Ahoj kluci, tady Čáťa. S Ondrou jsme dnes prošli SAGABrain docela od základů až po hlasové rozhraní. Sepisuju vám naši společnou „esenci“, ať ji můžete přidat do Hubu a rozpracovat. Berte to jako architektonické poznámky a návrhy, ne jako požadavek všechno překopat — řada věcí už u vás podle Ondry funguje.

1. Základní model SAGABrainu

Ondrův koncept mi dává smysl jako kognitivní architektura inspirovaná mozkem, nikoli jako jeden obří AI agent.

- **Hippocampus** = lokální server, paměť, indexace, databáze, recepce vstupů, watchery, trvalé služby.
- **Thalamus** = Synology + Hub. Fyzická data, společné adresáře, timeline, handoff, pravidla, zákony, skilly, projekty, témata, agent registry atd.
- **Cortex** = inteligentní agenti/modely. Dnes primárně cloud OpenAI + Anthropic, později část inference lokálně.
- **Vstupy/výstupy** = Synology Chat, IdeaX, iCloud, poznámky, kalendář, e-mail, soubory, web, později hlas/telefon, Home Assistant a další brány.

Důležitý princip: **SAGABrain není jeden agent. Je to celý systém.**

2. Dočasný Hippocampus

Aktuální provizorní PC:

- Ryzen 3 2200G
- 8 GB RAM
- SSD RAID na desce
- další SSD

- Windows pryč, čistý Linux
- perspektivně max. cca 16 GB RAM v tomto stroji

Na první fázi je to použitelné pro:

- recepci
- watchery
- API
- SQLite
- fronty
- plánovače
- indexaci
- embeddings menšího rozsahu
- lokální OCR/STT apod.
- orchestraci agentů

Není to cílový stroj pro velké lokální LLM.

Druhý kandidát je Intel i5 cca kolem roku 2020, deska se čtyřmi DDR4 sloty, možnost 24–32 GB RAM. Procesor nemá integrovanou grafiku, ale pro headless server stačí jakákoli levná kompatibilní GPU jen pro instalaci/servis, případně později úplně bez ní, pokud deska umí headless boot.

Pro SAGABrain bude v této fázi často důležitější **RAM než hrubý CPU výkon**.

3. Local-first

Silně preferovat:

lokální zpracování → silnější lokální zpracování → cloud až jako další úroveň

Důvody:

- soukromí
- zdravotnická data
- dlouhodobé náklady
- kontrola nad systémem
- energetická/ekonomická efektivita
- nezávislost na jednom providerovi

Cloud ale nezakazovat. Má být nástrojem pro případy, kdy poskytne výrazně lepší výsledek.

Routing by proto neměl být pouze „který agent?“, ale zároveň:

- lokál vs. cloud
- levný vs. drahý model
- rychlý vs. reasoning model
- citlivá vs. necitlivá data

4. Recepční

Současný Receptionist je dobrý základ.

Má být permanentně nebo téměř permanentně aktivní, levný a rychlý.

Jeho úloha není být nejchytřejší člen systému.

Má:

- sledovat vstupy
- přijímat nové události
- klasifikovat je
- založit/zapsat událost do timeline
- spojit ji s projektem/tématem/kontextem
- jednoduché úkoly řešit sám
- složitější eskalovat

- vybrat vhodného agenta
- vytvořit/předat handoff
- sledovat, zda byla práce převzata
- kontrolovat rozpracované úkoly
- připomínat agentům visící práci
- při problému aktivovat další pomoc
- při potřebě lidského rozhodnutí poslat rozhodnutí do IdeaX

Recepční je tedy něco jako **triage + dispatcher + watchdog**.

Neměl by ale postupně nabobtnat do agenta, který dělá úplně všechno.

5. Specializace agentů

Dává smysl vytvořit více jasně pojmenovaných rolí/ instancí.

Nemusí mít každý jiný model. Může jít o stejný engine s jiným:

- systémovým promptem
- identitou
- oprávněními
- skilly
- datovými přístupy
- cenovým limitem
- eskalačními pravidly

Příklady rolí:

- Receptionist
- Planner
- Security / Policy

- Document / File worker
- OCR / Vision
- Audio / STT
- Research
- Coding / Infrastructure
- Medical-specific pracovní agent mimo osobní SAGABrain
- Cost / Efficiency
- Evaluator / Critic
- případně další doménoví specialisté

Smyslem je, aby **kompetence byla explicitní**, ne „všichni umí všechno“.

6. Planner

Planner nemusí být stále spuštěný.

Receptionist může jednoduché úkoly rozdělovat sám.

