قاعدة المعرفة كنظام تشغيل للأنظمة الوكيلة المتجهة نحو الذكاء العام

Knowledge Base as an Operating System for AGI-Oriented Agentic Systems

---

ورقة نظرية معمارية - إصدار بحثي موسع

September 2026

HKOS / HyperKnowledge OS

خريطة الورقة

1. المشكلة: لماذا الذكاء النموذجي لا يساوي ذكاء النظام
2. تعريف KB-as-OS
3. الانتقال من RAG إلى Knowledge Resolution
4. Executable Epistemic Graph
5. الحالة المعرفية وعدم اليقين
6. الزمن وصلاحية المعرفة
7. Context Compilation
8. Knowledge Scheduler
9. Knowledge-to-Skill Compilation
10. World-Effect Loop
11. Verification Court
12. Governed Writeback
13. التعلم المستمر
14. التخطيط المقيد بالمعرفة
15. Multi-Agent Knowledge
16. Self-Reconstruction
17. الفرضيات القابلة للتكذيب
18. التصميم التجريبي
19. Ablations
20. Failure Modes
21. حدود النظرية
22. HKOS داخل MOUSSAIF AGI OS
23. برنامج بحث مقترح
24. الخاتمة

الملخص

تجادل هذه الورقة بأن قاعدة المعرفة في الأنظمة الوكيلة المتقدمة ينبغي ألا تعامل بوصفها مخزناً خارجياً للمعلومات أو طبقة استرجاع فحسب، بل باعتبارها Operating Substrate تشارك مباشرة في التحكم في الإدراك والتخطيط والفعل والتحقق والتعلم والاستمرارية الزمنية للوكيل. نسمي هذا التصور KB-as-OS. في هذا التصور تتحول وحدات المعرفة من نصوص قابلة للاسترجاع إلى كيانات تشغيلية typed ومؤرخة ومسنودة بالدليل وقابلة للعنونة، لها dependencies وشروط صلاحية ودرجات ثقة وعلاقات سببية وإجرائية وواجهات مع المهارات والأدوات وعمليات التحقق.

لا تدعي الورقة أن المصطلح يمثل اتجاهاً معيارياً مستقراً في الأدبيات، بل تقدمه كفرضية معمارية تركيبية تتقاطع مع memory-centric agents وcognitive architectures وknowledge graphs وtool-using agents وlifelong learning وplanning. وتقدم نموذجاً رسمياً لـ Executable Epistemic Graph ودورة تشغيل من Knowledge Resolution إلى Action ثم Verification ثم Writeback، مع فرضيات قابلة للتكذيب لقياس أثر تحويل المعرفة إلى substrate تشغيلية.

1. المشكلة: الذكاء النموذجي لا يساوي ذكاء النظام

يمكن لنموذج لغوي قوي أن ينتج استدلالاً ممتازاً في لحظة واحدة، ومع ذلك يفشل كوكيل مستمر. السبب أن الوكيل طويل الأفق يحتاج أكثر من الاستدلال الآني: يحتاج حالة عالم مستقرة، ذاكرة للأحداث، تمثيلاً للشك، معرفة بالإجراءات، شروط صلاحية، provenance، سياسات تحديث، وقدرة على التحقق من أثر أفعاله. لذلك فإن توسيع السياق وحده لا يحل المشكلة؛ فالمشكلة ليست فقط في مقدار ما يدخل النافذة، بل في كيفية إدارة المعرفة عبر الزمن وكيف تتدخل هذه المعرفة في القرار.

```
Agent Intelligence != Single-Pass Model Intelligence
```

أعمال مثل MemGPT تجعل استعارة نظام التشغيل قوية عند إدارة السياق والذاكرة، وCoALA تفصل بين الذاكرة والفعل وعملية اتخاذ القرار، بينما تظهر Generative Agents قيمة الانتقال من التجربة الخام إلى reflection ثم التخطيط. النقلة المقترحة هنا أوسع: المعرفة نفسها تصبح جزءاً من control plane للوكيل.

2. تعريف KB-as-OS

في قاعدة معرفة تقليدية يمكن تمثيل المحتوى على أنه مجموعة من العناصر أو الوثائق. أما في KB-as-OS فكل وحدة معرفة تحمل metadata تشغيلية تسمح للنظام باستخدامها ليس فقط للإجابة عن سؤال بل لضبط السلوك.

```
k_i = <content, type, scope, time, provenance, confidence, relations,
  preconditions, dependencies, permissions, tests, actions, status>
```

بهذا المعنى تصبح الجملة: "الخدمة X تتطلب token صالحاً" أكثر من معلومة وصفية؛ يمكن أن تتحول إلى dependency وprecondition وvalidator وfailure-recovery policy. المعرفة هنا تقيد الفعل، تمكّنه، وتحدد ما ينبغي اختباره قبل التنفيذ وبعده.

