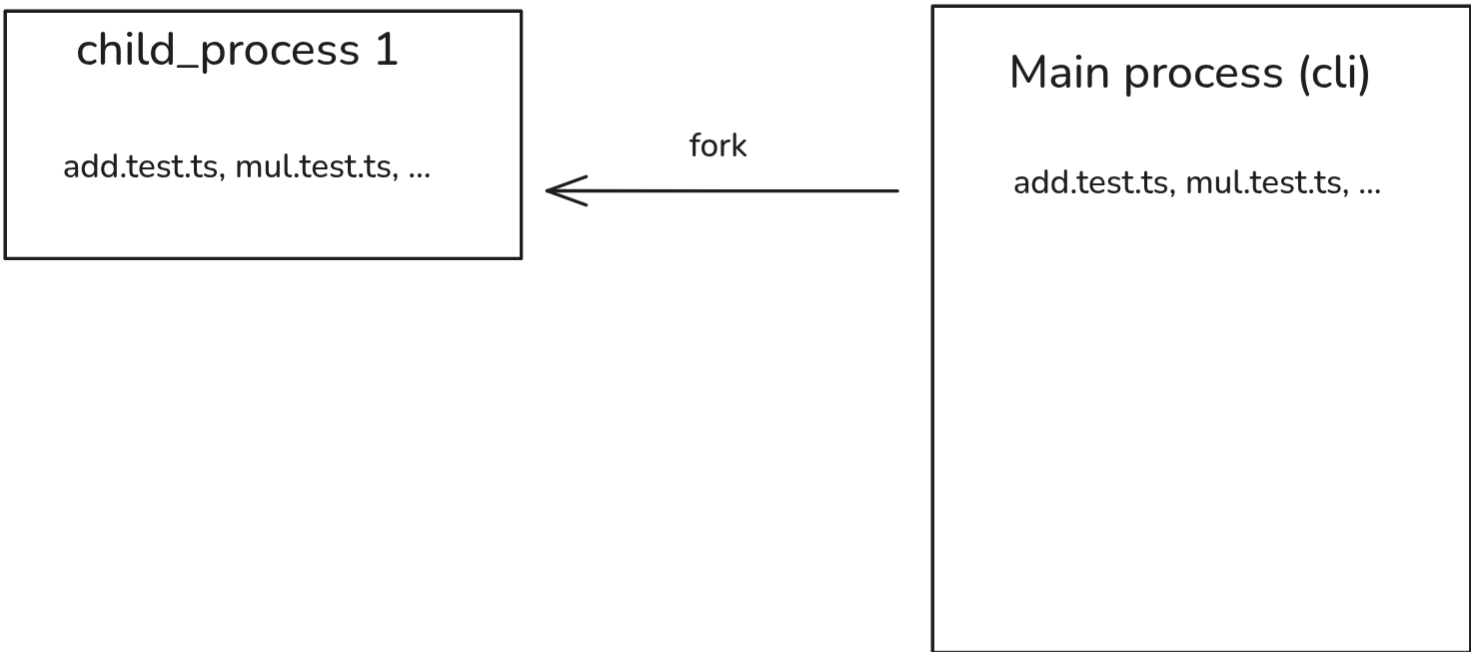
packages: `@vitest/browser-playwright`, `@vitest/browser-webdriverio`

- Browser Mode



Test runner orchestration

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



About isolation and pool

- `pool: "forks", "threads", "vmThreads"`
 - `forks` as default for stability
- `isolate: false` to opt-out from isolation
 - Reusing existing child process / worker thread can save time to spawn for each test file. Runtime's module graph is also reused, so it avoids evaluating same modules multiple times when shared by multiple test files.
 - This mode still allows splitting multiple test files into multiple pools for parallelization to benefit multiple CPUs.
- Docs Improving Performance

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

Isolation example

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

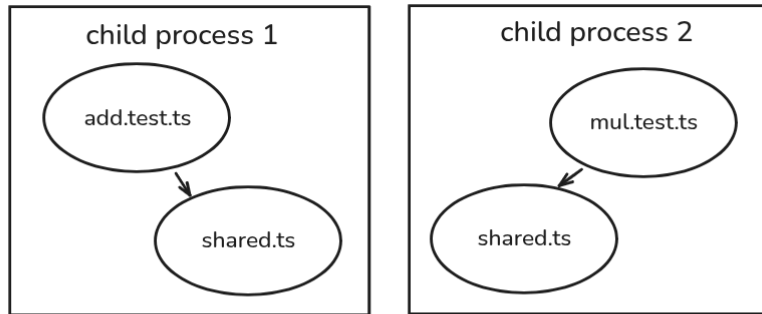
test("add", ...)
```

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

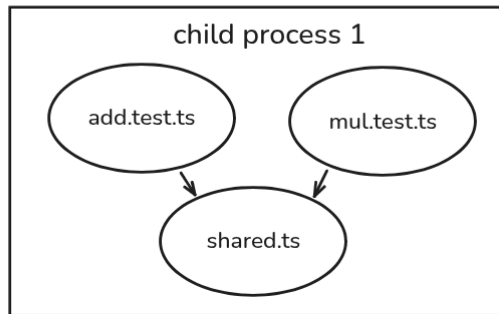
```
test("mul", ...)
```

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

isolate: true



isolate: false



Collecting tests

- Execute test files to collect test cases
- Main process only knows about test files.
- Let test runner discover test cases as it executes each test file.

Creating Task tree

package: @vitest/runner

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

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