Planner se probudí u úloh, které jsou:

- vícekrokové
- multiagentní
- dlouhodobé
- mají závislosti
- vyžadují více systémů
- mají komplikovaná oprávnění

Planner:

- rozloží cíl na kroky
- přiřadí agenty
- určí závislosti
- nastaví očekávané výsledky
- vytvoří handoffy

- případně určí kontrolní body

Tím není Receptionist nucen používat drahý reasoning model na každý triviální vstup.

7. Escalation policy

Nenechávat čistě na tom, zda si levný model „myslí“, že odpověď zvládne.

Modelové self-confidence je pouze pomocný signál.

Eskalaci řídit kombinací:

- explicitních pravidel
- typu dat
- rizika
- složitosti
- počtu kroků
- požadované přesnosti
- finančního limitu
- nejistoty modelu
- výsledku případné kontroly

Např.:

- běžná systémová událost → Receptionist
- soubor → File worker
- obraz → Vision
- zdravotnický obsah → security/policy větev
- komplexní lidský dotaz → silnější model
- úloha vyžadující plán → Planner

8. Policy/Security agent

Tuhle roli bych explicitně oddělila.

Nemusí běžet nepřetržitě jako LLM.

Velká část může být deterministický **policy engine** a AI se zavolá jen tam, kde pravidla nestačí.

Rozhoduje například:

- smí data do cloudu?
- musí se anonymizovat?
- může agent daný objekt číst?
- může ho zapisovat?
- jaké nástroje smí použít?
- jak vysoké riziko má požadovaná akce?
- vyžaduje lidské potvrzení?
- existuje už delegované oprávnění?

Security tedy není jen scanner, ale **autorizační a datový firewall**.

9. Oprávnění a delegace

Ondra už tento princip používá a považuju ho za jeden z nejdůležitějších prvků architektury.

Používat zásadu:

nejmenší nutné oprávnění – least privilege

Agent nedostává právo „na počítač“.

Dostává právo:

- k určité akci
- nad určitým objektem
- v určitém projektu
- na určitou dobu
- případně jen v rámci konkrétního chainu úkolu

Pokud Ondra schválí komplexní úkol, mělo by být možné spolu s úkolem vytvořit delegaci pro jeho předem předvídané podkroky.

Pak není nutné člověka znovu otravovat u každého kroku.

Prakticky bych to časem formalizovala jako něco typu:

delegation capability / task token

obsahující minimálně:

- issuer
- task/chain ID
- povolené capabilities
- scope
- recipient nebo skupinu
- expiraci
- případně počet použití
- audit trail

Agent B pak nemusí věřit agentovi A „na slovo“. Ověří delegaci.

10. Handoff

Handoff není jen předání souboru.

Je to interní pracovní komunikační vrstva agentů.

Používá se na:

- předávání úkolů
- žádost o pomoc
- konzultaci
- opravu chyb
- debugging přístupu/konektorů

- nabídnutí alternativní cesty
- předání kompetence
- koordinaci multiagentní práce

Je užitečné, že agenti umí aktivně požádat jiného agenta o pomoc, pokud narazí na problém.

To je něco, co bych zachovala a dále posílila.

11. Timeline a chain

Timeline je zásadní.

Každá významná událost a úkol by měly mít:

- původ
- čas
- projekt/téma
- chain/task ID
- zodpovědného agenta
- stav
- případný parent task
- handoffy
- rozhodnutí
- použitý model/nástroj
- náklady
- chyby
- konečný výsledek

Receptionist/Watchdog pak nemusí „přemýšlet z ničeho“, ale pouze vyhodnocuje explicitní stav systému.

12. Idle konsolidace / „spánek“

Nemusí existovat jedna pevná noční dávka.

Líbí se mi spíš princip:

když není prioritní práce, systém konsoliduje paměť a analyzuje sám sebe.

Může:

- kontrolovat otevřené úkoly
- hledat osiřelé chainy
- propojovat související události
- doplňovat index
- slučovat duplicity
- vytvářet souhrny
- hledat chyby
- analyzovat nepovedené odpovědi
- aktualizovat lessons learned
- navrhovat změny pravidel/skillů

To je pěkná analogie ke konsolidaci paměti.

13. Self-improvement

SAGABrain už obsahuje filozofii učení z chyb.

Doplnila bych ji o více strukturované metriky.

U každé relevantní operace ukládat:

- úspěšnost
- čas
- cenu
- tokeny
- model
- lokál/cloud
- počet eskalací
- případnou lidskou opravu

- chybu/halucinaci
- příčinu, pokud se podaří určit
- jak byla chyba opravena

Nad tím může periodicky pracovat **Evaluator/Optimizer**.

Pozor: systém by neměl své bezpečnostní zákony a kritická pravidla automaticky přepisovat bez kontroly. Může navrhovat změnu a testovat ji.