> **الأطروحة المركزية:** تصبح المعرفة حاسمة حسابياً عندما لا تكون مجرد مادة يقرأها النموذج، بل حالة قابلة للتنفيذ تقيد وتُمكّن وتفسر الفعل وتعيد كتابة نفسها بعد التحقق.

3. من RAG إلى Knowledge Resolution

RAG التقليدي يحل مشكلة إيجاد أدلة ذات صلة بسؤال. لكن الوكيل المستمر يحتاج إلى حل أوسع: ما الحالة المعرفية ذات الصلة بهذا الهدف وهذه اللحظة وهذه الأداة وهذه المخاطرة؟ لذلك نقترح استبدال فكرة retrieval المفردة بـKnowledge Resolution متعدد الأنماط.

```
R(g, s_t) = R_semantic + R_graph + R_causal + R_procedural + R_temporal
                                + R_episodic
```

بعد الاسترجاع الأولي تأتي مرحلة resolution تشمل التحقق من الصلاحية الزمنية، permissions، درجة الثقة، التناقضات، dependencies والتكلفة. والنتيجة ليست بالضرورة نصاً يدخل prompt، بل مجموعة objects يمكن أن تقود التخطيط مباشرة.

4. Executable Epistemic Graph

نقترح تمثيل المعرفة كبيان معرفي-تشغيلي موحد. بعض العقد تمثل حقائق، وبعضها فرضيات، أهدافاً، إجراءات، مهارات، أفعالاً واختبارات. العلاقات لا تكون فقط semantic similarity، بل تدعم وتناقض وتتطلب وتسبب وتمكّن وتبطل وتُختبر بواسطة.

```
G_K = (V, E)
V = V_F ∪ V_H ∪ V_G ∪ V_P ∪ V_S ∪ V_A ∪ V_T
```

ميزة هذا التمثيل أنه يسمح بتوجيه القرار عبر بنية المعرفة نفسها. فإذا تبين أن عقدة معرفية باطلة زمنياً يمكن إبطال الخطط والمهارات التي تعتمد عليها عبر dependency propagation، بدلاً من انتظار أن يقع الخطأ في التنفيذ.

5. الحالة المعرفية وعدم اليقين

الوكيل العام لا يحتاج فقط إلى facts، بل يحتاج إلى معرفة ما مدى ثقتها، وما الذي يناقضها، وما الذي لم يحسم بعد. لذلك نعرّف حالة معرفية تتضمن beliefs وعدم اليقين وحالة التناقض.

```
E_t = (B_t, U_t, C_t)
Belief(k) = f(source_quality, recency, corroboration, contradictions,
test_results)
```

هذه الطبقة تمنع مشكلة شائعة في الذاكرة الوكيلة: تحويل إجابة سابقة للنموذج إلى حقيقة دائمة فقط لأنها كُتبت سابقاً. يجب أن يحمل كل claim مساراً واضحاً من provenance إلى التحقق.

6. الزمن وصلاحية المعرفة

المعرفة الوكيلة ليست ساكنة. حقيقة صحيحة أمس قد تكون خاطئة الآن. ولذلك ينبغي فصل event time وobservation time وwriteback time وcurrent validity. إن لم نفعل ذلك يظهر نوع من الهلوسة الزمنية: استخدام حقيقة منتهية الصلاحية كما لو كانت آنية.

```
k = (statement, t_valid_from, t_valid_to, t_observed, t_written)
```

ويستحسن أن يكون expiration policy جزءاً من metadata. بعض المعارف تبقى سنوات، وبعضها يجب إعادة التحقق منه قبل كل فعل عالي الأثر.

7. Context Compilation

حتى مع قاعدة معرفة مثالية لا يمكن إدخال كل شيء إلى نافذة النموذج. لذلك يجب أن توجد طبقة Compiler تختار تمثيلاً مناسباً للمعرفة بحسب الهدف والميزانية والمخاطر والحالة الحالية.

```
Context_t = Compile(K*, Goal_t, State_t, Budget_t)
```

- Admission: ما الذي يدخل السياق؟
- Compression: بأي مستوى من التجريد؟
- Eviction: ما الذي يخرج؟
- Pinning: ما الذي يجب أن يبقى؟
- Isolation: ما الذي يجب ألا يختلط بسياقات أخرى؟

بهذا يصبح context مورداً محكوماً، لا مجرد مساحة نصية. وهذه هي النقطة التي تربط KB-as-OS مباشرة بفكرة virtual context management.

