CONCEVOIR ET EXÉCUTER DES SYSTÈMES AGENTIQUES

L'architecture cible : le cadre de décision, du langage à l'évaluation, pour qu'ils tiennent en production.

ARCHITECTURE CIBLE DES SYSTÈMES AGENTIQUES

Thibault Souris.

L'architecture d'abord, le modèle ensuite.

Commencer simple, isoler par contrat, complexifier seulement sur preuve.

Note de l'auteur

Pendant deux ans, l'IA générative a été vécue comme une rupture : on s'émerveillait qu'une machine sache écrire, résumer, coder. Ce temps est passé. La question n'est plus de savoir si la technologie impressionne, mais comment l'intégrer pour de bon dans des processus, des métiers, des organisations. Et là, le constat est brutal : presque toutes les applications se disent « agentiques », presque aucune ne tient vraiment en production. La grande majorité des projets pilotes échouent à franchir ce pas, et toujours pour les mêmes raisons : on ne sait pas évaluer un comportement non déterministe, on ne sait pas le gouverner, on ne sait pas dire à l'avance quand il se trompe. Ce ne sont pas des problèmes de modèle. Ce sont des problèmes d'architecture.

« Adopter l'IA » recouvre d'ailleurs deux réalités qu'on confond trop souvent. Donner à des équipes des assistants génératifs pour rédiger ou analyser relève de l'acculturation : utile, mais relativement simple et sans grand risque structurel. Intégrer des systèmes agentiques dans des processus, existants ou à inventer, est tout autre chose : il faut repenser les flux, la confiance, la gouvernance, la fiabilité. C'est cette seconde adoption, la plus difficile, qui est ici en jeu ; et la confusion entre les deux explique une bonne part des désillusions.

Le signe que l'enjeu a basculé du modèle vers l'intégration tient dans une décision que les plus grands fournisseurs de modèles viennent de prendre : plutôt que de livrer un modèle de plus, ils montent des structures de déploiement et envoient des ingénieurs s'installer chez leurs clients. Quand ceux qui fabriquent les modèles concluent que le goulot n'est plus le modèle mais son intégration dans le réel, la cause est entendue : la valeur se joue autour du modèle, pas dedans.

Trop d'équipes, pourtant, choisissent encore un framework avant d'avoir pensé leur architecture, et paient ce choix des mois durant. La conviction qui porte ces pages est l'exact inverse : ce n'est pas le modèle qui dicte l'architecture, c'est l'architecture qui encadre le modèle. Le modèle se loue et se remplace ; l'architecture se possède et se capitalise. Ce renversement admis, le reste (langage, topologie, mémoire, évaluation, sécurité) cesse d'être un pari initial pour devenir une suite de décisions réversibles, prises derrière des contrats stables.

Ces pages s'adressent à quiconque conçoit, exécute ou défend de tels systèmes, et veut le faire sur des bases qui durent.

Thibault Souris · 2026

SOMMAIRE

PARTIE I 4 SECTIONS

CADRE · DÉCOUPLAGE · LANGAGE · DÉCOUPAGE

Les fondations

Le socle : pourquoi l'architecture encadre le modèle, et comment le découplage rend langage et topologie réversibles.

DANS CETTE PARTIE

SECTION 01

Cadre, lecture et principe directeur

Le passage de la démonstration à l'exécution se joue sur une question d'ingénierie : comment faire tourner un système agentique fiable sans se lier à une stack, à un fournisseur ou à une topologie qu'il faudra défaire plus tard ?

Le principe directeur, rappelé

Le principe fondateur est simple : c'est l'architecture qui encadre le modèle, pas l'inverse. En découle une conséquence pratique forte : **le langage et la topologie de déploiement sont des décisions d'adaptateur**, prises derrière des contrats stables, et non des engagements structurants pris au premier jour. Le domaine métier, lui, ne change pas quand la stack change.

> Corollaire opérationnel : on ne choisit pas une stack puis une architecture. On pose d'abord les frontières (ports, contrats, protocole d'intégration), ce qui rend ensuite le choix de langage et de découpage réversible et local.

De quoi parle ce document, et de quoi il ne parle pas

Les mots prêtent à confusion, et la confusion coûte cher. L'intelligence artificielle est le champ entier. L'IA générative en est un sous-ensemble : elle produit du contenu, et un assistant qui rédige ou résume en relève. Un système agentique, lui, ne se contente pas de générer : il planifie, appelle des outils, agit en boucle vers un but, avec mémoire et une autonomie bornée. Il utilise un modèle génératif comme composant, mais il se définit par son orchestration, pas par sa génération. À l'autre bord, l'automatisation classique, dite RPA, est déterministe et scriptée, donc cassante au moindre changement.

Le flou est réel : un enchaînement figé autour d'un seul appel de modèle n'est pas de l'agentique, c'est de l'automatisation augmentée ; un agent sur rails fixes est proche de la RPA. L'agentique commence avec le raisonnement non déterministe, les outils, l'autonomie et la mémoire ; entre le chemin figé et la délégation dynamique, il y a tout un spectre.

Ce document traite l'architecture technique de ces systèmes agentiques, et leur intégration dans des processus, existants ou nouveaux. Il ne traite pas de l'acculturation des personnes aux assistants génératifs : équiper des équipes est un autre sujet, essentiel mais distinct. C'est l'intégration en exploitation, là où la plupart des projets échouent, qui est ici en jeu. Il laisse de même de côté deux questions voisines et complémentaires : l'arbitrage construire ou acheter de chaque couche, et le mappage à des textes réglementaires précis. Ce sont des couches qui se posent par-dessus, pas des angles morts : les principes qui les gouvernent, briques partagées, souveraineté graduée, registre des usages, sont posés en section 15.

Leur paysage d'offres, mouvant et daté par nature, est volontairement laissé hors de ces pages : la doctrine vise ce qui tient quels que soient les fournisseurs et les juridictions.

Un dernier cadrage, essentiel : ce document décrit l'architecture cible, complète, celle vers laquelle on tend, et non un préalable à bâtir d'un seul bloc. Le plancher d'un premier système tient en peu de choses : un orchestrateur unique, un port de modèle, une persistance, quelques garde-fous et de l'observabilité. Tout le reste, ingénierie fine de la mémoire, externalisation de l'état, multi-agent, boucle d'évaluation, ne s'ajoute que lorsqu'un déclencheur objectif ou un gain mesuré le justifie. La matrice de décision (section 13) et les recommandations par horizon (section 14) disent ce qu'on pose au jour zéro et ce qu'on diffère. La richesse de ces pages n'est donc pas une complexité imposée d'emblée : c'est une cible vers laquelle on avance par paliers, chaque brique justifiée par une preuve.

Ce qui est hérité, ce qui est neuf

La plomberie de cette doctrine n'est pas inventée ici : journal immuable et rejeu, découverte de service, compensation, idempotence, suspension et reprise sont des patrons de systèmes distribués éprouvés depuis des décennies. C'est une force : on hérite de patrons validés à grande échelle, et l'on ne paie le coût de conception qu'une fois, sur ce qui est propre au composant. Car quatre choses, elles, n'existaient pas dans le distribué classique, et justifient une doctrine propre. Le cœur est non déterministe : un garde-fou ne peut pas être un outil que le modèle choisit d'appeler, il s'impose au middleware (sections 05 et 12). L'entrée non fiable est indistincte de l'instruction : le contenu récupéré se traite comme une donnée, jamais comme un ordre (section 11). La mémoire et l'oubli deviennent de l'ingénierie : transformer un modèle sans état en agent à état exige une consolidation et un oubli explicites (section 06). Et l'agentivité porte un rayon d'impact inédit : un agent compromis agit vite et de façon irréversible, d'où l'autonomie bornée et l'approbation sur l'action (sections 09, 11 et 13).

Les questions traitées

QUESTION D'ÉQUIPE	TRAITÉE EN
Full TypeScript, ou un back en Python, Rust ou Go ?	Sections 02 et 03
Monolithe ou microservices, et quand découper ?	Section 04
Comment ça tourne réellement, et comment ça tient en charge ?	Sections 05, 07 à 09
Comment l'agent se souvient, et oublie sans se trahir ?	Section 06
Comment garder la confiance : observabilité, sécurité, évaluation ?	Sections 10 à 12
Et ce qui est partagé entre plusieurs applications ?	Section 15

Les hypothèses retenues

- Un système agentique utile en production est d'abord une affaire d'architecture autour du modèle, pas de framework.
- Les meilleurs systèmes utilisent des patterns simples et composables, et n'ajoutent de la complexité que lorsqu'elle améliore réellement le résultat.
- Chaque décision lourde doit rester réversible, parce qu'elle est isolée derrière un contrat.

À RETENIR

- Un cadre de décision, pas une stack imposée.
- Langage et topologie sont des choix d'adaptateur, réversibles, pris après les frontières.
- On ajoute de la complexité seulement quand elle se justifie par le résultat.
- La plomberie est héritée du distribué ; quatre nouveautés justifient une doctrine propre : cœur non déterministe, entrée indistincte de l'instruction, oubli gouverné, fort rayon d'impact.

SECTION 02

Le découplage qui rend langage et topologie secondaires

Avant de parler langage et déploiement, il faut comprendre pourquoi ces choix deviennent secondaires. Deux frontières y suffisent : les ports du domaine, et un protocole d'intégration standardisé entre processus.

Frontière 1, les ports du domaine

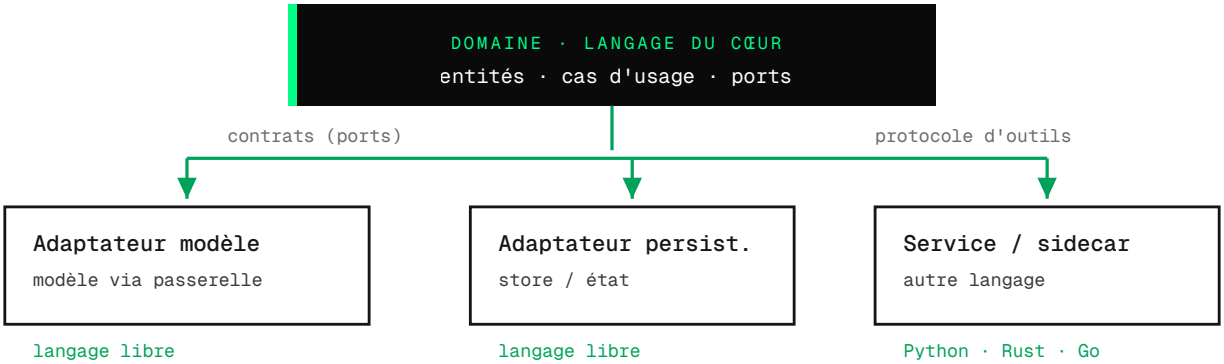
Le domaine exprime ses besoins comme des contrats (ports) : fournir une génération de modèle, persister un état, exécuter une action, lire une source. Chaque contrat est implémenté par un adaptateur. L'adaptateur peut être écrit dans n'importe quel langage tant qu'il honore le contrat. Le langage devient un détail d'implémentation d'adaptateur. Ce découplage porte un nom, l'architecture hexagonale, ou « ports et adaptateurs », formalisée par Alistair Cockburn en 2005. L'appliquer au modèle lui-même, traité comme un adaptateur parmi d'autres, est une idée que l'écosystème partage de plus en plus ; ce qui suit n'en revendique pas la paternité, mais la pousse jusqu'à ses conséquences, de l'évaluation à la mémoire, de la sécurité à la gouvernance de contrats.

Cette dernière idée mérite d'être poussée jusqu'au bout, car elle fonde la durabilité de tout l'édifice. Le moteur placé derrière le port modèle n'est lié que par son contrat, pas par sa nature. C'est aujourd'hui un modèle de langage ; ce pourrait être demain un moteur de raisonnement d'un autre type, et l'architecture y survivrait sans réécriture, du moment qu'il honore le même contrat. Ce qui resterait alors à traiter n'est pas la frontière mais les propriétés du moteur : tant qu'il raisonne de façon non déterministe sur des entrées ouvertes, les exigences d'évaluation, de mémoire et de sécurité décrites plus loin tiennent à l'identique. Autrement dit, le modèle n'est pas le centre que l'architecture organise, c'est un composant qu'elle peut remplacer, jusqu'à son paradigme. Soyons justes, toutefois, sur le bénéfice quotidien de cette frontière : il tient moins au remplacement effectif du modèle, plus rare qu'on ne le dit, qu'à la capacité de tester et de raisonner sur le domaine sans dépendre du modèle. La substituabilité est l'assurance que l'on garde en réserve ; la testabilité est le dividende de tous les jours. Et une précision qui vaut pour tout le reste : le port est propre pour le contrat typé, mais la frontière comportementale, elle, ne se garantit pas par le type, elle se valide (section 12).

Frontière 2, le protocole d'intégration entre processus

Quand une capacité vit dans un autre processus, voire un autre langage, elle est exposée via un protocole d'outils standardisé. Le système consomme ce processus comme un fournisseur d'outils, et expose lui-même ses cas d'usage de la même manière. La frontière inter-langage est nette et stable.

Schéma, le langage et la topologie derrière les contrats



> Le cœur ne sait pas dans quel langage tournent les adaptateurs.

Conséquence pratique

Le débat full TypeScript contre back Python, Rust ou Go cesse d'être un dilemme d'architecture. C'est un arbitrage local, pris adaptateur par adaptateur, selon l'écosystème de la capacité concernée. Le même raisonnement vaut pour le découpage en services : un service est un adaptateur qui a migré dans son propre processus, sans que le domaine change.

Le contrat, au-delà du type

Un contrat ne se réduit pas à un schéma typé : il se gouverne, il porte du sens, et il doit composer avec l'existant. Trois facettes d'une même exigence.

Il se gouverne. Un contrat stable n'est pas un contrat figé. Tout contrat, port du domaine, message échangé ou schéma de persistance, évolue ; sans discipline, il dérive et casse ses consommateurs en silence, le mal classique des architectures orientées services. Quatre règles le gouvernent. Il porte une version. Son évolution est additive par défaut, et le consommateur est un lecteur tolérant : il ignore ce qu'il ne connaît pas et accepte l'absence d'un champ optionnel, ce qui rend l'ajout non cassant. Un changement réellement cassant ne se fait pas d'un bloc, mais en deux temps, étendre puis contracter : on introduit le nouveau, on migre les consommateurs, on retire l'ancien une fois que plus personne ne l'utilise. Enfin, la provenance, qui enregistrait déjà ce qu'un prompt a vu et ce que la mémoire a fourni, s'étend aux contrats et aux événements : savoir qui produit, qui consomme et sous quelle version permet de mesurer l'impact d'un changement avant de le faire. La vérification, elle, appartient au consommateur (section 12) : il exprime ses attentes sous forme de contrat exécutable, et un changement du producteur qui les briserait échoue à l'intégration, avant d'atteindre la production.