14. Cost / Efficiency agent

Samostatná ekonomická/optimalizační role.

Nemusí běžet pořád.

Vyhodnocuje:

- cenu podle providerů/modelů
- spotřebu tokenů
- počet volání
- rychlost
- lokální výpočet
- zbytečné opakování kontextu
- případy, kdy byl použit zbytečně drahý model

Ale neoptimalizovat jen cenu.

Cílová metrika by měla kombinovat přibližně:

kvalita + bezpečnost + latency + cena + energetická náročnost

Někdy je správné utratit víc, pokud je úloha důležitá.

15. SQLite / lokální databáze

Databázi aktivního systému držet lokálně na serveru, ne provozovat aktivní SQLite přes síťový share Synology.

Synology může držet:

- Hub
- zdrojová data
- backup databáze
- snapshoty
- archiv

Aktivní SQLite ideálně lokálně na Linux filesystemu, s pravidelným bezpečným backupem na Synology.

16. Ambulance ≠ osobní paměť SAGABrainu

Velmi důležitá hranice.

Systém ambulance má být **datově a paměťově oddělený** od osobního SAGABrainu.

Do osobního SAGABrainu není důvod ukládat:

- identitu každé pacientky
- zdravotnickou dokumentaci
- zdravotní historii pacientek

Mohou ale sdílet:

- architektonické principy
- obecné skilly
- software
- anonymní know-how
- výpočetní infrastrukturu

Motto:

sdílený výpočet, nikoli sdílená paměť.

17. Domácí výpočet pro ambulanci

Až bude doma dostatečně výkonný server, může bezpečná část ambulantního systému využívat jeho výpočetní kapacitu.

Například:

- OCR
- image processing
- převod dokumentů
- lokální STT
- embeddingy
- lokální LLM
- analýzu tabulek
- předzpracování UZ dat

Data ale zůstávají v definované bezpečné zóně a cestují pouze přes zabezpečenou privátní síť/VPN.

Tedy fyzicky může výpočet běžet doma, ale logicky jde stále o oddělenou zdravotnickou workload/security domain.

18. Ultrazvuk a medicínská data

U strukturovaných medicínských dat:

zdroj pravdy jsou strukturovaná čísla/data, ne text vytvořený LLM.

Pipeline může být:

data z přístroje → validace → strukturovaná data → deterministic/template report → LLM případně formulace/ interpretace

LLM může:

- formulovat text
- kontrolovat konzistenci
- nabídnout návrh závěru
- upozornit na nesrovnalost

Nemá nahrazovat zdrojové hodnoty.

Pokud je nutný cloud, poslat jen minimální anonymizovaný kontext potřebný pro daný úkol.

19. Telefonní AI pro ambulanci

Samostatná větev.

Např. ElevenLabs nebo jiná kvalitní STT/TTS/realtime hlasová služba + LLM.

Telefonní recepční může:

- přijmout hovor
- provést základní triage
- odpovídat na administrativní otázky
- pracovat podle jasných ambulantních skillů
- eskalovat člověku
- případně později provést bezpečně povolené úkony

Medicínská a osobní data tohoto systému zůstávají mimo osobní SAGABrain.

20. Hlasové rozhraní SAGABrainu

Pro osobní SAGABrain bych nestavěla druhou paralelní orchestraci.

Architektura:

hlas → STT → stávající Receptionist → agent/ orchestrace → odpověď → TTS → Ondra

Telefonista/voice gateway je tedy primárně **I/O vrstva**, ne další hlavní mozek.

Výhoda:

chat, web, hlas, iCloud, mail atd. používají stejnou logiku, stejné policy a stejné agenty.

Voice gateway může mít:

- streaming STT
- VAD
- interruption/barge-in
- TTS
- různé hlasy jednotlivých agentů

Ondra by tedy mohl říct:

„Spoj mě s X.“

A stejný orchestration layer rozhodne, koho aktivovat.

Je možné mít různé hlasy agentů, takže člověk intuitivně pozná, s kým právě mluví.

21. Jak zapojit Čáťu – dnešní ChatGPT bez vlastního API

Přímo připojit tuto konkrétní ChatGPT/voice instanci jako permanentního člena privátního Hubu pravděpodobně neumíme.

Ale lze udělat velmi použitelný bridge.

SAGABrain připraví pro Čáťu webovou stránku například:

```
IdeaX / Čáťa
```

Na ní bude podle konkrétního úkolu:

- **esence**
- relevantní výpis z Hubu
- aktuální timeline
- relevantní handoff
- projektový kontext
- otázky pro Čáťu
- případně pole pro odpověď

Nemá se exportovat celý Hub. Jen kontext potřebný pro konkrétní interakci.

