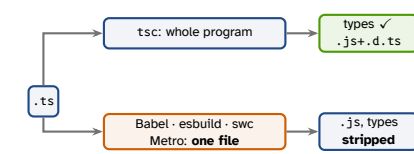


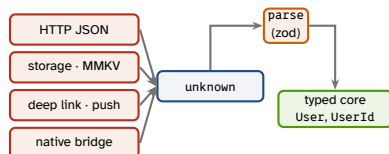
In one line: Types are **erased**: the compiler proves consistency *inside* the program, so the senior's job is the **edges** — a strict tsconfig, module settings that match whoever actually **loads** the code, **runtime validation** where data enters, and a build that stays fast as the codebase grows.

1 who compiles your .ts?



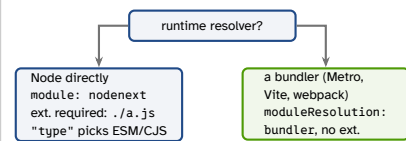
one file at a time can't tell if {foo} is a type → isolatedModules / verbatimModuleSyntax: say import type
RN/Vite: bundler emits, tsc --noEmit checks in CI

2 erased types → validate at the edge



res.json() as User: a claim, not a check
inside the core: no casts, no any, brands mint only in the parser

3 who loads the imports?



both honour package.json "exports";
a library targets nodenext (works everywhere)

How it works — config

- **strict** = noImplicitAny, noImplicitThis, alwaysStrict, strictNullChecks, strictFunctionTypes, strictBindCallApply, strictPropertyInitialization, useUnknownInCatchVariables (4.4), strictBuiltinIteratorReturn (5.6). New members join on upgrade — pin TS versions.
- **Not** in strict, worth it: noUncheckedIndexedAccess (a[i]: T | undefined), exactOptionalPropertyTypes, noImplicitOverride, noImplicitReturns, noFallthroughCasesInSwitch.
- **Modules**: nodenext models Node exactly — .mts/.cts and "type": "module" decide ESM vs CJS per file; relative imports name the *output* (.util.js; 5.7 rewriteRelativeImportExtensions allows .ts). bundler (5.0) + module: esnext/preserve (5.4): no extensions. **Interop**: esModuleInterop emits helpers so import x from 'cjs' gets module.exports.
- **verbatimModuleSyntax** (5.0): what you write is what is emitted — import type {A} vanishes, import {type A} leaves import {} from 'a' (side effect kept). isolatedModules: forbids what a per-file transpiler can't do (type re-exports w/o export type, ambient const enum).
- **Declarations**: .d.ts = types only. Untyped lib: declare module 'x' (bare = all any) or @types/* (DefinitelyTyped); "types" limits auto-included globals; skipLibCheck: faster, hides clashes.
- **Project references**: composite + references; tsc -b rebuilds only stale projects (.tsbuildinfo) and enforces layering (Swift: SPM targets).
- **Speed**: --extendedDiagnostics, --generateTrace; interfaces over big &; annotate exported returns (isolatedDeclarations 5.5: parallel .d.ts); tame mega-unions. TS 7 = native Go port (previewed 2025).

Example — the edges

```

const User = z.object({ id: z.string(), age: z.number() });
type User = z.infer<typeof User>; // one source of truth
async function loadUser(res: Response): Promise<User> {
  return User.parse(await res.json()); // throws on bad shape
}
declare const tag: unique symbol; // brand: nominal, 0 bytes
type UserId = string & { readonly [tag]: 'UserId' };
const toUserId = (s: string) => s as UserId; // the one cast
const Dir = { Up: 'up', Down: 'down' } as const;
type Dir = (typeof Dir)[keyof typeof Dir]; // 'up' | 'down'
// globals.d.ts -- has export, so it is a module: augment
export {};
declare global { var __APP_VERSION__: string }
// assets.d.ts -- no import/export: a script, ambient modules
declare module '*.png' { const id: number; export default id; }
  
```

Enums vs unions

- enum emits a runtime object; numeric ones add a reverse map (Object.values gives names *and* numbers); a string enum rejects the bare literal 'up' — nominal-ish.
- **const enum** inlines, but breaks per-file transpilers and package boundaries. 5.8 erasableSyntaxOnly bans enums, namespaces, parameter properties (Node type stripping). Default: literal union / as const.

Interview traps

- ! JSON.parse and Response.json() return any: the lie spreads silently. Type the result unknown, then parse.
- ! catch (e) is unknown under strict: e instanceof Error before e.message.
- ! paths aliases are *not* rewritten in output — Metro/Jest/Node need the same alias (babel-plugin-module-resolver, moduleNameMapper).
- ! declare global in a file with no import/export is an error: add export {}.
- ! @ts-ignore hides every future error on that line; @ts-expect-error (3.9) *fails* once the error is gone — self-cleaning.

JS → TS migration

allowJs (mixed build) → checkJs// @ts-check + JSDoc → rename leaves first (utils, API client) upward; type the **boundaries** first — most value. Turn strict on early with a @ts-expect-error baseline (or a stricter tsconfig for converted dirs); lint no-explicit-any, track the count down.

Remember

Strict inside · parse at the edge · config matches the loader.

Likely questions

1. node16/nodenext vs bundler? — who resolves imports at runtime.
2. Why verbatimModuleSyntax? — per-file transpilers can't know what is a type.
3. Nominal IDs? — a brand: string & { [tag]: 'UserId' }.
4. Why validate if we have types? — they're erased; the server/storage can lie.
5. Slow tsc? — trace it, split with project references, annotate exported returns, shrink mega-unions.