Les messages et les événements qui franchissent une frontière portent une enveloppe stable, séparée de leur charge utile : une identité, un type qualifié, une référence de schéma, un horodatage, une origine. Cette taxonomie est ce qui permet à un consommateur de router, filtrer et tracer sans connaître le détail du producteur.

Il porte du sens. Un contrat honoré dans sa forme peut encore désigner des choses différentes. « Client », « compte », « engagement », « validation » n'ont pas le même sens d'un système à l'autre, ni d'un agent à l'autre. L'interopérabilité difficile n'est pas « comment s'appeler » mais « parle-t-on de la même chose ». La réponse n'est pas un modèle de données central qui dirait tout pour tout le monde : un tel modèle devient vite ingérable, lent à faire évoluer, et finit contourné. C'est, par contexte, un langage commun explicite, celui que partagent le métier et le code, des frontières nettes entre contextes, et un vocabulaire minimal partagé aux seules jointures. L'adaptateur y joue son second rôle : au-delà de traduire un protocole, il aligne le sens, exactement comme la couche anticorruption traduit le dialecte d'un système hérité. Cet alignement compte doublement pour un système agentique : faute de sémantique fixée, un modèle rapprochera de force des concepts voisins mais distincts et inventera des correspondances fausses. Le contrat sémantique est, contre ces alignements hallucinés, ce que le schéma typé est contre la donnée malformée.

Il compose avec l'existant. Sur du neuf, le port est une frontière nette et l'adaptateur est mince. Sur un système existant, la frontière n'est pas donnée : ses contrats sont implicites, sa sémantique non documentée. L'adaptateur devient alors une couche anticorruption qui traduit le dialecte du système hérité dans le langage du domaine et empêche le modèle ancien de contaminer le neuf. La bascule d'un existant se fait par remplacement progressif, fonctionnalité par fonctionnalité, jamais d'un seul geste. Il faut le dire sans détour : le modèle ports et adaptateurs tient le mieux en terrain neuf ; sur le patrimoine il tient toujours, mais l'adaptateur paie une traduction explicite, et c'est là que se concentre le coût réel d'intégration. L'isolation est précisément ce qui rend cette intégration propre, à condition de ne jamais prétendre que la frontière est gratuite.

À RETENIR

- Deux frontières rendent le langage secondaire : les ports du domaine et un protocole d'outils inter-processus.
- Un adaptateur peut être écrit dans n'importe quel langage tant qu'il honore le contrat.
- Un service n'est qu'un adaptateur qui a migré dans son propre processus, le domaine ne change pas.
- Le contrat est un artefact gouverné : versionné, additif par défaut, lu par un consommateur tolérant, et fait évoluer en étendant puis en contractant, jamais en cassant en silence.
- Le contrat porte aussi du sens : un langage commun par contexte et un vocabulaire minimal aux jointures évitent les alignements hallucinés, là où un modèle de données central deviendrait ingérable.
- Face au patrimoine, l'adaptateur est une couche anticorruption : le modèle tient le mieux en neuf, et la frontière n'est jamais gratuite sur l'existant.

SECTION 03

Choix de langage, full TypeScript ou polyglotte

La question n'est pas un langage unique pour tout, mais le bon langage au bon endroit, derrière des contrats. Voici la position recommandée, fondée sur les systèmes déjà construits.

1 TypeScript pour l'orchestration et l'interface

Pour le coeur agentique (orchestrateur, registre d'outils, cas d'usage) et le cockpit, un langage unique côté serveur et interface évite de doubler la stack et de multiplier les surfaces de maintenance. Les types se partagent entre le back et le front. L'abstraction du modèle reste légère : un SDK direct, pas un gros framework de chaînes, pour garder la maîtrise des prompts, des événements et du coût en jetons.

2 Un langage spécialisé en sidecar quand la bibliothèque l'exige

Quand une capacité dépend d'un écosystème mûr dans un autre langage (automatisation de navigateur, calcul scientifique, modèles spécialisés), on l'isole dans un sidecar exposé via le protocole d'outils. C'est le cas, par exemple, d'un moteur d'exécution piloté depuis un sidecar dédié. Le coeur le consomme comme un fournisseur d'outils, sans absorber sa stack.

3 Rust ou Go pour les composants sensibles à la performance

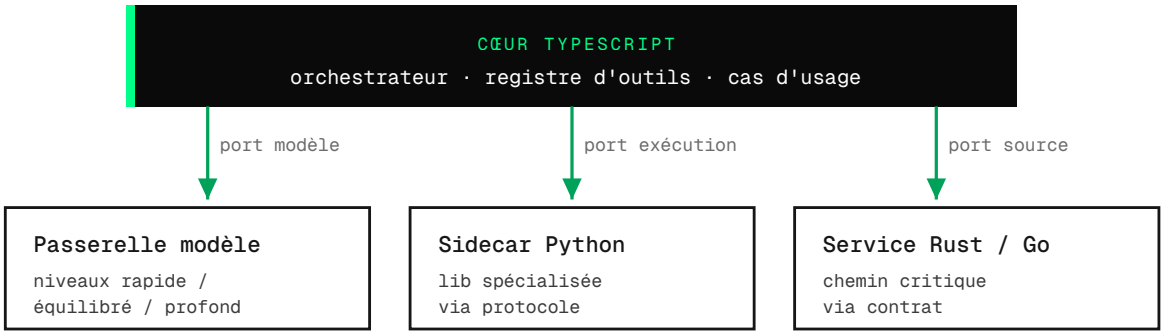
Pour un composant au chemin critique (passerelle à fort débit, traitement parallèle intensif, service à latence serrée), un langage compilé se justifie. Il est alors exposé comme un adaptateur ou un service derrière un contrat, sans contaminer le reste.

Matrice de décision du langage

COMPOSANT	RECOMMANDATION	POURQUOI
Orchestrateur, registre d'outils, cas d'usage	TypeScript	Stack unique avec l'interface, types partagés, abstraction légère du modèle.
Cockpit et interface	TypeScript	Continuité avec le back, streaming et rendu unifiés.
Capacité à écosystème spécialisé	Python en sidecar	Bibliothèques matures, isolées derrière le protocole d'outils.
Composant au chemin critique	Rust ou Go	Latence et débit, exposé derrière un contrat.

COMPOSANT	RECOMMANDATION	POURQUOI
Passerelle modèle	Indifférent	Gouvernance et routage centralisés, derrière le port modèle.

Schéma, un coeur TypeScript et des sidecars polyglottes



Le langage de chaque brique est un choix local, pas une décision d'architecture.

Anti-pattern à éviter : doubler la stack par défaut (un back dans un langage, une interface dans un autre) sans nécessité. Le polyglotte se justifie par une bibliothèque ou une contrainte de performance, pas par habitude. Quand il se justifie, il passe par un sidecar ou un service, jamais par un mélange dans le coeur.

À RETENIR

- TypeScript pour l'orchestration et l'interface, stack unique et abstraction légère du modèle.
- Python en sidecar quand une bibliothèque spécialisée l'exige, isolé via le protocole d'outils.
- Rust ou Go pour un composant au chemin critique, derrière un contrat.
- Le polyglotte se justifie par une raison technique, jamais par défaut, et toujours derrière une frontière.

SECTION 04

Monolithe modulaire ou microservices

La réponse n'est pas idéologique. Elle est séquentielle : commencer en monolithe modulaire bien découpé, puis externaliser un composant en service uniquement quand un critère objectif le déclenche.

1 Commencer en monolithe modulaire

Un monolithe modulaire hexagonal réunit dans un seul déployable un domaine pur et ses adaptateurs, plus d'éventuels sidecars pour les capacités spécialisées. Il offre la simplicité de déploiement, la cohérence transactionnelle et la rapidité d'itération dont une équipe a besoin au démarrage, sans renoncer à la modularité, puisque les frontières sont déjà posées par les ports.

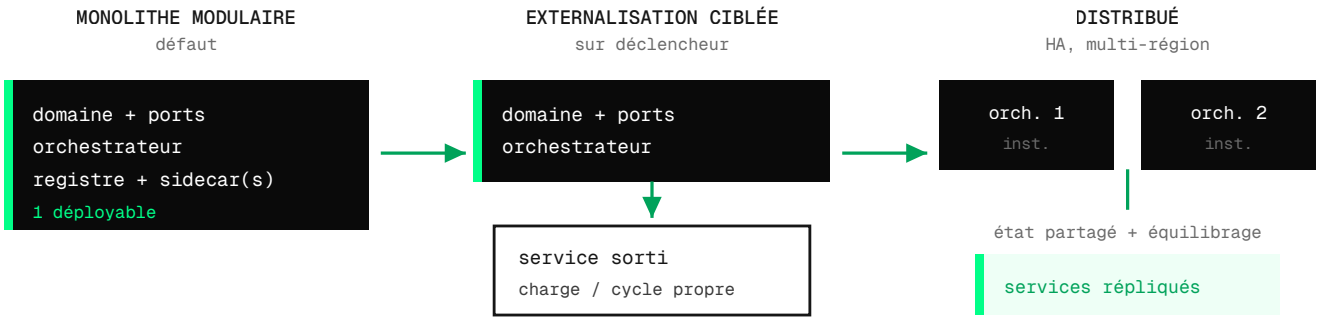
2 Externaliser un composant en service, sur déclencheur

Comme les frontières existent déjà, sortir un composant dans son propre processus est une opération locale, pas une refonte. On ne le fait que lorsqu'un critère le justifie. Tant qu'aucun déclencheur n'est atteint, le découpage en microservices ajoute du coût opérationnel sans bénéfice.

Matrice des déclencheurs d'externalisation

DÉCLENCHEUR OBSERVÉ	RÉPONSE
Une instance unique ne tient plus la charge	Externaliser l'état, puis répliquer le service concerné.
Un composant a un profil de charge ou de coût très différent	Le sortir en service pour le dimensionner indépendamment.
Un composant a un cycle de livraison propre	Le découpler pour le déployer séparément.
Un composant doit être isolé pour la résilience ou la sécurité	Le sortir derrière une frontière de processus dédiée.
Exigence de haute disponibilité ou multi-région	Topologie distribuée, état partagé, équilibrage de charge.
Aucun de ces critères	Rester en monolithe modulaire, c'est le bon défaut.

Schéma, trajectoire de topologie



Le cas particulier du multi-agent

Multiplier les agents indépendants est l'équivalent agentique des microservices, et appelle la même prudence. La forme qui domine en production reste l'orchestrateur unique qui dirige des spécialistes (moyeu et rayons). Le multi-agent réparti apporte des gains sur des tâches parallélisables, mais dégrade le raisonnement séquentiel et complique le débogage. Il se justifie pour de la haute complexité parallélisable, pas par principe.

Un mot sur l'émergence, car le terme prête à confusion et recouvre deux étages distincts. Au niveau de l'inférence, le comportement du modèle est déjà émergent : c'est le moteur, non déterministe, que toute cette architecture existe pour capter et rendre exploitable. On ne l'oppose à rien, on l'embrasse. Le compromis se situe ailleurs, au niveau de la topologie : faut-il un essaim d'agents qui s'auto-organisent, sans chef d'orchestre, ou un orchestrateur central qui dirige des spécialistes ? C'est là, et seulement là, qu'on choisit l'orchestration. L'auto-organisation promet plus d'autonomie et de résilience répartie, mais au prix de la prévisibilité, du débogage et de la traçabilité, trois exigences non négociables en exploitation régulée. On garde donc la main au niveau du système, sur un moteur qu'on laisse, lui, pleinement émergent, et l'on n'ouvre l'autonomie collective que par paliers gardés.

La règle qui décide est unique, et c'est la même pour toute cette section : on reste central tant que tout tient dans un seul processus orchestrateur ; on bascule un composant, qu'il s'agisse de délégation, de découverte ou de budget, sur son adaptateur distribué dès qu'il franchit une frontière de processus, d'hôte, de langage ou de confiance, et lui seul. Le store partagé qui débloque alors le multi-instance (section 07) héberge le registre de découverte et le registre d'allocations de budget : une seule frontière gouverne l'état, le budget et la découverte.

À RETENIR

- Défaut : monolithe modulaire hexagonal, frontières déjà posées par les ports.
- On externalise un composant en service sur déclencheur objectif, pas par principe.
- Le multi-agent réparti suit la même prudence que les microservices : utile pour la complexité parallélisable, coûteux pour le raisonnement séquentiel.
- L'orchestrateur unique dirigeant des spécialistes reste la forme dominante en production.
- Une seule règle de bascule : central tant qu'on tient dans un processus ; adaptateur distribué (délégation, découverte, budget) dès qu'on franchit une frontière de processus, d'hôte, de langage ou de confiance.

PARTIE II 4 SECTIONS

TOPOLOGIE · MÉMOIRE · EXÉCUTION

L'exécution d'un système agentique

Comment le système s'exécute : topologie runtime, ingénierie de la mémoire et du contexte, état, concurrence et travail asynchrone.