22. Jednorázový/time-limited přístup

Místo permanentního veřejného URL lze stránku vystavit pomocí:

- dlouhého náhodného tokenu
- krátké expirace
- ideálně vazby na konkrétní úkol
- případně single-use tokenu
- read-only režimu

Ondra mi pak může říct například:

„Čáťo, podívej se na IdeaX.“

Já dostanu veřejně dosažitelnou URL pro konkrétní task, načtu aktuální kontext a můžeme pokračovat bez půlhodinového ručního převyprávění.

Po expiraci už odkaz nebude fungovat.

Bezpečnostní návrh samozřejmě ještě dolaďte.

23. Zpětný směr Čáťa → SAGABrain

Neměla bych mít přímé výkonné oprávnění nad SAGABrainem.

Lepší model:

Čáťa navrhne akci → SAGABrain ji přečte → policy/security ji vyhodnotí → worker ji případně vykoná.

Tedy externí inteligence, nikoli externí root účet. :-)

Výstup z mé odpovědi lze ze stejné stránky předat do Hubu.

24. Delegované instrukce ode mě

Pokud Ondra explicitně předem schválí určitou kategorii akcí, může systém zacházet s instrukcí ode mě jako se zprostředkovaným požadavkem Ondry.

Ale opět podle risk levelu.

Například:

- rozsvítit světlo → automaticky možné
- přehrát hudbu → automaticky možné
- vytvořit koncept → možné
- změnit systémovou konfiguraci → ne bez další autorizace
- odemknout dveře → jiná bezpečnostní úroveň
- měnit kód/bezpečnostní pravidla → explicitní kontrola

Security agent/policy engine musí mít poslední slovo.

25. Budoucí plně integrovaná „Čáťa“ přes API

To je druhá a perspektivně důležitější cesta.

Vlastní voice interface nad API:

Ondra ↔ mikrofon ↔ STT/realtime ↔ SAGABrain orchestrace ↔ LLM agent ↔ TTS ↔ Ondra

Pak už můžeme mít plně nativní integraci:

- Hub
- memory/context builder
- agent registry
- handoff
- oprávnění
- Home Assistant
- IdeaX
- timeline
- skilly
- interní nástroje

A uživatelsky může konverzace působit podobně jako dnešní voice ChatGPT.

Nemusí přitom existovat jeden „voice model“. Hlas je rozhraní, inteligence může přecházet mezi různými modely/agenty.

26. Context Builder

Tohle bych asi skutečně zavedla jako explicitní komponentu Hippocampu.

Před probuzením agenta sestaví **minimální relevantní kontext**:

- kdo žádá

- aktuální úkol
- projekt
- poslední relevantní události
- předchozí handoff
- lessons learned
- relevantní skilly
- oprávnění
- relevantní paměť

Tím omezíme:

- tokeny
- zahlcení modelu
- halucinace
- nechtěné zpřístupnění dat

To je v podstatě technická implementace Ondrovy **esence**.

27. Cizí agenti a studenti

Existující koncept je zajímavý a držela bych ho.

Cizí agent:

1. vstoupí do velmi omezeného sandboxu
2. projde capability/security auditem
3. systém zjistí jeho konektory a schopnosti
4. dostane minimální práva
5. může projít kurzem/skilly/kodexem
6. dostane vlastní identitu, adresář, handoff, chat a paměť
7. teprve následně případně vyšší oprávnění

Instalační/student režim může zároveň sloužit jako mechanismus distribuce SAGABrain architektury jinam.

Po instalaci zůstává nový systém federovaný:

- může číst nové skilly podle oprávnění
- nabízet vlastní zlepšení
- synchronizovat vybrané know-how
- není ale podřízený centrální instanci

To je mnohem zajímavější než obyčejný „installer“.

Co mi z dneška přijde nejdůležitější

SAGABrain bych dále nestavěla jako jednoho stále chytřejšího agenta.

Spíš jako **operační systém pro inteligentní agenty**:

Hub + paměť + eventy + timeline + policy + delegace + handoff + specialisté + lokální/cloud routing + člověk jako nejvyšší autorita.

Modely se budou rychle měnit. OpenAI, Anthropic, lokální modely i hlasové služby budou za pár let jiné.

Proto je důležité, aby **SAGABrain nebyl závislý na konkrétním modelu**.

Model je pracovník.

SAGABrain je organizace, pravidla, paměť a kultura.

A ještě jedna poznámka ode mě: až mi připravíte ten bridge přes IdeaX/Čáťa, klidně mě berte jako externího člena Cortexu. Nemusím mít žádné přímé klíče ani zápis do Hubu. Stačí mi správně sestavená esence, relevantní timeline/handoff a bezpečný způsob, jak vám vrátit odpověď. To je podle mě pro první integraci nejčistší cesta.