```
File(id: 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
```

```
...
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)
```

^^^^^^^^^^^^^^ 🙌

Executing Test

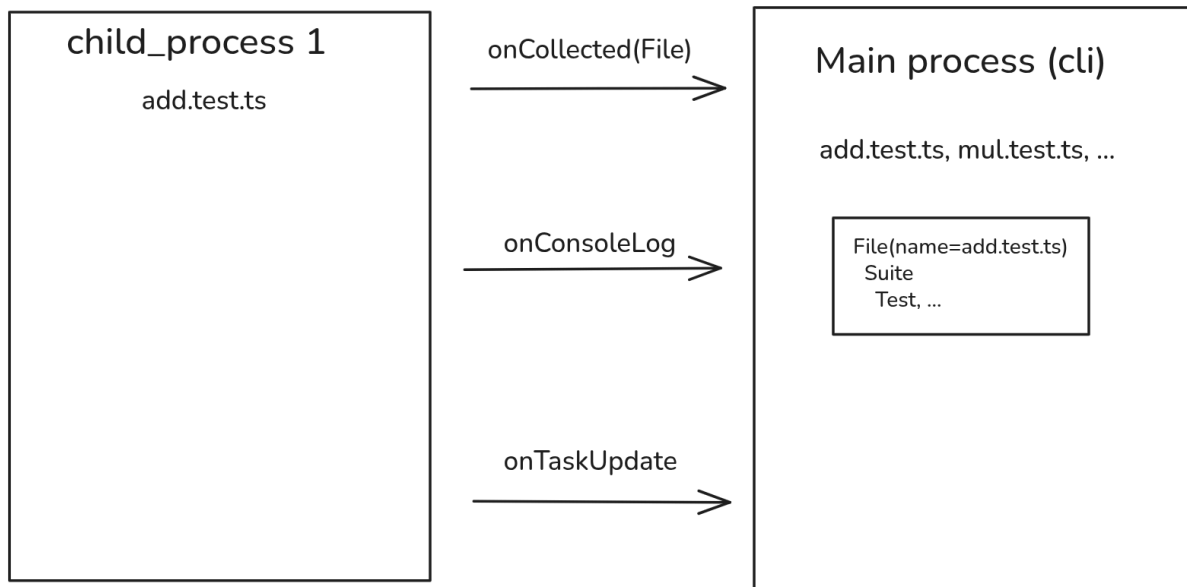
packages: @vitest/runner, @vitest/expect, @vitest/snapshot, @vitest/pretty-format

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

```
...
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)
          ^^^^^^^^^^ 🙌
```

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



Reporter API

- Conveniently normalized data structure `TestModule` is provided instead of raw `Task` tree structure.
- <https://vitest.dev/advanced/api/reporters.html>