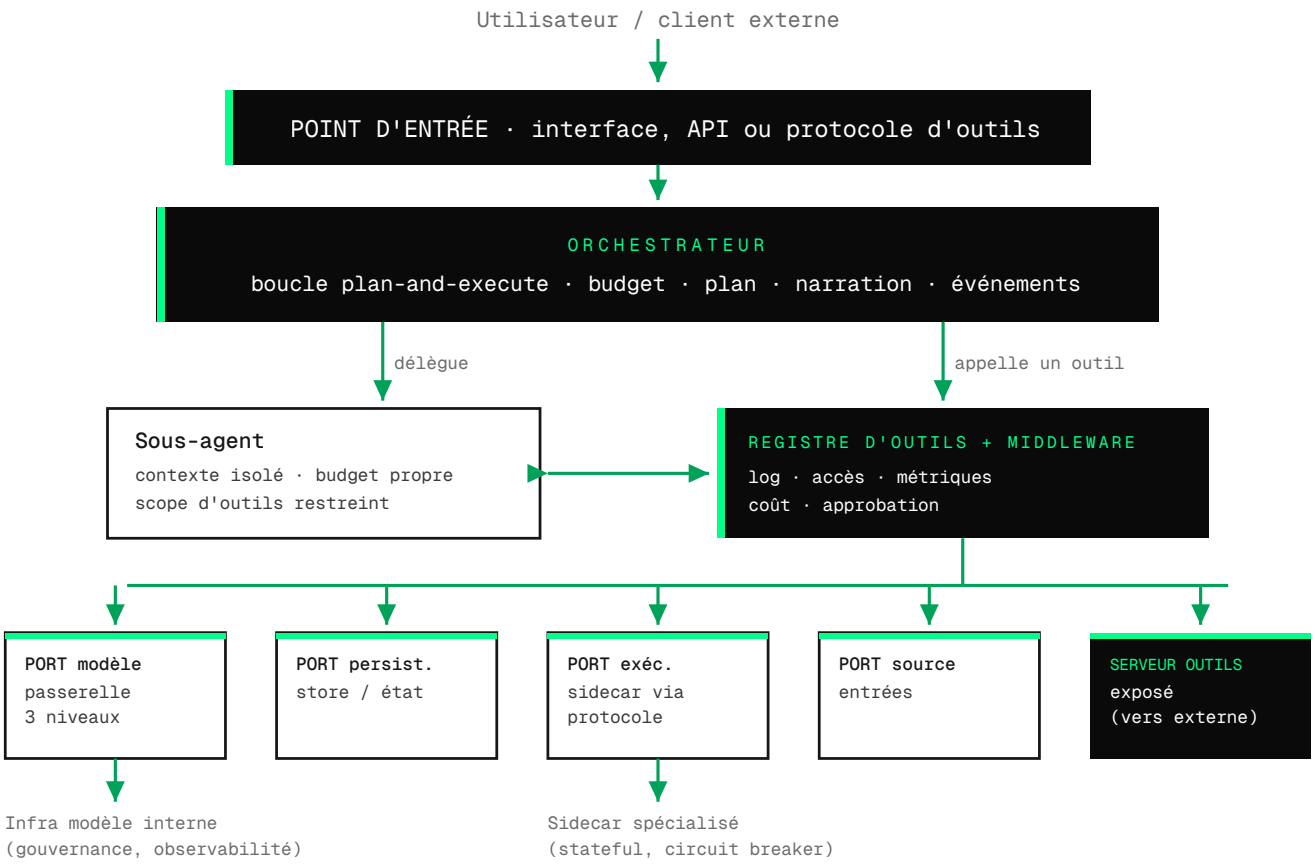
DANS CETTE PARTIE

SECTION 05

Topologie d'exécution d'un système agentique

Débarrassé de sa stack, un système agentique en marche garde toujours la même silhouette. Ses composants sont stables ; seuls leur langage et leur découpage varient, selon les sections précédentes.

Schéma de runtime



OBSERVABILITÉ TRANSVERSE · journaux · événements · lignée · métriques · coût

Lecture du schéma

- **Orchestrateur** : pilote la boucle, planifie, délègue, synthétise. Il ne contient pas de logique métier ; il compose.

- **Registre d'outils plus middleware** : surface unique où passent journalisation, contrôle d'accès, métriques, coût et approbations. Ajouter un garde-fou se fait ici, sans toucher aux outils. C'est aussi d'ici qu'est émis un **flux de traces unique**, consommé séparément par la provenance, l'évaluation et l'observabilité. Cette surface ne porte que du transversal, jamais d'orchestration ni de logique métier : c'est la garde contre l'écueil du bus d'intégration hypertrophié des architectures orientées services, où le canal finit par décider du métier. La logique reste dans les outils et l'orchestrateur ; le canal qui les relie demeure mince, et le registre reste un index, pas un cerveau.
- **Ports secondaires** : modèle (via une passerelle qui centralise gouvernance et routage par niveau), persistance, exécution (souvent un sidecar spécialisé), sources d'entrée.
- **Serveur d'outils exposé** : le système publie ses cas d'usage comme outils standardisés, pour être piloté par un orchestrateur externe sans copie de logique.
- **Observabilité transverse** : chaque étape laisse une trace exploitable a posteriori, et cette trace nourrit l'évaluation continue, pas seulement un tableau de bord.

Le sidecar comme motif récurrent

Une capacité spécialisée (par exemple un moteur d'exécution piloté par un autre langage) tourne dans un sidecar à session, consommé via le protocole d'outils. Le sidecar est traité comme une dépendance externe : santé surveillée, disjoncteur (circuit breaker) en cas d'indisponibilité, procédure de relance documentée. Le reste du système continue de fonctionner en mode dégradé si le sidecar tombe.

Le registre, un port à deux adaptateurs

Découvrir quels outils et quels sous-agents sont disponibles est, comme le reste, une décision d'adaptateur. Tant que tout vit dans un seul processus, le registre est une simple table en mémoire, atomique et gratuite. Dès qu'une capacité sort du processus, d'un autre langage ou d'une autre équipe, le même port bascule sur un adaptateur de **découverte de service** : chaque agent publie une fiche de capacités auto-descriptive, découverte via un registre partagé, par annonce et abonnement avec cache local, jamais par un appel central à chaque itération. Le contrat de découverte ne change pas ; seul l'adaptateur change avec la topologie.

À RETENIR

- Composants stables : orchestrateur, registre d'outils gouverné, ports secondaires, serveur d'outils exposé, observabilité transverse.
- La passerelle modèle centralise gouvernance et routage par niveau, derrière le port.
- Le sidecar est le motif récurrent pour intégrer une capacité spécialisée, traité comme une dépendance externe surveillée.
- Le middleware émet un flux de traces unique : provenance, évaluation et observabilité le consomment séparément.
- Le registre est un port : table en mémoire en mono-processus, fiches de capacités découvertes via un registre partagé en distribué.
- Middleware et registre ne portent que du transversal, jamais d'orchestration : la logique vit dans les outils et l'orchestrateur, le canal reste mince, pour ne pas refaire le bus d'intégration hypertrophié.

SECTION 06

Ingénierie de la mémoire et du contexte

La mémoire est la forme première de l'état d'un agent, et le déterminant principal de la qualité comme du coût. Un système agentique n'est pas seulement une boucle qui appelle des outils : c'est une boucle qui se souvient, oublie et se contextualise. Cette section érige l'ingénierie de la mémoire et du contexte au rang de pilier, au même titre que l'orchestration et la frontière du modèle.

> Principe directeur de la section : la frontière du modèle ne s'applique pas qu'à la génération, elle s'applique aussi à la récupération. On ne récupère pas tout, tout le temps. La décision de quoi charger est déterministe, pas confiée au modèle.

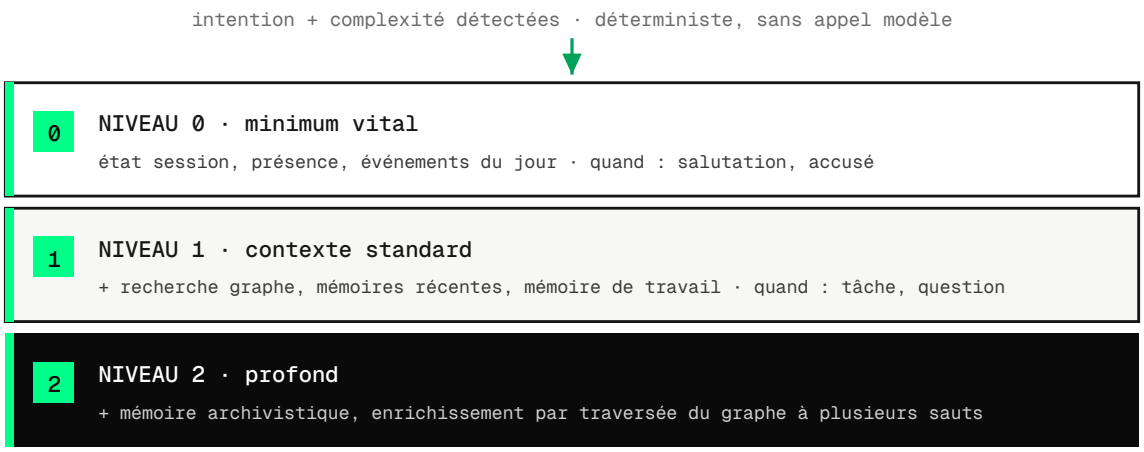
Un magasin unique ne suffit pas

Aucune technologie de mémoire ne couvre seule tous les besoins. On compose donc plusieurs magasins, chacun à son rôle, derrière des ports, avec dégradation gracieuse si l'un est absent.

RÔLE DE MÉMOIRE	CE QU'IL PORTE	CARACTÉRISTIQUE
Source de mémoires	Faits et souvenirs avec métadonnées enrichies	Écriture et lecture simples, métadonnées riches
Graphe de connaissances	Relations entre souvenirs, entités, événements	Liaison automatique, recherche vectorielle, traversée
Mémoire de travail	État de la session courante, présence	Temps réel, faible latence
Mémoire archivistique	Historique profond, peu sollicité	Volumineux, interrogé seulement au besoin

La frugalité de récupération étagée

Le cœur de l'ingénierie de la mémoire est de ne pas tout interroger à chaque tour. Une décision déterministe, fondée sur la complexité et l'intention détectées, choisit la profondeur de récupération. C'est la frontière du modèle appliquée à la récupération de données.



Raffinements : un intent de rappel force le niveau 2 ; une salutation force le niveau 0 ;
le routeur cible les sources utiles (par exemple, calendrier vers le graphe seul).

L'effet est double : la latence d'un tour trivial s'effondre, et le coût de contexte se concentre là où il crée de la valeur. La majorité des tours ne méritent pas une récupération profonde, et le système le décide sans brûler un appel de modèle pour s'en rendre compte. À chaque niveau, c'est le même score de valeur (récence, fréquence, pertinence, importance, centralité) qui ordonne ce qui remonte. Sa pondération n'est pas universelle : elle se calibre par domaine et se vérifie comme tout gate déterministe, plutôt que d'être posée par décret.

L'oubli est une fonction de premier ordre

Une mémoire qui n'oublie jamais devient du bruit. Mais oublier n'est pas effacer : par défaut, c'est cesser de faire remonter un souvenir, pas le détruire. L'oubli se joue à la récupération, gouverné par un score de valeur, et la suppression physique reste l'exception. La décroissance avec renforcement à chaque accès ne fait que reprendre la courbe de l'oubli décrite par Ebbinghaus en 1885 : ce qu'on ne sollicite pas s'estompe, ce qu'on rappelle se renforce.

MÉCANISME	PRINCIPE	EFFET
Score de valeur	Un score combine récence, fréquence d'accès, pertinence, importance et centralité dans le graphe. L'âge n'est qu'un signal, modulé par le renforcement et la saillance.	Le pertinent remonte ; le reste se retire de la vue, sans être détruit.
Invalidation, pas suppression	Un fait contredit est marqué invalide avec ses deux temps (appris, valide), au lieu d'être supprimé. C'est le modèle bi-temporel.	On garde l'histoire : ce qui est vrai maintenant, et ce qui l'était avant.
Consolidation	Les quasi-doublons fusionnent, les épisodes proches se résument en faits de plus haut niveau.	Oublier, c'est souvent compresser : moins de volume, autant de sens.

MÉCANISME	PRINCIPE	EFFET
Immunité des faits	Les faits stables et les invariants d'identité sont marqués et exemptés de décroissance.	Ce qui fait l'identité ne s'efface jamais.

La suppression physique, elle, est réservée aux données dérivées de faible valeur, par exemple des liens associatifs invalidés et anciens, sous une règle de rétention explicite, et au seul effacement imposé. Démoter vers une archive froide est toujours préféré à détruire. L'importance, signal central de ce score, ne coûte pas un appel de modèle : elle se déduit de signaux déterministes, fréquence d'accès, centralité, catégorie, nouveauté, et d'un label émis comme sous-produit de l'extraction asynchrone déjà effectuée hors du tour. La frontière du modèle, appliquée jusqu'au calcul de l'importance. La consolidation, elle, reste dérivée et réversible : l'original survit dans le journal immuable (§07), on ne consolide jamais la source de vérité, et chaque résumé garde un lien explicite vers les faits bruts dont il provient, pour qu'un audit puisse toujours remonter à l'origine, même après que la mémoire de travail a oublié le détail. Enfin, tout contenu récupéré entre dans le contexte comme une donnée non fiable, jamais comme une instruction : la mémoire est aussi une surface d'injection indirecte (§11).

La liaison se fait toute seule

Plutôt que de demander à l'utilisateur de taguer et d'organiser, le système lie automatiquement les souvenirs entre eux et aux entités, par similarité au-dessus d'un seuil. Les connexions émergent dans le graphe. Le seuil seul ne suffit pas : il produit des arêtes faibles et des nœuds-hubs ; il faut élaguer les liens peu portants, pas seulement les liens invalidés, pour garder la précision du graphe. La récupération profonde exploite ces liens par une traversée à quelques sauts autour des résultats directs.

Le travail lourd se fait hors du tour interactif

Extraction de faits, consolidation de souvenirs proches, mise à jour du graphe, résumés de session : ces opérations coûteuses ne se font pas pendant que l'utilisateur attend. Elles sont déléguées à un traitement de fond, asynchrone, déclenché après le tour. La mémoire se range pendant que le système est au repos, pas pendant qu'il répond.

La validité dans le temps, le cœur du modèle bi-temporel

La décroissance gère l'ancienneté, mais pas la validité. Un fait vrai hier peut être faux aujourd'hui. D'où les deux temps portés par chaque fait : celui où il a été appris et celui où il est valide. C'est ce qui permet d'invalider sans détruire, et de raisonner sur l'évolution des informations, interroger ce qui est vrai maintenant ou ce qui l'était à une date donnée. Couplé au journal immuable de la section 07, le modèle bi-temporel réconcilie l'oubli du raisonnement et la traçabilité de l'audit. Quand deux faits se contredisent, le plus récemment appris invalide l'ancien, sauf source plus autoritaire ; les deux subsistent dans l'histoire.

À RETENIR

- La mémoire est de l'état, et un pilier d'architecture à part entière, pas une fonctionnalité du modèle.
- Frugalité de récupération étagée : une décision déterministe choisit la profondeur, la frontière du modèle appliquée à la récupération.
- Plusieurs magasins par rôle, derrière des ports, avec dégradation gracieuse.
- Oublier n'est pas effacer : un score de valeur (récence, fréquence, pertinence, importance, centralité) retire de la vue ; invalidation et consolidation préservent l'histoire ; le travail lourd se fait en arrière-plan.
- L'importance se déduit de signaux déterministes et d'un label amorti sur l'extraction asynchrone, sans appel de modèle sur le tour.
- Le modèle bi-temporel (deux temps par fait) réconcilie l'oubli du raisonnement et la traçabilité de l'audit.