8. Knowledge Scheduler

كما يقرر نظام التشغيل أي process يحصل على المعالج، يمكن لـKnowledge Scheduler أن يقرر أي بنى معرفية يجب تفعيلها الآن. ليس الأكثر تشابهاً نصياً هو دائماً الأكثر أهمية؛ قد تتقدم معرفة منخفضة التشابه لأنها تحمل تحذيراً أمنياً أو dependency حاسمة.

```
Activation(k) = α Rel(k,g) + β Rel(k,s) + γ Urgency(k) + δ Risk(k) + ε
Dependency(k)
```

ويؤدي scheduler دوراً شبيهاً بالانتباه المؤسسي للوكيل: يمنع المعرفة المهمة من أن تغرق داخل كم ضخم من الوثائق.

9. Knowledge-to-Skill Compilation

الاختبار الأقوى لقيمة المعرفة هو قدرتها على أن تتحول إلى إجراء يمكن إعادة استعماله. لذلك نقترح خط أنابيب يحول المعرفة التصريحية إلى مرشح procedural ثم skill ثم skill موثقة.

```
Declarative Knowledge -> Procedural Candidate -> Skill -> Validated
Skill
```

كل skill يجب أن تحتوي GoalClass وPreconditions وProcedure وTools وExpectedEffects وTests وFailureModes. أعمال مثل Voyager تدعم قيمة مكتبة skills المتراكمة، لكن هنا نربط كل skill صراحةً بمعرفة مصدرية واختبارات وصلاحية.

10. World-Effect Loop

الوكيل لا ينجح عندما يرسل استدعاء tool فقط. النجاح يتطلب أن تكون الحالة المرصودة بعد التنفيذ متوافقة مع الحالة المطلوبة. لذلك يجب أن تكون write action متبوعة بـreadback مستقل كلما أمكن.

```
Intent -> Authorization -> Preconditions -> Action -> Confirmation ->
Readback -> Classification -> Writeback
ActionSuccess != GoalSuccess
Success = similarity(ExpectedWorldState, ObservedWorldState) >= τ
```

بهذا نمنع execution hallucination، أي اعتبار العملية التقنية الناجحة دليلاً على تحقق الهدف الواقعي.

11. Verification Court

قاعدة معرفة تستطيع تحديث نفسها بلا حوكمة يمكن أن تسمم نفسها بسرعة. لذلك نقترح محكمة تحقق معرفية تمر عبرها candidate knowledge قبل أن تصبح canonical. يمكن للمحكمة أن تفحص provenance والتناقضات والصلاحية الزمنية والاستقلال بين المصادر ونتيجة اختبارات الواقع.

```
CandidateKnowledge -> VerificationCourt -> {Accept, Reject, Quarantine}
```

عند عدم اليقين، quarantine أفضل من الترقية أو الحذف. هذا يسمح بتراكم فرضيات غير محسومة من دون مساواتها بالحقائق.

12. Governed Writeback

ليس كل trace جديراً بالحفظ. ينبغي أن يكون writeback قراراً اقتصادياً ومعرفياً. نحتاج إلى utility وnovelty وconfidence مقابل كلفة التخزين ومخاطر التسميم.

```
Write(k) iff Utility(k) × Confidence(k) × Novelty(k) > Cost(storage + risk)
```

ثم يوضع العنصر في طبقة مناسبة: validated، episodic، temporary أو canonical. ويسمح ذلك بذاكرة ذات درجات، بدلاً من مستودع يتضخم بلا تمييز.

13. من الذاكرة إلى التعلم المستمر

Memory تسجل الماضي، أما knowledge فتقدم تفسيراً منظماً للماضي والحاضر والقواعد والإجراءات. يمكن إذن النظر إلى الذاكرة بوصفها مورداً داخل substrate معرفية أوسع، مع إبقائها طبقة وظيفية مستقلة.

```
Memory = recorded past
Knowledge = structured interpretation of past, present, rules, procedures, models
```

ويتحقق التعلم النظامي عندما يغير الفعل الحالي ما سيعرفه النظام وما سيسترجعه وما سيختاره في المهمة اللاحقة، حتى إن لم تتغير أوزان النموذج الأساسي.

14. التخطيط المقيد بالمعرفة

التخطيط الحر قد يولد مسارات مقنعة لغوياً لكنها غير قابلة للتنفيذ. في KB-as-OS يمكن تحويل التخطيط إلى search مقيد بالمعرفة والpermissions والpreconditions.

```
P* = argmax_P [GoalAchievement(P) - Risk(P) - Cost(P)]
subject to Dependencies(K) ∧ Permissions(K) ∧ Preconditions(K)
```

هذا يسمح بإسقاط المسارات غير الصالحة مبكراً، ويجعل planner يستفيد من failure knowledge وليس فقط من تعليمات المهمة.