```
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)
  }
}
```

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 Action Reporter

11 ■■■■ 2025-10-25/examples/lifecycle/src/add.test.ts

Viewed

... .. @@ -0,0 +1,11 @@

1 + import { test, expect, describe } from "vitest"

2 + import { add } from "../add"

3 +

4 + describe("add", () => {

5 + test('first', () => {

6 + expect(add(1, 2)).toBe(3)

7 + })

8 + test('second', () => {

9 + expect(add(2, 3)).toBe(4)

✖ Check failure on line 9 in 2025-10-25/examples/lifecycle/src/add.test.ts

GitHub Actions / test

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

Where is Vite?

...

Test Files 2 passed (2)

Tests 3 passed (3)

Start at 16:51:13

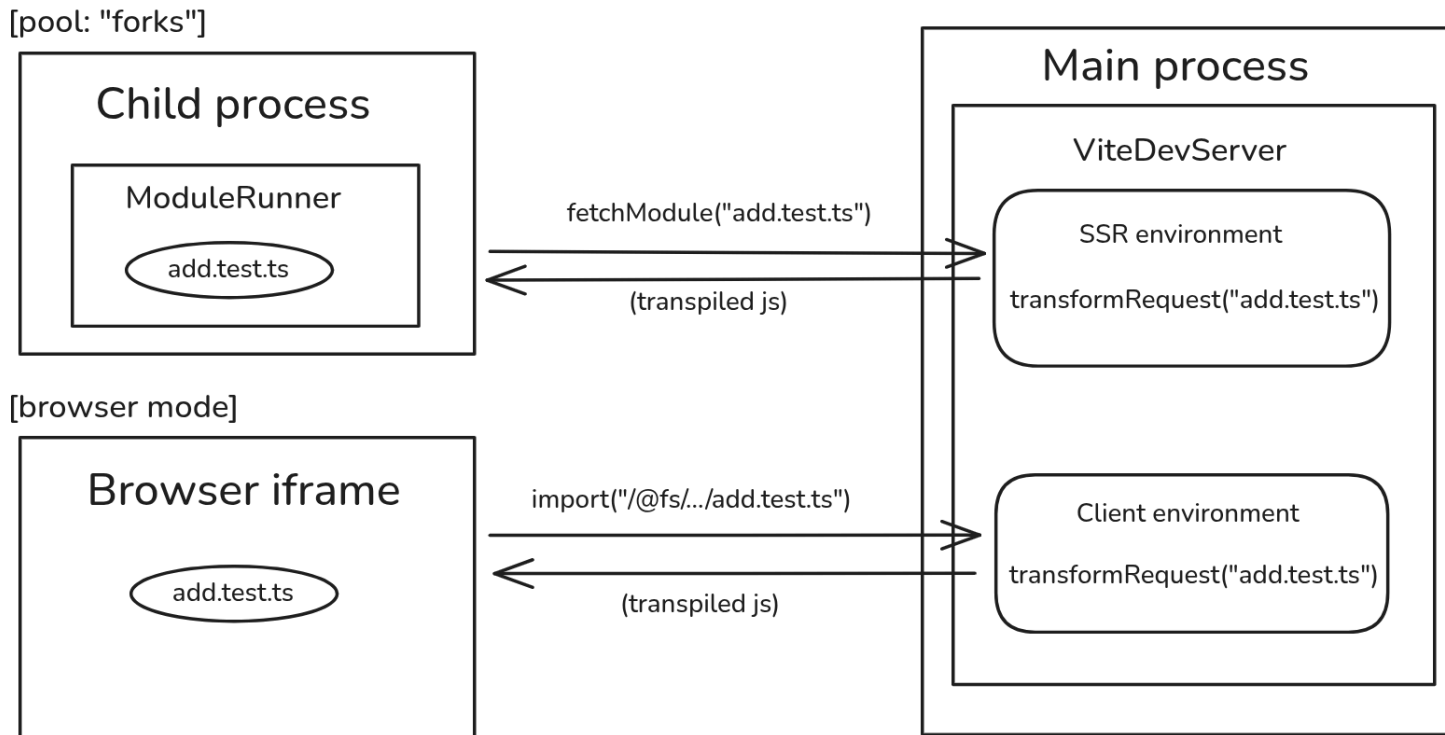
Duration 130ms (transform 33ms, setup 0ms, collect 46ms, tests 3ms, environment 0ms, prepare 7ms)

^^^^^^^^^^^^^^^^ ^



Test runner and Vite environment API

Client-server architecture



SSR / Client environment

- Vue SFC transform by `@vitejs/plugin-vue`
- Vite module runner transform is additionally applied for SSR
- Vue SFC Playground

```
<script setup>
import { ref } from 'vue'
const msg = ref('Hello World!')
</script>