SECTION 07

État, concurrence et passage à l'échelle

Le passage à l'échelle d'un système agentique se joue moins sur le modèle que sur la gestion de l'état. La règle est simple : l'agent est un réducteur sans état caché, et l'état vit à l'extérieur. Mais « l'état » n'est pas monolithique : il se lit en trois couches qu'il faut distinguer pour ne pas confondre ce qui doit durer et ce qui doit pouvoir s'oublier.

1 L'agent comme réducteur sans état

Un tour d'agent se traite comme une fonction qui prend un état d'entrée et un événement, et produit un état de sortie, sans mémoire interne cachée entre les appels. Chaque tour n'agit que sur le contexte explicite qu'on lui donne. Cela rend les exécutions reproductibles, débogables, et permet de répliquer le service sans surprise.

2 L'état explicite, en trois couches

Toutes les interactions sont enregistrées comme un journal d'événements en ajout seul (event sourcing), rattaché à une entité métier. Le procédé est ancien : c'est l'héritier de la comptabilité en partie double, codifiée par Luca Pacioli en 1494, le premier registre où l'on n'efface jamais : on ajoute une écriture qui corrige. Mais ce journal n'est pas la mémoire de travail de l'agent : il en est la source froide. Trois couches se distinguent.

- **Le journal d'interaction** est immuable : source de vérité, moyen d'auditer et de rejouer une trajectoire à partir des sorties enregistrées. On n'y efface rien.
- **La mémoire de travail** (section 06) est une vue dérivée, une projection (CQRS) de ce journal et des extractions qui en découlent. C'est elle qui oublie, se densifie et se contextualise, librement, car le journal la réconcilie toujours.
- **La provenance de prompt** (section 08) relie les deux : à chaque inférence, on garde l'empreinte de ce qui a été réellement injecté, pour rejouer exactement ce que le modèle a vu, même après oubli.

Confondre ces couches, faire du journal « à la fois la mémoire et l'audit », force des compromis impossibles : soit on alourdit l'agent de plusieurs contextes, soit on détruit de la donnée d'audit en voulant alléger la mémoire. Les séparer lève le dilemme.

Rejouer ne veut pas dire ré-appeler le modèle. Un appel de modèle n'est pas déterministe : on ne le rejoue pas, on relit la sortie qu'il a produite, enregistrée par la provenance. Le flot de la trajectoire, lui, est déterministe et se rejoue fidèlement. Trois usages se distinguent : le rejeu d'audit reconstruit une trajectoire à partir des sorties déjà enregistrées ; la reprise repart d'un point de reprise sans tout rejouer ; la réplication, qu'autorise précisément l'absence d'état caché, fait porter le même état explicite par une autre instance. La

cohérence temporelle, savoir quelle version de prompt et de modèle a produit quelle sortie, est portée par la provenance.

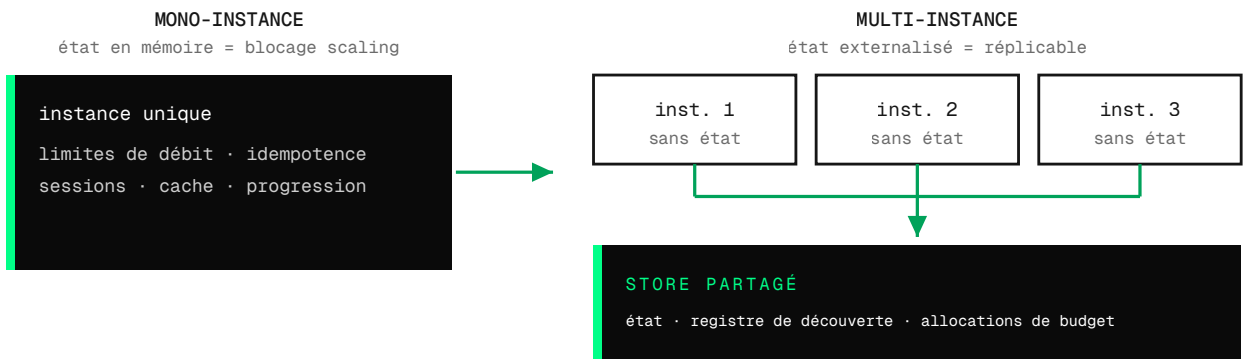
On n'efface pas la vérité, on la refroidit. Le journal immuable croît, mais sa partie chaude reste bornée : segments anciens compactés et archivés à froid, conservés pour l'audit. La suppression physique est l'exception, réservée aux données dérivées de faible valeur sous règle de rétention, et au seul effacement imposé, rendre une donnée illisible sans rompre la chaîne d'audit.

3 Du mono-instance au multi-instance

En mono-instance, des structures en mémoire suffisent pour les limites de débit, l'idempotence, les sessions, le cache et la progression. Ce sont précisément ces structures qui bloquent la réplication. Les passer dans un store partagé est le geste qui ouvre le multi-instance, sans réécrire le domaine. Ce même store partagé, dès que l'on franchit la frontière du processus unique, héberge trois choses d'un coup : l'état externalisé, le registre de découverte des capacités, et le registre d'allocations de budget.

Le domaine ne change pas, mais la sémantique, si, et ce geste n'est pas gratuit. Un accès local et atomique devient un appel réseau, avec sa latence et ses pannes partielles. La cohérence forte cède la place à une cohérence éventuelle au niveau global : l'état converge sans être instantanément partagé. La cohérence de session, elle, reste souvent exigée, relire ses propres écritures, suivre l'ordre de causalité, ce qui n'est pas la même chose qu'une cohérence éventuelle pure. Une opération qui s'étend sur plusieurs processus n'est plus une transaction unique ; elle devient une suite d'étapes locales, chacune dotée d'une action de compensation qui annule son effet si une étape ultérieure échoue. L'ordre causal se préserve explicitement, un effet ne précède jamais sa cause, par des numéros de séquence par entité ou des horloges logiques ; et l'idempotence doit tenir sous concurrence, pas seulement au nouvel essai, par des clés d'idempotence et un store de déduplication. C'est le prix d'entrée du distribué : on ne le paie que lorsqu'un déclencheur objectif l'exige, et en connaissance de cause.

Schéma, externalisation de l'état pour répliquer



Concurrence et budgets

Plusieurs sous-agents peuvent travailler en parallèle ; chacun a son budget d'itérations propre, plafonné, et un plafond global par tâche racine borne l'ensemble. Tant que tout tient dans un processus, ce plafond se tient par un compteur agrégé en mémoire, atomique et gratuit parce que la boucle d'événements sérialise les accès, une garantie qui tombe dès qu'on se distribue. Dès que les sous-agents se répartissent entre processus, le compteur central au jeton devient un goulot : on passe à une allocation par blocs (lease). L'orchestrateur attribue des lots ; chaque sous-agent tient un compteur local, redemande à l'approche de l'épuisement, et restitue l'inutilisé, contrôle strict sans verrouiller le store en permanence. Le lot est daté : un sous-agent qui tombe le voit expirer et libérer, plutôt que d'immobiliser le budget. La concurrence est utile pour les sous-tâches indépendantes ; elle ne doit pas masquer un raisonnement séquentiel qui, lui, gagne à rester dans un seul fil.

À RETENIR

- L'agent est un réducteur sans état caché : chaque tour agit sur un contexte explicite et produit un état explicite.
- L'état se lit en trois couches : journal d'interaction immuable (vérité, audit, rejeu), mémoire de travail dérivée (qui oublie), provenance de prompt (qui réconcilie).
- Rejouer relit les sorties enregistrées, sans ré-appeler le modèle non déterministe ; on distingue rejeu d'audit, reprise et réplication.
- Franchir la frontière du processus change la sémantique : cohérence éventuelle, compensation au lieu de transaction unique, ordre causal explicite, idempotence sous concurrence. Un coût qu'on assume sur déclencheur, pas par défaut.
- On n'efface pas la vérité, on la refroidit : journal compacté et archivé à froid, suppression physique réservée aux dérivés et à l'effacement imposé.
- Externaliser l'état débloque le multi-instance ; le store partagé héberge alors état, registre de découverte et registre d'allocations de budget.
- Budgets bornés par sous-agent et plafond global : compteur central en mono-processus, allocation par blocs (lease) dès qu'on se distribue.

SECTION 08

Travail asynchrone et ordonnancement

Un système agentique de qualité fait peu de choses pendant que l'utilisateur attend, et beaucoup de choses quand il a le dos tourné. Le tour interactif reste mince ; le travail lourd, coûteux ou différé est repoussé en arrière-plan. Cette section est le runbook de cette séparation : ce qui se fait en synchrone, ce qui se fait en asynchrone, par quels déclencheurs, et sous quels garde-fous.

> Règle directrice : tout ce qui n'est pas nécessaire à la réponse du tour courant sort du tour. Extraction, consolidation, synchronisation, résumés, rappels : en arrière-plan. Le tour interactif ne paie que la latence indispensable.

Deux régimes : événementiel et ordonnancé

RÉGIME	DÉCLENCHEUR	USAGE
Événementiel	Un événement métier est émis (fin de tour, fin de session, rappel créé)	Réaction immédiate mais hors tour : on traite dès que possible, sans bloquer l'utilisateur.
Ordonnancé	Une horloge (expression cron) déclenche à heure fixe	Entretien périodique : décroissance, élagage, nettoyage, détection de tendances.

Schéma, le pipeline post-tour et l'ordonnanceur

1 Le pipeline post-tour

Après chaque tour, un événement est émis et un pipeline asynchrone se déclenche. Il enchaîne des étapes indépendantes : détecter les engagements pris, détecter les suivis à programmer, extraire les faits à mémoriser, décider si un résumé de conversation est nécessaire, et préchauffer le contexte profond pour le prochain tour. Chaque étape est isolée : elle a son propre nouvel essai, et l'échec d'une étape ne fait pas échouer les autres. C'est aussi ici que se calcule l'importance des souvenirs, comme sous-produit de l'extraction, sans appel de modèle dédié.

Le tour dépose enfin, à l'émission, son **empreinte de provenance** : un identifiant de requête et la signature de ce qui a été réellement injecté dans le contexte. C'est elle qui permet de rejouer exactement ce que le modèle a vu, même après que la mémoire de travail a oublié (cf. section 07). Et cette même trace alimente l'évaluation comportementale : un seul flux émis par le middleware, plusieurs consommateurs, provenance, évaluation (section 12) et observabilité.

2 Le pipeline de fin de session

À la clôture d'une session, un second pipeline archive la session, extrait et structure son contenu vers le graphe de connaissances, en dérive les faits durables, et journalise l'activité. Les opérations indépendantes y sont lancées en parallèle, puis recombinées.

3 L'ordonnanceur périodique

L'entretien du système est confié à des tâches planifiées par expression cron. Exemples de cadence, lisibles directement dans la configuration :

TÂCHE	CADENCE (CRON)	RÔLE
Décroissance mémoire	`0 4 * * *` (quotidien, 04:00)	Met à jour le score de valeur des souvenirs ; gate la récupération, ne supprime pas.
Élagage des liens	`0 5 * * 0` (hebdo, dimanche 05:00)	Élague les seuls liens dérivés invalidés et anciens, sous rétention.
Archivage à froid	`0 2 * * 0` (hebdo, dimanche 02:00)	Compacte et refroidit les segments anciens du journal, conservés pour l'audit.
Détection de tendances	`0 3 * * 1` (hebdo, lundi 03:00)	Repère des patterns récurrents à capitaliser.
Rappels d'expiration	`0 9 * * *` (quotidien, 09:00)	Émet les rappels arrivés à échéance.
Effacement imposé	Sur demande	Rend illisible la donnée d'un sujet (effacement légal) sans rompre la chaîne d'audit.

4 Garde-fous de l'asynchrone

Un travail de fond mal gardé est pire que pas de travail de fond. Quatre garde-fous sont systématiques :

- **Nouvel essai par étape** : chaque étape se rejoue indépendamment, avec un nombre d'essais borné, sans rejouer ce qui a réussi.
- **Idempotence** : une tâche rejouée ne produit pas de doublon ; l'effet est le même qu'elle tourne une ou trois fois. C'est elle qui protège la cohérence quand une tâche est livrée ou rejouée plusieurs fois, y compris sous concurrence. La question de l'ordre, distincte, relève de la causalité (section 07), pas de l'idempotence.
- **Concurrence bornée** : le nombre de tâches simultanées est plafonné, et souvent cloisonné par utilisateur, pour éviter qu'un acteur sature la file.
- **Observabilité des jobs** : chaque exécution est tracée, le retard de la file et le taux d'échec sont des métriques de premier plan.

Runbook, synchrone ou asynchrone

OPÉRATION	OÙ
Produire la réponse du tour	Synchrone
Récupération mémoire du niveau requis pour répondre	Synchrone, étagée
Extraction de faits, consolidation, mise au graphe	Asynchrone, post-tour
Archivage et résumé de session	Asynchrone, fin de session
Décroissance, élagage, nettoyage, tendances	Asynchrone, ordonnancé
Préchauffage du contexte du prochain tour	Asynchrone, post-tour

À RETENIR

- Le tour interactif reste mince ; le travail lourd part en arrière-plan, par événement ou par cron.
- Le pipeline post-tour enchaîne des étapes à nouvel essai indépendant ; la fin de session archive et consolide.
- Le tour dépose une empreinte de provenance ; cette même trace, émise une fois par le middleware, sert la provenance, l'évaluation et l'observabilité.
- L'entretien périodique met à jour les scores, invalide, refroidit à froid et élague les seuls dérivés ; il ne détruit pas la vérité.
- Quatre garde-fous non négociables : essais par étape, idempotence, concurrence bornée, observabilité des jobs.

PARTIE III

4 SECTIONS

RÉSILIENCE · OBSERVABILITÉ · SÉCURITÉ · ÉVALUATION

Robustesse, sécurité et évaluation

Tenir dans la durée : résilience d'exécution, observabilité, maîtrise du coût, sécurité et évaluation comportementale des agents.

DANS CETTE PARTIE

RÉSILIENCE · OBSERVABILITÉ · SÉCURITÉ

SECTION 09

Résilience d'exécution

Un système agentique compose des dépendances non déterministes et faillibles : modèle, sidecars, services externes. La résilience se conçoit par défaut, pas après l'incident.

Qualifier l'erreur avant de réagir

Chaque échec est d'abord qualifié, car la bonne réponse dépend du type d'erreur. On distingue les erreurs transitoires, les erreurs de validation, et les erreurs métier.

