How to Think in Manaknight (Mental Model)

Manaknight is not “Node but safer” or “Rust but simpler”.

It’s a **capability-based, deterministic execution language**.

> If something can change the world (IO, time, randomness), **the compiler must see it**.

Everything below is framed as:

> “Here’s how you do this in Node / Go / Rust / WASM — here’s the *equivalent* Manaknight way.”

Mutation → Value Replacement

❌ Node / Go / Rust

```
let x = 5;
x = x + 1;
```

✅ Manaknight

```
let x = 5
let y = x + 1
```

Why: Mutation breaks auditability, replay, and reasoning. Manaknight forces **explicit state evolution**.

Global State → Explicit Data Flow

❌ Node / Go

```
global.cache[userId] = user;
```

✅ Manaknight

```
function updateCache(cache: Map<Int, User>, user: User): Map<Int, User> {
  set(cache, user.id, user)
}
```

Why: No hidden dependencies. Every function is reviewable in isolation.

Time & Random → Explicit Effects

❌ Node

```
Date.now()
Math.random()
```

❌ Go / Rust

```
time.Now()
rand.Int()
```

✅ Manaknight

```
effect time
effect random

function now() uses { time } {
  time.now()
}
```

Why: Time and randomness are *sources of nondeterminism*. Manaknight treats them as **capabilities**, not utilities.

IO Anywhere → IO Only Where Declared

❌ Node

```
fetch(url)
```

❌ Go / Rust

```
http.Get(url)
```

✅ Manaknight

```
effect http

function fetchUser(id: Int) uses { http } {
  http.get("/users/" + id)
}
```

Rule: Pure functions **cannot** call effectful functions. Undeclared IO = compile error.

Exceptions → Typed Results

❌ Node / Go / Rust

```
throw new Error("fail");
```

✅ Manaknight

```
Result<User, Error>
```

```
match result {
  ok(user)  -> user.name
  err(e)    -> "error"
}
```

Why: No hidden control flow. Errors are **data**, not side channels.

Classes & Inheritance → Algebraic Data Types

❌ Node / Java / Rust Traits

```
class Payment {}
class Card extends Payment {}
```

✅ Manaknight

```
type Payment {
  | Card(number: String)
  | Wire(ref: String)
}
```

Why: ADT + pattern matching = exhaustive, auditable logic.

Reflection & Metaprogramming → Explicit Structure

❌ Node / Java / Rust

```
obj[propName]
reflect.TypeOf(x)
```

✅ Manaknight

Not allowed.

Why: Reflection destroys static guarantees and audit trails.

Concurrency → Deterministic Isolation

❌ Go / Rust / WASM

- goroutines
- threads
- shared memory
- atomics

✅ Manaknight

- No shared mutable state
- One request = one isolated VM
- Parallelism is a **runtime concern**, not a language feature

Why: Determinism > throughput for critical logic.

Dynamic Code → Forbidden by Design

❌ Node / WASM

```
eval(code)
import(moduleName)
```

✅ Manaknight

Compile-time only. No dynamic loading. No runtime surprises.

Mental Shift Summary

If you want to…	In Node / Go / Rust	In Manaknight
Track state	Mutate	Create new values

If you want to…	In Node / Go / Rust	In Manaknight
Do IO	Anywhere	Explicit effects
Handle errors	Exceptions	Result types
Model variants	Classes / traits	ADTs
Run APIs	Frameworks	Language feature
Be auditable	Hope	Guaranteed