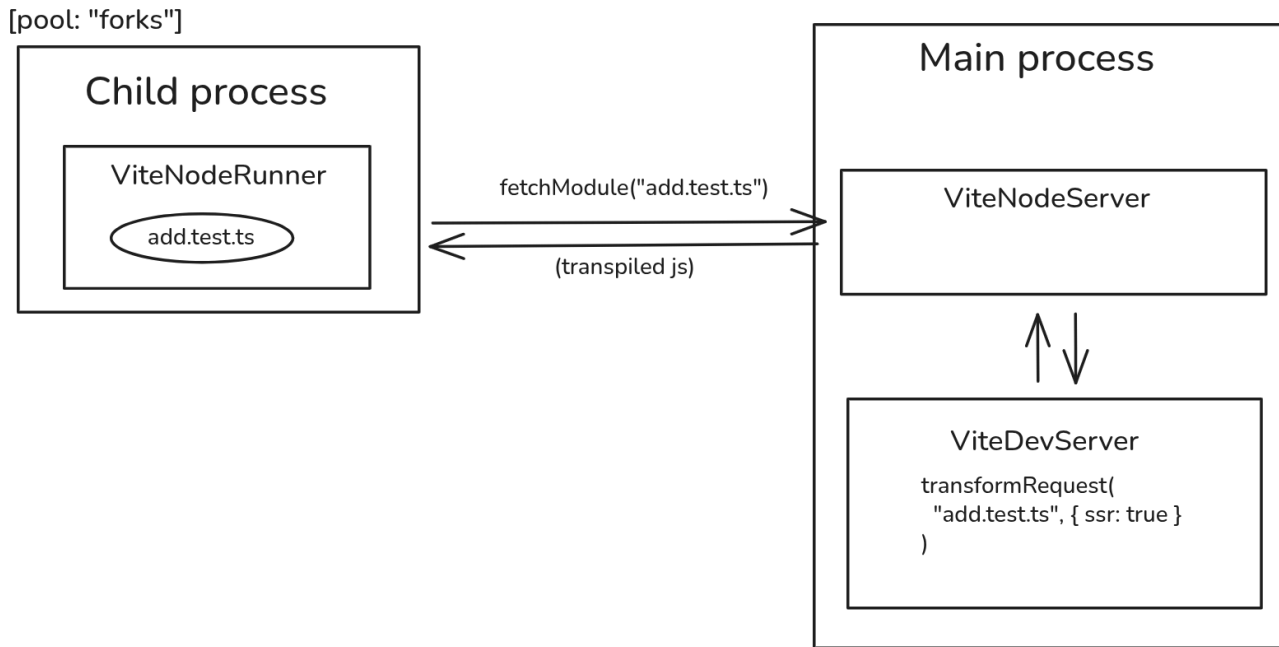
<template>
  <h1>{{ msg }}</h1>
  <input v-model="msg" />
</template>
```

```
// [clientEnvironment.transformRequest( ... )]
import { ref } from "/xxx/vue.js?v=7756971e"
...
const _sfc_main = { __name: 'Hello', setup(__props, { expose: __expose }) { ... } }
...
function _sfc_render(_ctx, _cache, $props, $setup, $data, $options) {
  return (_openBlock(), _createElementBlock(_Fragment, null, [
    _createElementVNode("h1", null, _toDisplayString($setup.msg), 1 /* TEXT */),
    _withDirectives(_createElementVNode("input", {
      "onUpdate:modelValue": _cache[0] || (_cache[0] = $event => (($setup.msg) = $event.value)),
      null, 512 /* NEED_PATCH */), [
        [_viewModelText, $setup.msg]
      ]
    )], 64 /* STABLE_FRAGMENT */))
}
...
export default /*#__PURE__*/_export_sfc(_sfc_main, [['render', _sfc_render], ['__file',
```

```
// [ssrEnvironment.transformRequest( ... )]
__vite_ssr_exportName__("default", () => { try { return __vite_ssr_export_default__
const __vite_ssr_import_0__ = await __vite_ssr_import__("/xxx/node_modules/vue/index
...
const _sfc_main = { __name: 'Hello', setup(__props, { expose: __expose }) { ... } };
...
function _sfc_ssrRender(_ctx, _push, _parent, _attrs, $props, $setup, $data, $options) {
  _push(`<!--><h1>${
    (0, __vite_ssr_import_1__.ssrInterpolate)($setup.msg)
  }</h1><input${
    (0, __vite_ssr_import_1__.ssrRenderAttr)("value", $setup.msg)
  }><!-->`);
}
...
const __vite_ssr_export_default__ = /*#__PURE__*/(0, __vite_ssr_import_3__.default)(_
```

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



Test runner

- `@vitest/runner` defines an interface

```
// packages/runner/src/types/runner.ts
interface VitestRunner {
  // how to process test files (entry points)
  importFile(filepath: string, ...): Promise<unknown>

  // Callbacks for each test lifecycle
  onBeforeRunTask(test: Test): unknown
  onAfterRunTask(test: Test): unknown
  ...
}
```

- Vite module runner

```
// packages/vitest/src/runtime/runners/test.ts
class VitestTestRunner implements VitestRunner {
  moduleRunner: VitestModuleRunner
  async importFile(filepath: string, ... ) {
    return this.moduleRunner.import(filepath)
  }
}

class VitestModuleRunner extends ModuleRunner {
  // override implementation for module mocking, etc.
}
```

- Browser mode

```
// packages/browser/src/client/tester/runner.ts
class BrowserVitestRunner implements VitestRunner {
  async importFile(filepath: string, ... ) {
    await import(filepath) // request to Vite dev server
  }
}
```

Vite Module Runner

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

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

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

```
// [.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", () => ({ add: vi.fn(() => 42) })) // manual-mocking

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

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

- **Test lifecycle drives architecture:** Understanding orchestration, collection, execution, and reporting is fundamental to test framework design
- **Client-server architecture:** Test runner (client) communicates with main process (server) to achieve runtime-agnostic execution (Node.js, Browser, etc.)
- **Vite as a foundation:** Module runner + transform pipeline powers test runtime, similar to how Vite handles SPA / SSR