TYPE D'ERREUR	STRATÉGIE
Transitoire (réseau, délai, surcharge)	Réessai avec recul exponentiel borné.
Validation (sortie non conforme au contrat)	Réessai unique avec le diagnostic réinjecté dans le contexte.
Métier (ressource manquante, action impossible)	Délégation à une étape de diagnostic, ou remontée pour revue humaine.
Fournisseur de modèle indisponible	Bascule automatique vers un fournisseur de repli, derrière le même port.
Service non critique indisponible	Mode dégradé, l'application continue sans la fonction concernée.

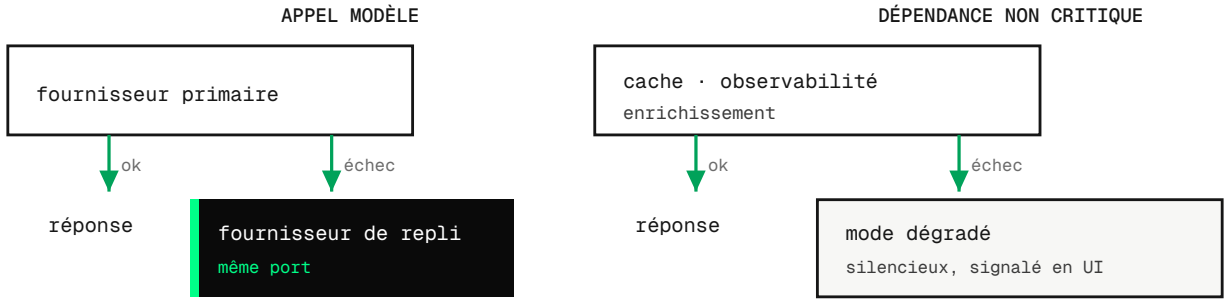
1 Fail-open pour le non critique, fail-closed pour le sensible

Une dépendance non critique qui tombe ne doit pas faire tomber le système : cache, observabilité, enrichissement passent en mode dégradé silencieux. À l'inverse, une action sensible (mutation, écriture, push externe) échoue de manière fermée et explicite plutôt que de s'exécuter dans le doute. La règle : dégrader la lecture, jamais l'action.

2 Détection de fonctionnalités et déploiement bimodal

Au démarrage, le système détecte les services réellement disponibles selon la configuration. Une dépendance absente désactive proprement la fonction associée, sans planter. Il en résulte un déploiement bimodal : un mode minimal qui tourne avec le strict nécessaire, et un mode complet quand toutes les dépendances sont présentes. L'interface signale ce qui est actif plutôt que d'échouer silencieusement.

Schéma, bascule et dégradation



ACTION SENSIBLE : pas de repli silencieux. Échec fermé, explicite, tracé.

À RETENIR

- Qualifier l'erreur d'abord : transitoire, validation, métier, fournisseur, non critique.
- Fail-open pour la lecture et le non critique, fail-closed et explicite pour les actions sensibles.
- Bascule multi-fournisseur derrière le même port, sans réécriture. À distinguer de la promotion d'un modèle nouveau, qui se valide d'abord en déploiement fantôme (section 12) : le repli de disponibilité est immédiat, la promotion se mérite.
- Détection de fonctionnalités au démarrage et déploiement bimodal minimal ou complet.

SECTION 10

Observabilité et maîtrise du coût

Un système agentique non observé est ingouvernable et imprévisible en coût. L'observabilité n'est pas une option de fin de projet, c'est une propriété de conception, portée par la chaîne de middleware. Et la trace qu'elle produit ne sert pas qu'au tableau de bord : c'est le même flux qui nourrit l'évaluation comportementale continue (section 12) et la provenance (section 08).

Trois niveaux d'observabilité indépendants

NIVEAU	QUOI	USAGE
Journaux structurés	Une ligne par événement, contexte typé (module, identifiant de requête, session).	Agrégation standard, recherche, alerte.
Événements de cycle	Chaque étape de l'orchestrateur et chaque appel d'outil persistés.	Rejeu d'une trajectoire, audit comportemental.
Métriques par appel de modèle	Jetons entrée et sortie, cache, niveau utilisé, durée, statut, tentative.	Suivi du coût et de la performance par agent.

La propagation d'un identifiant de requête

Un identifiant unique est propagé de l'entrée jusqu'à chaque appel d'outil et de modèle, et jusqu'aux sous-agents via la lignée parent et enfant. C'est lui qui permet de reconstituer une trajectoire complète a posteriori, condition de l'auditabilité exigée en environnement régulé.

Reconstruire le contexte d'une décision passée

Auditer une décision, ce n'est pas seulement retrouver qu'un appel a eu lieu, c'est reconstruire le contexte tel qu'il était au moment où l'agent a décidé. Or la mémoire de travail consolide et oublie (section 06). L'observabilité doit donc pouvoir déplier un souvenir compressé vers les faits bruts dont il provient, à l'horodatage exact de la décision, en suivant les pointeurs déposés à la consolidation. L'exécution courante reste légère ; l'explication d'une décision passée reste possible, même après que la mémoire de travail l'a résumée. C'est la condition d'un audit qui tient en environnement régulé.

Une conséquence souvent oubliée : ces traces contiennent les prompts et les sorties, donc de la donnée. Les exporter vers un service d'observabilité tiers est une exfiltration comme une autre ; elles suivent le même régime de souveraineté que la donnée qu'elles transportent (section 15).

Le coût se pilote par l'architecture, pas par la chance

Trois leviers, tous structurels, gardent le coût sous contrôle :

- **Frontière du modèle** : tout ce qui est déterministe ne paie pas le prix d'un appel de modèle.
- **Routage par niveau** : la complexité estimée d'une tâche choisit le niveau rapide, équilibré ou profond. En cas de doute, on monte d'un niveau plutôt que de risquer une mauvaise réponse.
- **Cache et stabilité des prompts** : les segments stables du prompt sont positionnés pour être réutilisés. La stabilité du contenu prime sur la seule réduction de jetons.

! Plafonds explicites en exploitation continue

Pour un système qui tourne en fond, un compteur de coût agrégé par tâche racine et des plafonds par fenêtre de temps sont indispensables. Au dépassement, l'orchestrateur s'arrête et rend une synthèse, plutôt que de poursuivre sans contrôle. Le coût récurrent devient ainsi une grandeur pilotée, pas subie.

> À RETENIR
>
> - Trois niveaux indépendants : journaux structurés, événements de cycle, métriques par appel de modèle.
> - Un identifiant de requête propagé partout rend chaque trajectoire rejouable et auditable.
> - Le coût se pilote par la frontière du modèle, le routage par niveau et le cache, plus des plafonds explicites.
> - La trace n'est pas qu'un tableau de bord : c'est le flux unique qui alimente aussi l'évaluation continue et la provenance.
> - Les traces contiennent prompts et sorties, donc de la donnée : même régime de souveraineté, l'export tiers est une exfiltration.
> - Auditer, c'est reconstruire le contexte d'une décision : on déplie un souvenir consolidé vers ses faits bruts à l'horodatage, via les pointeurs de consolidation.

SECTION 11

Sécurité d'exécution

Principe directeur : la sécurité se joue sur les actions, pas sur l'affichage. La lecture peut rester ouverte ; ce sont les mutations et les effets de bord qui sont gardés.

Contrôle d'accès à trois niveaux

NIVEAU	MÉCANISME	EFFET
Outil	Filtrage du registre selon le rôle, avant l'envoi au modèle.	Le modèle ne connaît pas un outil qu'il n'a pas le droit d'appeler.
Donnée	Garde qui vérifie l'appartenance ou le périmètre avant une mutation.	Un acteur ne modifie que ce qui le concerne.
Requête	Lecture libre, écriture bloquée pour les accès génériques (par exemple requêtes en lecture seule).	Analyse large possible, mutation impossible par ce canal.

L'identité de l'agent

Avant de filtrer ce qu'un agent peut faire, il faut savoir qui agit. Chaque agent est un principal identifié, distinct de l'utilisateur pour le compte duquel il opère. Il porte sa propre identité, à moindre privilège ; quand il agit pour un utilisateur, la délégation est explicite, bornée dans le temps et dans le périmètre, jamais l'emprunt des pleins droits de la personne. L'audit distingue ainsi toujours deux choses : qui a agi, l'agent, et pour qui, l'utilisateur. Corollaire de registre : un agent qui agit sans être connu du registre est une anomalie de sécurité, à traiter comme telle. L'emplacement de cette identité, gestion d'accès existante ou brique dédiée, est une décision d'infrastructure qui se prend par-dessus ; le principe, lui, ne varie pas.

L'approbation humaine ancrée dans le contrat de l'outil

Les outils à impact déclarent dans leur contrat qu'ils requièrent une approbation. Avant l'exécution, l'infrastructure déclenche une demande d'approbation : accepter, refuser, ou éditer les arguments. Si refusé, l'exécution n'a jamais lieu. La garde ne dépend pas de la discipline du modèle ; elle est portée par l'infrastructure, donc garantie.

Mais une approbation ne doit pas geler un processus vivant. Si l'humain n'est pas là, l'agent ne reste pas suspendu en mémoire à monopoliser des ressources jusqu'au délai d'expiration. Il passe dans un état explicite « suspendu » : sa trajectoire est sérialisée dans l'état durable, une projection de la mémoire de travail ; le processus se libère ; et le système réhydrate l'agent à l'arrivée de la décision, des minutes ou des heures plus tard, exactement là où il s'était arrêté. C'est la reprise sur point de reprise (section 07) appliquée

à l'attente humaine : une suspension, pas un blocage. Les demandes à faible urgence sont regroupées et priorisées dans une file, présentées en résumé, pour ne pas noyer l'humain ni l'inciter à valider en aveugle. Ce résumé est lui-même une surface d'attaque : il doit être déterministe et fidèle aux arguments réels de l'outil, jamais une reformulation par le modèle, sinon une action injectée se glisse dans le lot et l'injection se réintroduit au point de contrôle. Le fail-closed reste entier, sans payer le prix d'un agent gelé.

Schéma, le cycle de suspension

Suspendre, pas bloquer : le fail-closed sans jamais geler l'agent.

Isolation des dépendances et surface exposée

- **Sidecars isolés** : une capacité spécialisée tourne dans son propre processus, avec accès restreint et disjoncteur en cas de défaillance.
- **Exécution éphémère** : une exécution à risque, code généré ou traitement de contenu non fiable, tourne dans un environnement jetable, provisionné par tâche et détruit après usage. Ce qui est détruit ne se compromet pas durablement : l'éphémère borne dans le temps ce qu'une compromission peut atteindre.
- **Surface d'outils exposée gardée** : le serveur d'outils que le système publie est protégé par clé et limite de débit, et n'est ouvert au-delà du local que de manière contrôlée.
- **Secrets et données sensibles** : isolés derrière la configuration typée, jamais dans le domaine. Et un cran plus loin pour les secrets d'exécution : le secret ne traverse jamais la frontière du modèle ni celle de l'environnement d'exécution. L'agent ne manipule qu'un substitut ; la clé réelle est attachée par l'infrastructure à la frontière réseau, vers les seules destinations explicitement autorisées, et se remplace sans redéploiement. Un secret que le modèle n'a jamais vu ne peut pas être divulgué : cette garde est structurelle, pas comportementale, et elle retire à l'injection son butin le plus convoité.

Entrées non fiables et injection de prompt

La menace de premier rang d'un système agentique est l'injection de prompt : des instructions hostiles glissées dans l'entrée. Sa forme directe vient de l'utilisateur ; sa forme indirecte, plus pernicieuse, se cache dans le contenu récupéré (page, document, souvenir) et se déclenche quand le modèle l'ingère. Elle est donc **intrinsèque à toute mémoire ou récupération** (§06). Le pire cas naît d'une triade : accès à des données privées, ingestion de contenu non fiable, et moyen de communication vers l'extérieur, réunis sur un même chemin non gardé ; retirer un seul des trois le désamorce. D'où une règle opérationnelle, dont la fenêtre est celle du contexte : tant qu'un contenu non fiable est présent dans le contexte, n'autoriser que deux de ces trois propriétés à la fois ; si les trois sont réellement nécessaires, l'action ne s'exécute pas en

autonomie, elle passe sous supervision humaine. La fenêtre n'est ni le tour, trop étroit puisque le contenu ingéré persiste au-delà, ni la session entière, trop large : elle s'ouvre à l'ingestion et ne se referme qu'avec la purge du contenu hors du contexte.

On ne s'en défend pas par la détection, peu fiable sur l'injection indirecte, mais par l'architecture : le modèle n'est pas un périmètre de confiance, et une fois du contenu non fiable ingéré, l'éventail d'actions se referme. Le socle, non négociable dès qu'on ingère de l'externe :

- **Contenu non fiable séparé des instructions** : le récupéré est traité comme une donnée, jamais comme un ordre.
- **Moindre privilège des outils** : le modèle ne voit que ce qu'il a le droit d'appeler (cf. contrôle d'accès ci-dessus).
- **Approbation humaine** sur l'action conséquente et la communication externe.
- **Validation des sorties** d'outils et du modèle avant qu'elles n'agissent.
- **Journal immuable** (§07) : toute action issue d'une injection reste traçable et réversible.

Pour les déploiements à fort privilège, on contraint davantage l'agent par des patterns dédiés, proportionnés au risque : n'autoriser qu'un choix d'actions pré-approuvées (action-selector), figer le plan avant d'exposer le modèle aux sorties d'outils (plan-then-execute), isoler le traitement du contenu non fiable, le purger du contexte après usage, voire séparer un modèle privilégié qui planifie d'un modèle « en quarantaine » qui ne fait que lire le non-fiable (dual LLM). Aucun modèle n'étant immunisé, la sécurité reste une propriété de l'architecture, dont la profondeur s'ajuste au risque.

Conformité par construction

La combinaison contrôle d'accès, séparation du contenu non fiable, approbation humaine, traçabilité complète et traitement souverain des données **outille** les exigences de documentation, de supervision humaine et de transparence attendues en environnement régulé : elle y contribue sans s'y substituer. Le droit à l'effacement est tenu sans rompre l'audit, en rendant une donnée illisible plutôt qu'en brisant la chaîne immuable (§08). La sécurité n'est pas une couche ajoutée, elle est répartie sur les frontières du système.

À RETENIR

- La sécurité porte sur les actions, pas l'affichage : lecture ouverte, mutation gardée.
- Contrôle d'accès à trois niveaux : outil, donnée, requête.
- L'agent est un principal identifié, distinct de l'utilisateur qu'il sert : délégation explicite et bornée, jamais l'emprunt des pleins droits ; l'audit distingue qui a agi et pour qui.
- L'approbation humaine est ancrée dans le contrat de l'outil, garantie par l'infrastructure.
- Une approbation suspend l'agent (état durable) et le réhydrate à la décision, plutôt que de bloquer le processus ; demandes à faible urgence regroupées en file priorisée.
- Sidecars isolés, surface exposée gardée, données souveraines ; le secret d'exécution ne traverse jamais la frontière du modèle ni de l'environnement : substitut côté agent, clé réelle attachée à la frontière réseau.
- Injection de prompt = menace de premier rang, intrinsèque à la mémoire : on ne détecte pas, on contraint par l'architecture (moindre privilège, approbation, séparation du non-fiable, triade létale brisée sur la fenêtre du contexte).

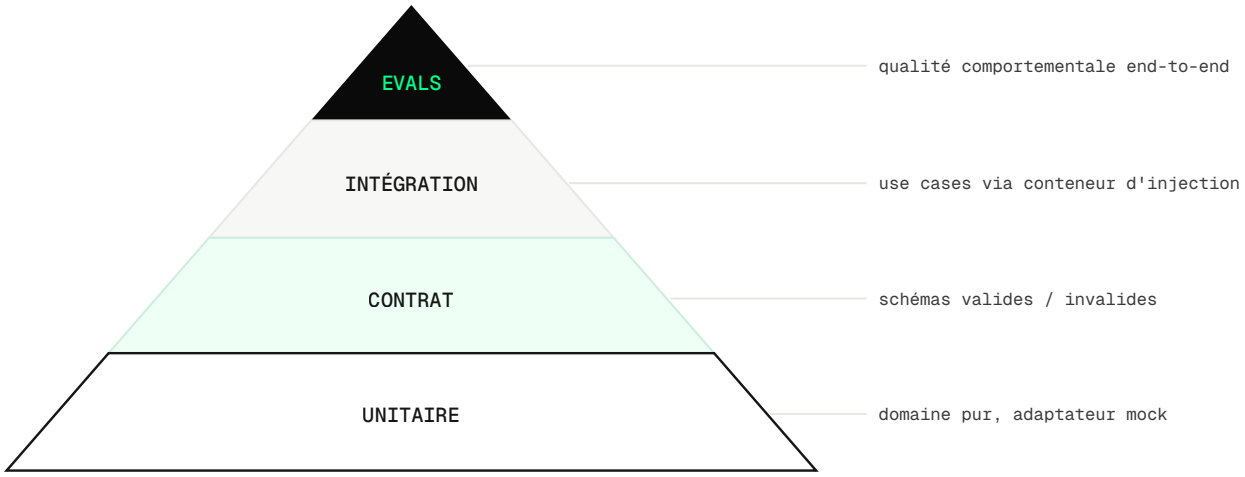
SECTION 12

Évaluation et tests des agents

Un système déterministe se teste ; un système agentique se teste et s'évalue. Le code qui l'entoure se vérifie comme n'importe quel logiciel, mais le comportement du modèle, non déterministe, demande une mesure de qualité dédiée. Cette section est le runbook des deux : la pyramide de tests adaptée aux agents, et la boucle d'évaluation comportementale.

> Distinction fondatrice : un test répond par vrai ou faux sur du code déterministe. Une évaluation note une qualité sur un comportement non déterministe. Les deux sont nécessaires, ils ne se remplacent pas.

La pyramide de tests adaptée aux agents



1 Unitaire, le domaine sans réseau

Le domaine étant pur et le fournisseur de modèle isolé derrière un port, on teste les cas d'usage sans appel réseau, en injectant un adaptateur de modèle mock qui retourne des fixtures déterministes. Les tests sont rapides, reproductibles, et ne dépendent d'aucun fournisseur. La pureté du domaine, posée en Section 02, paie ici directement.

2 Contrat, le schéma comme garde-fou

Chaque entrée et sortie traversant une frontière est validée par un schéma typé. Les tests de contrat vérifient qu'un schéma accepte les charges valides et rejette les invalides : une charge correcte ne lève pas,

une charge malformée lève. Leur vraie valeur est de détecter le décalage silencieux entre le schéma et les données réellement produites, le drift le plus coûteux d'un système typé. Poussé un cran plus loin, le contrat est dirigé par le consommateur : chaque consommateur exprime ses attentes sous forme de contrat exécutable, et un changement du producteur qui les briserait échoue à l'intégration, avant d'atteindre la production. C'est le pendant exécutable de la gouvernance de contrat posée en section 02.

```
charge valide      → schema.parse(charge)   ne leve pas   ok, contrat respecte
champ manquant     → schema.parse(charge)   leve          ok, rejet attendu
mauvais type       → schema.parse(charge)   leve          ok, rejet attendu
```

3 Intégration, l'orchestration avec mocks injectés

Au niveau intégration, on assemble les cas d'usage via le conteneur d'injection en y plaçant des adaptateurs mock (persistance, modèle, exécution). On vérifie l'orchestration et les frontières sans toucher d'infrastructure réelle. Le conteneur d'injection, posé en Section 05, rend ces tests possibles par simple substitution.

4 Évaluations, la qualité comportementale

Au sommet, les évaluations mesurent ce que les tests ne peuvent pas : la qualité du comportement de l'agent sur des scénarios réalistes. Un banc d'évaluation de la mémoire, inspiré des protocoles publics de mémoire longue, couvre six capacités, sur des conversations multi-tours, avec une réponse attendue et une rubrique de notation.

CAPACITÉ ÉVALUÉE	CE QU'ELLE VÉRIFIE
Rappel direct	Retrouver un fait donné plusieurs tours plus tôt.
Rappel multi-contraintes	Combiner plusieurs informations dispersées pour répondre.
Mise à jour de connaissance	Restituer la valeur la plus récente après une correction.
Raisonnement temporel	Restituer ce qui était vrai à une date donnée, et distinguer quand un fait a été appris de quand il était valide. C'est le banc d'essai du modèle bi-temporel de la section 06.
Abstention	Refuser d'inventer quand l'information n'a jamais été donnée.
Longue session	Se souvenir après de nombreux tours, ce qui déclenche le résumé.

La notation suit une rubrique explicite : une réponse complète qui reformule tous les éléments clés vaut le maximum, une réponse partielle qui en manque un vaut moins, une réponse vague mais non fausse vaut la

moitié. Chaque scénario porte une liste de termes attendus et une liste de termes interdits, ce qui rend la mesure de l'abstention aussi importante que celle du rappel : ne pas halluciner est un critère noté.

De la pyramide à la boucle d'exécution

Placée en fin de cycle, l'évaluation passerait pour un simple portillon de préproduction. Pour un système non déterministe soumis à la dérive des modèles, elle doit devenir une **boucle d'exécution continue**. Le flux de traces émis par le middleware (section 05) est échantillonné en arrière-plan, hors du chemin chaud : aucune latence ajoutée au tour. Sur cet échantillon, un scoring **hybride** combine des vérifications déterministes et un **modèle-juge** à rubrique pour les critères comportementaux, l'abstention en tête. Le score alimente un **profil de qualité par agent**, qui étend la notion de confiance du seul credential au comportement observé. Un piège à éviter : le modèle-juge est lui-même un modèle, qui sera mis à jour. Le jour où il change, la série de qualité dans le temps devient incomparable, en silence. On l'ancre donc, comme on promeut le modèle servant : version figée, ou jeu d'ancrage rejoué à chaque changement de juge pour recalibrer la mesure.

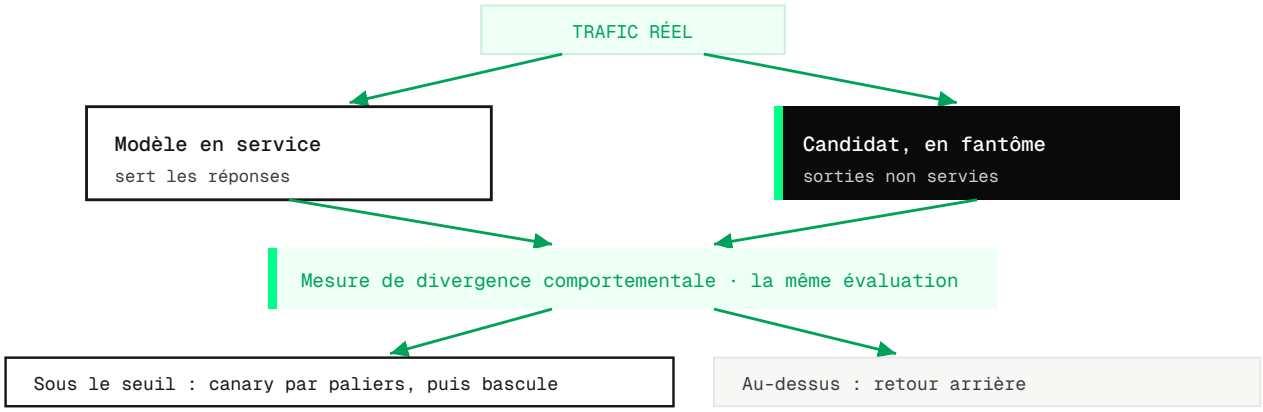
La boucle reste **ouverte à la frontière de l'action**. Elle produit des signaux et des recommandations ; agir sur eux passe soit par un réglage borné dans une enveloppe pré-approuvée et auditée, soit par une validation humaine, jamais par une réécriture autonome. Une boucle fermée naïve qui optimiserait directement le score du juge dériverait vers ce qui le flatte sans améliorer la qualité réelle ; ce qui fait tenir l'auto-réglage, c'est un garde non manipulable, un hold-out vérifiable que l'optimiseur ne voit pas et qui déclenche un retour arrière. Ce garde fort suppose une vérité-terrain vérifiable ; pour les comportements purement subjectifs, où elle manque, le filet redevient l'échantillon humain, et l'enveloppe d'auto-réglage reste étroite. Le principe « l'humain décide » s'applique jusqu'ici.

Pourquoi pas un chœur de vérificateurs ? La tentation est grande de remplacer les garde-fous déterministes par des agents probabilistes qui se relisent les uns les autres, et de laisser l'auto-correction émerger. C'est séduisant, mais plus faible là où une garantie est requise, pour trois raisons. D'abord, on ne fabrique pas une propriété dure, une autorisation, une idempotence, la conformité d'un schéma, à partir de jugements mous : empiler des vérificateurs faillibles empile des incertitudes, il ne les annule pas. Ensuite, l'argument suppose des erreurs indépendantes, or des modèles qui partagent données et biais partagent leurs angles morts : leurs erreurs se corrèlent, au point qu'un vote majoritaire peut amplifier l'erreur commune au lieu de la corriger, et une entrée adverse qui trompe l'un trompe le chœur. Enfin, chaque vérification est une inférence de plus, coûteuse et lente, là où un contrôle déterministe est gratuit et sûr. C'est l'erreur de catégorie que la frontière du modèle évite : déterministe pour ce qui admet une garantie, modèle-juge pour la seule qualité comportementale irréductible, et toujours sous le hold-out non manipulable décrit plus haut. À mesure que les modèles gagnent en fiabilité, la part confiée au juge peut croître ; mais le plancher, sûreté, sécurité, autorisation, audit, ne quitte jamais le déterministe. Et il faut le dire sans détour : le juge que ce cadre retient n'échappe pas à ces limites, faillible et non déterministe lui aussi. Il ne vaut que sous le hold-out non manipulable et derrière le plancher déterministe, jamais comme garantie autonome. La couche d'assurance comportementale est utile, elle n'est pas aussi dure que le socle.

Valider un changement de modèle avant de lui faire confiance

Substituer le modèle est un droit que l'architecture garantit ; lui faire confiance se mérite. Un modèle nouveau ou mis à jour peut altérer le comportement de façons qu'aucun banc d'évaluation hors ligne ne capture, et cela ne se voit que sur du trafic réel : on ne bascule donc jamais d'un seul geste. Le candidat

tourne d'abord en **déploiement fantôme**, derrière le même port, sur le trafic réel mais en silence : ses sorties ne sont pas servies, on mesure leur divergence comportementale avec le modèle en service, par la même évaluation que celle qui surveille la dérive. Si la divergence reste sous un seuil défini à l'avance, on bascule une part croissante du trafic, sous la même surveillance, avec retour arrière immédiat à la moindre régression. C'est la discipline de migration du port modèle, sœur de l'étendre-puis-contracter des contrats (section 02) : on valide en parallèle, puis on bascule par paliers. Une réserve, toutefois : ce gate automatique ne vaut que pour les critères à vérité-terrain vérifiable. Pour les dimensions purement subjectives, où le signal automatique manque, la promotion ne s'auto-déclenche pas, elle reste sous validation humaine.



Runbook d'évaluation

- **Ajouter un scénario** : décrire la conversation multi-tours, la question finale, les termes attendus et interdits, et la catégorie de capacité visée.
- **Exécuter** : rejouer les tours dans l'ordre contre le système courant, poser la question finale, comparer la réponse aux termes attendus et interdits.
- **Noter** : appliquer la rubrique, suivre le score par capacité dans le temps pour détecter les régressions comportementales.
- **Déclencher les évaluations** : relancer le banc d'évaluation sur tout changement qui touche la mémoire, le prompt ou le routage, là où une régression comportementale est possible, et suivre le score par capacité dans le temps.

À RETENIR

- Tester le code déterministe, évaluer le comportement non déterministe : les deux sont nécessaires.
- Pyramide : unitaire sans réseau (adaptateur mock), contrat de schéma, intégration via conteneur d'injection, évaluations au sommet.
- Les tests de contrat détectent le drift schéma vs données, le défaut silencieux le plus coûteux ; dirigés par le consommateur, ils empêchent un changement producteur de casser un consommateur en silence.
- Les évaluations couvrent six capacités, dont le raisonnement temporel (qui éprouve le bi-temporel) et l'abstention : ne pas halluciner est un critère noté.
- L'évaluation n'est pas un portillon final mais une boucle d'exécution continue, échantillonnée hors du chemin chaud, nourrie par le flux de traces.
- Scoring hybride (déterministe plus modèle-juge), profil de qualité par agent ; l'auto-réglage reste borné, gardé par un hold-out non manipulable et l'override humain.
- Un changement de modèle se valide en déploiement fantôme puis par paliers, divergence mesurée par la même évaluation, jamais par simple bascule.
- Un chœur de vérificateurs probabilistes ne fait pas une garantie : leurs erreurs se corrèlent. Le déterministe tient le plancher (sûreté, sécurité, audit) ; le modèle-juge ne couvre que l'irréductiblement comportemental.

PARTIE IV 3 SECTIONS

MATRICE · SYNTHÈSE · PREUVES DANS LE CODE

Décision et synthèse

Trancher : matrice de décision récapitulative, synthèse des recommandations et preuves dans le code.

DANS CETTE PARTIE

MATRICE · SYNTHÈSE · PREUVES

SECTION 13

Matrice de décision récapitulative

Tout ce qui précède tient dans un tableau de décision. Chaque ligne est un arbitrage, avec son défaut recommandé et le signal qui commande d'en changer.

Matrice des arbitrages

DÉCISION	DÉFAUT RECOMMANDÉ	ÉVOLUER QUAND
Langage du coeur	TypeScript (orchestration et interface)	Jamais sans raison ; le polyglotte passe par un sidecar ou un service.
Capacité spécialisée	Sidecar dans le langage de la bibliothèque	Dès qu'une bibliothèque mûre vit dans un autre écosystème.
Composant au chemin critique	Service compilé derrière un contrat	Latence ou débit prouvés insuffisants.
Découpage	Monolithe modulaire hexagonal	Charge, cycle, isolation ou haute disponibilité l'exigent.
Intégration au patrimoine	Couche anticorruption derrière le port	Remplacement progressif, fonctionnalité par fonctionnalité, jamais d'un bloc.
Nombre d'agents	Orchestrateur unique dirigeant des spécialistes	Complexité parallélisable, jamais pour du raisonnement séquentiel.
Abstraction du modèle	Port unique, trois niveaux, SDK direct	Jamais un gros framework de chaînes par défaut.
État	Agent sans état, journal immuable plus vue de travail dérivée	Multi-instance : externaliser les structures en mémoire.
Franchir la frontière du processus	Le plus tard possible, sur déclencheur objectif	Assumer alors cohérence éventuelle, compensation et ordre causal explicite.
Évolution de contrat	Versionné, additif, lu en lecteur tolérant	Changement cassant : étendre puis contracter, sous vérification dirigée par le consommateur.

DÉCISION	DÉFAUT RECOMMANDÉ	ÉVOLUER QUAND
Mémoire et oubli	Score de valeur, invalidation (pas suppression), consolidation avec pointeurs vers les faits sources	Suppression physique réservée aux dérivés invalidés sous rétention.
Évaluation	Tests, plus boucle d'évaluation continue et gardée	Auto-réglage borné seulement, gardé, sous override humain.
Budget et découverte multi-agent	Compteur agrégé central, registre en mémoire	Allocation par blocs et fiches de capacités partagées dès la sortie du processus.
Modèle indisponible	Bascule multi-fournisseur derrière le port	Disponible dès la mise en service réelle.
Changer de modèle (nouveau ou mis à jour)	Valider en déploiement fantôme, puis par paliers	Jamais d'un seul geste ; retour arrière à la moindre divergence.
Attente d'une approbation humaine	Suspendre et libérer (état durable), réhydrater à la décision	Jamais bloquer le processus ; demandes à faible urgence regroupées en file.
Service non critique	Fail-open, mode dégradé signalé	Action sensible : au contraire fail-closed explicite.
Autonomie de délégation	Bornée : gabarits, budgets, approbations	Élargir seulement si le résultat s'améliore réellement.
Entrées non fiables	Contenu récupéré non fiable, moindre privilège, approbation sur l'action	Fort privilège : patterns dédiés (plan figé, quarantaine du non-fiable).
Identité d'agent	Principal propre, distinct de l'utilisateur servi ; délégation explicite, bornée en temps et en périmètre	Jamais l'emprunt des pleins droits de l'utilisateur ; agent hors registre = anomalie de sécurité.
Secrets d'exécution	Substitut côté agent, clé réelle attachée à la frontière réseau, destinations autorisées	Jamais de clé dans le contexte du modèle ni dans l'environnement d'exécution.
Brique partagée (connecteurs, compétences, mémoire transverse, livraison)	Consommée via un port ; adaptateur = client de découverte via un index (section 15)	L'index reste un index ; la livraison livre, l'orchestrateur décide.

Arbre de décision express

La capacité dépend d'une bibliothèque mûre dans un autre langage ?

OUI — sidecar dans ce langage	NON — rester dans le cœur TypeScript

Un composant a-t-il charge, cycle ou isolation propres ?

OUI — le sortir en service	NON — rester en monolithe modulaire

La tâche est-elle parallélisable et à haute complexité ?

OUI — plusieurs agents sous un orchestrateur	NON — orchestrateur unique, un seul fil

Doit-on tenir la charge sur plusieurs instances ?

OUI — externaliser l'état, puis répliquer	NON — mono-instance, état en mémoire ok

À RETENIR

- Chaque décision a un défaut sobre et un déclencheur d'évolution explicite.
- La règle transverse : commencer simple, isoler par contrat, complexifier seulement sur preuve.
- Aucune de ces décisions n'est irréversible, car toutes sont prises derrière une frontière.

SECTION 14

Synthèse et recommandations

Au fond, tout se résume à une posture et à cinq gestes d'ingénierie.

1 Poser les frontières avant la stack

Définir d'abord les contrats (ports) et le protocole d'intégration. Le langage et la topologie deviennent alors des choix locaux et réversibles, pris au bon endroit, pas des paris initiaux.

2 Commencer simple, isoler par contrat, complexifier sur preuve

Monolithe modulaire, orchestrateur unique, port de modèle unique. On externalise un service, on ajoute un agent ou on monte un niveau de modèle uniquement quand un critère objectif ou un gain mesuré le justifie.

3 Sortir le déterministe du modèle

Classification, routage, validation, idempotence et garde-fous restent dans du code testable et cachable. Le modèle garde son budget pour ce que lui seul fait mieux.

4 Rendre l'état explicite, et la mémoire gouvernée

Agent sans état caché. Le journal d'événements est immuable, pour l'audit et le rejeu ; la mémoire de travail en est une vue dérivée, qui oublie par un score de valeur, invalide plutôt que supprime, et se refroidit en archive. Les structures en mémoire s'externalisent pour répliquer : le passage à l'échelle devient une migration d'état, pas une réécriture.

5 Gouverner, tracer et évaluer dès le départ

Chaîne de middleware unique pour la journalisation, l'accès, le coût et l'approbation ; identifiant de requête propagé partout ; bascule multi-fournisseur et mode dégradé pour la résilience. La même trace nourrit une évaluation continue et gardée, qui surveille la dérive comportementale sans jamais laisser le système se réécrire seul. La confiance se construit par l'instrumentation.

Recommandations aux équipes techniques

ACTION	HORIZON
Poser les ports et le protocole d'intégration avant tout choix de stack	Jour zéro

ACTION	HORIZON
Démarrer en monolithe modulaire avec orchestrateur unique	Jour zéro
Brancher journalisation, métriques, identifiant de requête et approbations	Avant toute ouverture
Externaliser l'état dès que le multi-instance est en vue	Sur déclencheur de charge
Brancher une évaluation comportementale continue et gardée sur le flux de traces	Avant toute ouverture
Introduire sidecar, service ou agents additionnels sur critère objectif	Sur preuve de besoin

Position d'ensemble

Faire tourner un système agentique fiable n'est pas un problème de framework ni de langage, c'est un problème d'**architecture autour du modèle**. En posant les frontières d'abord, le choix full TypeScript ou polyglotte et le choix monolithe ou microservices cessent d'être des dilemmes structurants : ce sont des arbitrages locaux et réversibles, pris derrière des contrats stables. Et puisque le modèle lui-même vit derrière un port, il reste un composant que l'on remplace, jusqu'à son paradigme : ce qui se capitalise, ce n'est pas le modèle, c'est l'architecture qui l'encadre.

Ce que ce cadre établit, il le démontre : les principes ne sont pas spéculatifs, ils vivent dans trois systèmes construits indépendamment qui convergent sans partage de code (annexe). Pour valider une architecture, c'est la preuve la plus solide qu'on puisse fournir. Ce qui relève d'une autre discipline, l'adoption organisationnelle (qui possède quoi, cycles de vie, instances de décision) et la stratégie de données d'entreprise, n'est pas un angle mort de l'architecture mais une couche distincte, qui conditionne tout projet quelle qu'en soit l'architecture. Les confondre reviendrait à reprocher à un plan de structure de ne pas être un plan de financement. L'architecture, elle, tient, et elle est éprouvée.

La posture tient en une phrase : **commencer simple, isoler par contrat, complexifier seulement sur preuve**, en gardant l'état explicite, la gouvernance par conception et la souveraineté par construction. C'est la déclinaison opérationnelle, pour les équipes, d'une conviction simple : l'architecture d'abord, le modèle ensuite.

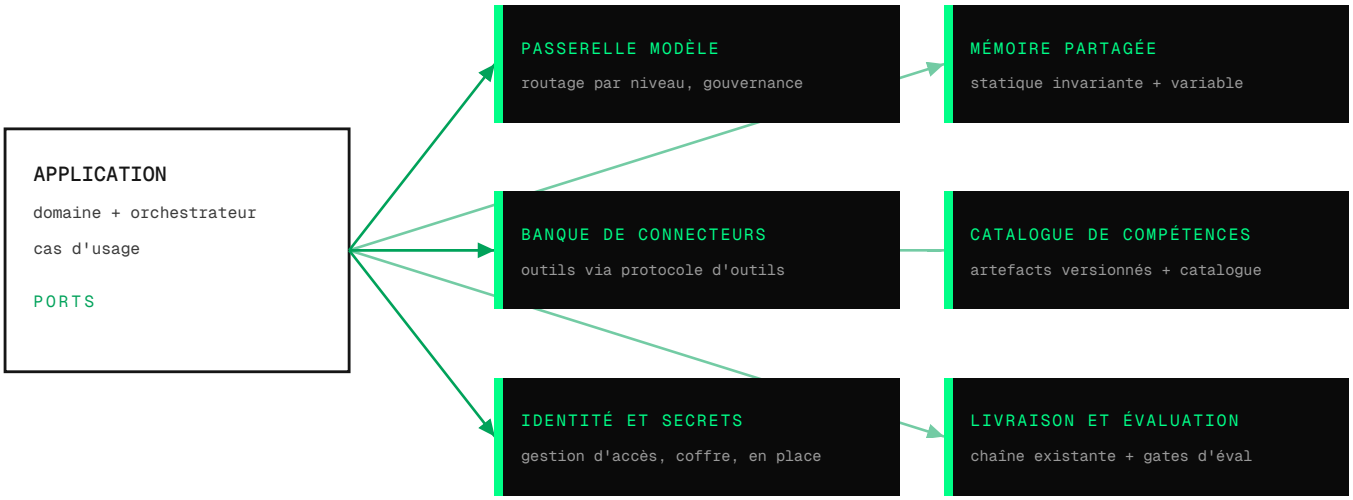
SECTION 15

Au-delà de l'application, les briques partagées

Tout ce qui précède traite une application : un domaine, ses ports, ses adaptateurs. Mais certaines capacités ne sont l'adaptateur d'aucune application en particulier : elles sont partagées par plusieurs, par nature. Une banque de connecteurs, un catalogue de compétences, une mémoire transverse, une chaîne de livraison et d'évaluation, une identité d'agent : ce sont des briques d'entreprise. Cette section en fixe les principes ; le paysage des emplacements et des offres, mouvant par nature, est volontairement laissé hors de ces pages, qui ne retiennent que ce qui tient quels que soient les fournisseurs.

Le déplacement, et la règle qui ne bouge pas

Loger une brique partagée dans une application la condamne à la duplication ; la loger ailleurs ne sort pas du modèle. La brique quitte l'application, mais l'application la consomme toujours à travers un port, un contrat stable ; ce qui change, c'est que l'adaptateur derrière ce port pointe vers un service d'entreprise au lieu d'une structure locale. La question n'est donc jamais port ou pas port : c'est quel emplacement derrière le port, infrastructure existante, plateforme dédiée ou service managé. Cet arbitrage d'emplacement est réversible précisément parce que le contrat, lui, ne bouge pas.



La brique quitte l'application ; l'application la consomme toujours via un port, un contrat stable.

Les emplacements, et les critères qui arbitrent

Derrière chaque port, cinq familles d'emplacement. Dans l'application elle-même : un adaptateur local, simple mais dupliqué dès la deuxième application. L'infrastructure existante, passerelle d'API, dépôt d'artefacts, chaîne d'intégration, gestion d'accès : souveraine et amortie, mais pas conçue pour l'agentique.

La plateforme interne dédiée : contrôle et réemploi, au prix de la construction et de l'exploitation. Le service managé d'un fournisseur d'infrastructure : mûr et rapide, moins souverain, dépendant. Le runtime managé d'un fournisseur de modèle : la vélocité maximale et la dépendance la plus forte, celle au fournisseur du composant central, avec une souveraineté partielle quand l'exécution des outils reste chez soi. Six critères arbitrent, et eux seuls sont stables : souveraineté et contrôle, réemploi entre applications, vélocité de mise en œuvre, coût et dépendance, réutilisation de l'existant, conformité. La grille est pérenne ; son remplissage dépend d'un paysage et d'un contexte datés, volontairement hors de ces pages.

Le motif récursif : contrat, index, implémentation

À l'échelle d'une application, l'adaptateur est local. À l'échelle d'une infrastructure, comme la brique est partagée, l'adaptateur devient un client mince qui résout la capacité à travers un index commun, puis l'appelle par son contrat. C'est l'extension directe du registre à deux adaptateurs (section 05) : le motif est récursif. Trois choses restent distinctes et ne doivent jamais fusionner : le contrat, qui dit ce que la brique promet ; l'index, qui publie et fait découvrir ; l'implémentation partagée, qui honore le contrat. Ce ne sont pas les adaptateurs qui deviennent l'index : c'est leur découverte qui passe par un index.

> Contrat, index, implémentation : trois choses distinctes. L'index n'est pas un cerveau.

Les gardes, amplifiées à l'échelle

Deux gardes de cette doctrine s'appliquent telles quelles et deviennent plus critiques à l'échelle d'une plateforme. La première : l'index, ou le catalogue, reste un index, pas un cerveau ; il publie et fait découvrir, il n'orchestre pas, sous peine de recréer le bus d'intégration hypertrophié (section 05). Le piège prend ici une forme précise : laisser la chaîne de livraison ou le catalogue décider à l'exécution quelle compétence appeler. La livraison livre ; l'orchestrateur de l'application décide. La seconde : une compétence ou un connecteur tiers est une entrée non fiable ; moindre privilège, approbation sur l'action, et la triade se brise sur la fenêtre du contexte (section 11). Un catalogue partagé amplifie cette surface : la curation et la revue de sécurité n'y sont pas optionnelles.

Le moteur d'exécution durable, un adaptateur parmi d'autres

Quand les tours s'allongent, attentes, reprises, compensations, un moteur d'exécution durable peut porter ces propriétés en brique partagée : journal d'étapes, ré-essai, suspension et réveil. Il reste un adaptateur derrière les ports, et la garde est la même que pour l'index : il porte le flot de contrôle et la durabilité, jamais le raisonnement ni le métier. L'appel de modèle y demeure une activité dont la sortie est enregistrée : on la

relit, on ne la rejoue pas (section 07). Laisser le moteur d'orchestration décider du métier reviendrait, là encore, à refaire le bus hypertrophié.

La matrice des briques

Comme pour la matrice de décision de la section 13, chaque brique a un défaut sobre et un déclencheur explicite ; le défaut s'applique tant qu'aucun déclencheur ne commande d'en changer.

BRIQUE	DÉFAUT RECOMMANDÉ	BASCULER QUAND
Passerelle modèle	La passerelle d'API existante, vue port modèle	Les fonctions agentiques (routage par niveau, gouvernance de coût) la débordent.
Banque de connecteurs	L'infrastructure existante porte transport et sécurité	Elle plafonne sur la sémantique d'outils et la découverte : brique dédiée ou managée.
Catalogue de compétences	Dépôt d'artefacts existant pour le paquet versionné	La curation et la revue de sécurité l'exigent : catalogue dédié, jamais sans curation.
Mémoire partagée	Dans l'application pour la mémoire de travail	Un besoin transverse réel entre applications, pas une cible lointaine.
Registre de découverte	Table en mémoire (section 05)	Franchissement d'une frontière de processus : registre partagé (section 07).
Exécution durable	Boucle simple dans l'application	Tours longs, attentes d'approbation, reprises après panne : moteur durable, qui ne porte jamais le métier.
Livraison et évaluation	Chaîne d'intégration existante plus gates d'évaluation	L'évaluation continue gardée (section 12) déborde la chaîne : brique d'évaluation dédiée.
Observabilité	Stack existante pour le transport des traces	Donnée régulée : auto-hébergée, car les traces sont de la donnée.
Identité et secrets	Gestion d'accès et coffre existants	Identité d'agent et délégation bornée requises (section 11) : brique dédiée.

La souveraineté se gradue par classe de donnée

La souveraineté n'est pas un interrupteur, c'est un dégradé : elle se décide par classe de donnée, pas en bloc. La règle qui en découle : souverain par défaut partout où la souveraineté ne coûte que de l'exploitation, passerelle, mémoire, registre, exécution, contrôle d'accès ; le seul tier où elle peut coûter en qualité est le modèle, et le port modèle rend précisément son routage par sensibilité local et réversible, meilleur moteur disponible pour la donnée ordinaire, moteur souverain pour la donnée régulée. Une conséquence souvent

oubliée : les traces d'exécution contiennent les prompts et les sorties, donc de la donnée ; elles suivent le même dégradé, et leur export vers un service tiers est une exfiltration comme une autre (section 10).

La conformité consomme l'architecture

La conformité réglementaire n'est pas un chantier séparé : c'est une couche qui consomme les primitives déjà posées, journalisation immuable, supervision humaine, moindre privilège, gouvernance et effacement de la mémoire, provenance. Une extension suffit à l'outiller : le registre de découverte s'étend en registre des usages, qui porte, par déploiement, la classe de risque, les classes de données et le propriétaire, sous un responsable identifié et une revue périodique. Car le risque se classe par déploiement, pas par plateforme : le même socle, réutilisé pour un usage sensible, fait basculer ce déploiement-là dans un régime plus exigeant, sans que l'architecture change.

La leçon des couches

Une leçon d'histoire, pour clore. Les infrastructures se sédimentent : chaque génération dépose une couche d'abstraction au-dessus de la précédente, et toute couche finit par se banaliser pendant qu'une nouvelle s'empile au-dessus d'elle. À chaque transition, les systèmes qui consommaient la couche du dessous par contrat l'ont traversée ; ceux qui s'y étaient soudés l'ont payée au prix fort. Les briques de cette section n'échapperont pas à la règle : elles se banaliseront à leur tour. C'est précisément pourquoi on les consomme derrière des ports et qu'on ne s'y soude pas : la discipline des contrats ne protège pas seulement d'un changement de fournisseur, elle fait traverser les couches elles-mêmes.

À RETENIR

- Certaines briques sont partagées par nature ; la consommation reste par port, la question est l'emplacement derrière le port, et cet arbitrage est réversible.
- Le motif est récursif : l'adaptateur devient un client mince qui résout via un index ; contrat, index et implémentation restent trois choses distinctes.
- L'index n'est pas un cerveau ; la livraison livre, l'orchestrateur décide ; un tiers est une entrée non fiable, la curation n'est pas optionnelle.
- Le moteur durable porte le flot de contrôle, jamais le raisonnement ; l'appel modèle reste une activité relue.
- Cinq familles d'emplacement derrière chaque port ; six critères stables arbitrent, leur remplissage est un exercice daté.
- Chaque brique a un défaut sobre et un déclencheur de bascule : la matrice des briques prolonge la matrice de décision.
- La souveraineté se gradue par classe de donnée ; seul le tier modèle peut coûter en qualité ; les traces sont de la donnée.
- La conformité consomme l'architecture ; le registre des usages en est la pièce maîtresse, et le risque se classe par déploiement.
- Toute couche finit par se banaliser : on la consomme par contrat, on ne s'y soude pas ; la discipline des ports fait traverser les transitions.

ANNEXE

Preuves dans le code

Cette annexe ancre les principes exposés dans des artefacts réels. Ils vivent dans trois systèmes construits séparément, sans partage de code : une chaîne d'agents de test issue de plusieurs expérimentations (système 1), une plateforme d'accompagnement de projets bâtie sur un graphe de connaissances (système 2), et un assistant personnel à mémoire longue (système 3). Les deux premiers couvrent l'ensemble des piliers et sont mis en regard ci-dessous ; le troisième, spécialisé, éprouve les piliers de la mémoire et de l'évaluation. Que ces trois systèmes convergent sur les mêmes patterns, sans partager une ligne de code, est un indice sérieux. Restons précis sur sa portée : c'est une validation par l'usage, pas une preuve d'indépendance, puisqu'une même main les a conçus. Absence de code partagé ne vaut pas indépendance de conception.

> Les artefacts ci-dessous sont décrits par leur fonction, pas par leur emplacement. Ce qui compte n'est pas le nom d'un fichier mais le fait que la brique existe et que les systèmes y convergent, vérifiable par une équipe disposant de l'accès aux dépôts. Aucun chemin, nom de fichier ni nom commercial n'est cité.

Socle, frontières et fournisseurs

PRINCIPE	SYSTÈME 1, AGENTS DE TEST	SYSTÈME 2, PLATEFORME PROJETS
Domaine pur, ports et adaptateurs	Domaine et cas d'usage purs, adaptateurs secondaires séparés du cœur	Ports du domaine isolés, adaptateurs de persistance dédiés
Injection de dépendances explicite	Dépendances passées par constructeur, fabriques centralisées	Conteneur d'injection à point d'entrée unique
Frontière du modèle, port à niveaux	Port de fournisseur de modèle, trois niveaux rapide, équilibré, profond	Adaptateurs de modèle derrière un routeur de complexité
Routage par niveau de modèle	Niveau déclaré par agent, montée en cas de doute	Classement de la tâche en simple, modéré, complexe
Détection de fonctionnalités, bimodal	Détection au démarrage, modes minimal, complet, intégré	Détection par service, dégradation gracieuse si absent

Orchestration, outils et gouvernance

PRINCIPE	SYSTÈME 1, AGENTS DE TEST	SYSTÈME 2, PLATEFORME PROJETS
Orchestrateur sans logique métier	Cas d'usage orchestrateur qui compose, sans métier propre	Point d'entrée d'orchestration mince, qui délègue au registre
Registre d'outils par assemblage	Outils assemblés et exposés par un constructeur dédié	Registre d'outils unique, enregistrement et résolution centralisés
Chaîne de middleware transverse	Traçage et enveloppe d'approbation autour de chaque outil	Middleware de journalisation, de métriques et garde d'accès aux données
Filtrage des outils par accès	Exposition contextuelle des outils au modèle	Sélection des outils selon le niveau d'accès, en amont
Approbation humaine sur l'action	Demande d'approbation déclarée dans le contrat de l'outil	Garde d'accès aux données avant toute mutation
Garde-fous et budgets	Filet de sécurité autour des outils, compteurs, limite de débit	Limites de débit, accès lecture seule pour les requêtes génériques

Exécution, état et observabilité

PRINCIPE	SYSTÈME 1, AGENTS DE TEST	SYSTÈME 2, PLATEFORME PROJETS
Capacité spécialisée en sidecar	Moteur d'exécution piloté via le protocole d'outils, port dédié	Serveurs d'outils externes via client dédié
Surface d'outils exposée et gardée	Serveur d'outils publié, protégé par une garde dédiée	Intégration de serveurs d'outils, garde et limites
Observabilité par événements	Événements de cycle tracés, export de traces	Middleware de métriques, journaux structurés
État externe et topologie	Persistance derrière son port, structures en mémoire identifiées	État temps réel et migrations dédiés, déploiement orchestré

! La délégation dynamique est déjà amorcée

Point notable pour la trajectoire vers l'orchestration adaptative : dans le système 1, l'orchestrateur dispose déjà de briques de délégation dynamique. On y trouve un invocateur de sous-agents, des gabarits de sous-agents, une délégation parallèle et par gabarit, une mémoire procédurale qui enregistre et rejoue des

routines, et une évolution de comportement soumise à validation humaine, encore tenue en statut à revoir. Le palier 3 de la maturité n'est donc pas une cible lointaine : son socle existe déjà.

! Mémoire, évaluation, budgets : où vivent les preuves

Les trois renforts de cette édition s'appuient eux aussi sur du code existant. Pour la mémoire (section 06), le troisième système, dédié à la mémoire longue, applique déjà un oubli non destructif : une tâche périodique met à jour le score de récupération sans supprimer les souvenirs, seuls des liens dérivés sont élagués sous seuil, et l'importance comme la saillance sont produites en sous-produit de l'extraction asynchrone, sans appel de modèle dédié ; la validité dans le temps, deux temps par fait, en est l'extension naturelle. Pour l'évaluation (section 12), le système 1 dispose déjà d'un banc d'évaluation de la mémoire, d'un export de traces, d'une boucle de feedback sur la fiabilité dont l'action reste volontairement manuelle, et d'un score de confiance par agent : la boucle d'exécution gardée en est l'aboutissement. Pour la délégation distribuée (sections 04, 05, 07), le système 1 signe déjà des fiches d'agent et borne le budget par un plafond agrégé en mode autonome ; la découverte par fiches et l'allocation de budget par blocs sont les adaptateurs distribués de ces mêmes ports.

> À RETENIR
>
> - Chaque principe exposé correspond à un artefact réel, dans trois systèmes construits séparément.
> - Leur convergence sans partage de code est un indice fort par l'usage ; pas une preuve d'indépendance, une même main les a conçus.
> - La délégation dynamique du palier 3 est déjà partiellement implémentée, pas seulement projetée.

Postface

Au départ de ce document, il y avait une inquiétude d'ingénieur : presque tout ce qui se dit agentique ne tient pas en production. Au terme, ce qui reste tient en une inversion. Le composant le plus impressionnant de nos systèmes est aussi le moins fiable ; tout l'édifice consiste à traiter cette merveille comme on traite un composant faillible, derrière des contrats, sous des gardes, avec des preuves. Ce n'est pas du scepticisme, c'est l'inverse : on ne discipline avec ce soin que ce qu'on a l'intention d'utiliser longtemps.

On en tire une leçon d'humilité et une de confiance. L'humilité : presque rien ici n'est inventé. La plomberie vient de décennies de systèmes distribués, et la tentation qu'il fallait vaincre n'était pas technique, c'était celle de tout réinventer parce que le composant est nouveau. La confiance : les quatre vraies nouveautés ont des réponses d'architecture, pas des incantations. Un système agentique fiable n'est pas un exploit, c'est une discipline.

Ces pages ne tournent pas. Une doctrine ne livre rien : la suite logique, pour qui les referme, n'est pas d'en débattre mais de poser un plancher exécutable, branché sur un flux réel, et de le laisser prouver. Tout ce qui précède est ordonné pour ça : ce qu'on pose au jour zéro tient en peu de choses, le reste attend son déclencheur. La gouvernance n'est pas davantage un chapitre qu'on ajoute à la fin : elle est déjà dans les frontières, et elle commence le premier jour ou jamais.

Ce document s'applique aussi à lui-même sa règle. Il a été écrit comme on tient un système : chaque position vérifiée contre l'existant, confrontée à l'état de l'art, verrouillée par une décision tracée avant d'être écrite. Il se révisera de la même façon, sur preuve, jamais par mode. C'est un document vivant ; s'il cesse un jour d'être vrai, il faudra le corriger, pas le défendre.

Restent les défis, et il serait malhonnête de clore sans les nommer. Les moteurs gagneront en fiabilité, et la frontière de ce qu'on leur délègue s'élargira ; elle s'élargira par paliers gardés, ou elle se paiera. Le plancher, lui, ne bougera pas : la sûreté, l'autorisation, l'audit ne quitteront jamais le déterministe. Et le chantier le plus dur n'est pas dans ces pages : c'est l'adoption, qui possède quoi, qui répond de quoi, qui apprend quoi. Il est humain avant d'être technique, et aucune architecture ne l'épargne ; elle peut seulement le rendre possible.

S'il ne fallait garder qu'une phrase, ce serait celle qui a ouvert ce document et qui peut le fermer : l'architecture d'abord, le modèle ensuite. Commencer simple, isoler par contrat, complexifier seulement sur preuve. Le reste est de l'exécution, et l'exécution est le seul juge.

THIBAULT SOURIS · 2026

L'architecture d'abord, le modèle ensuite.

Presque toutes les applications se disent agentiques ; presque aucune ne tient en production. L'écart n'est pas un problème de modèle, c'est un problème d'architecture.

Ces pages posent le cadre de décision qui fait la différence : où tracer les frontières, comment rendre le langage et la topologie réversibles, comment garder l'état explicite, gouverner la mémoire, sécuriser les entrées non fiables et évaluer un comportement non déterministe. Du premier port posé au jour zéro jusqu'à la boucle qui surveille la dérive, chaque section transforme un pari initial en décision réversible, prise derrière des contrats stables. Le modèle y reste ce qu'il doit être : un composant que l'on remplace, jusqu'à son paradigme, pendant que l'architecture se capitalise.

Pour quiconque conçoit, exécute ou défend de tels systèmes, et veut le faire sur des bases qui durent.

CONCEVOIR ET EXÉCUTER DES SYSTÈMES AGENTIQUES

Thibault Souris.