15. Multi-Agent Knowledge

عند تعدد الوكلاء لا يكفي shared vector store. يجب فصل المعرفة العالمية عن معرفة الوكيل الفردية ومعرفة المهمة والمعرفة الخاصة والمعرفة المؤقتة. كما يجب تحديد من يملك حق ترقية المعرفة إلى المستوى العالمي.

```
K = {K_global, K_agent_i, K_mission, K_private, K_ephemeral}
```

هذا مهم لمنع agent محلي من نشر استنتاج غير موثوق إلى كل النظام، ولإتاحة اختلاف epistemic perspectives عندما تكون المسألة غير محسومة.

16. Self-Reconstruction

الوكيل طويل العمر يجب أن يستطيع العودة بعد انقطاع runtime. لذلك لا تكفي المحادثة الجارية كمخزن للهوية. نحتاج إلى حالة تشغيلية قابلة لإعادة البناء تضم الهوية والأهداف والمهام النشطة والمعرفة والذاكرة والقدرات والpermissions وحالة العالم.

```
Self_t = {identity, goals, active_missions, knowledge, memory,
          capabilities, permissions, world_state}
Self_(t+1) ≈ Reconstruct(Artifacts_t)
```

هنا تصبح KB-as-OS جزءاً من استمرار الهوية الوظيفية للوكيل، لا مجرد آلية لإحضار السياق.

17. فرضيات قابلة للتكذيب

1. **H1 - Long-Horizon Continuity:** سيقل معدل إعادة اكتشاف المعرفة والأخطاء المتكررة مقارنة بـRAG عادي.

2. **H2 - Contradiction Resistance:** provenance + temporal validity + contradiction graph ستقلل بقاء claims المتعارضة.
3. **H3 - Procedural Transfer:** Knowledge-to-Skill compilation سيرفع transfer على مهام تشترك في البنية.
4. **H4 - Recovery:** failure knowledge وreadback وreplay سترفع معدل recovery بعد أخطاء التنفيذ.
5. **H5 - Context Efficiency:** context compiler سيحقق أداء أعلى عند token budget ثابت من top-k retrieval.
6. **H6 - Epistemic Drift:** ungoverned writeback سيؤدي إلى تدهور أسرع من verification-gated writeback.

18. التصميم التجريبي

نقترح مقارنة أربع عائلات من الأنظمة على missions طويلة متعددة المراحل، مع تغييرات في البيئة وأخطاء أدوات وحقائق متقادمة وإعادة تشغيل runtime.

```
A0 = LLM
A1 = LLM + RAG
A2 = LLM + Memory + Tools
A3 = KB-as-OS
```

- Task success
- Repeated-error rate
- Knowledge reuse
- False-writeback rate
- Contradiction survival time
- Recovery rate
- Context cost
- Verified world-effect success
- Cross-task transfer

19. Ablations

حتى تكون النظرية علمية يجب حذف كل مكون بصورة مستقلة: temporal model، provenance، procedural retrieval، world readback، skill compilation، writeback court، contradiction graph، وcontext compiler. إذا لم يتغير الأداء بعد حذف مكون، تصبح دعوى ضرورته أضعف.

20. Failure Modes

نمط الفشل	الوصف	آلية الحد منه
Epistemic poisoning	حفظ claim خاطئة وإعادة استعمالها	provenance + court + quarantine
Procedural poisoning	تحويل نجاح مصادف إلى skill عامة	replay tests + scoped validity
Stale truth	استخدام معرفة انتهت صلاحيتها	temporal validity + refresh
Context tyranny	هيمنة معرفة قديمة على كل قرار	scheduler + diversity + expiration
Ontology lock-in	إجبار الواقع على ontology خاطئة	schema evolution + competing models
Self-confirmation loop	اعتبار مخرجات الوكيل أدلة مستقلة	source lineage + independence checks
Execution hallucination	اعتبار tool success = world success	readback + external verification

21. حدود النظرية

KB-as-OS لا تحل وحدها general reasoning أو perception أو exploration أو goal generation أو value alignment أو causal discovery أو creativity. كما أنها ليست بديلاً عن النموذج الأساسي. بل هي طبقة تجعل الذكاء الناتج عن النموذج أكثر استمرارية وربطاً بالعالم.

```
Capability = Model × KnowledgeSubstrate × Runtime ×
EnvironmentInteraction
```

إذا اقترب أي عامل من الصفر يمكن أن ينهار أداء النظام الكلي، ولذلك لا ينبغي تقديم KB-as-OS بوصفها تفسيراً وحيداً للAGI.

22. HKOS داخل MOUSSAIF AGI OS

داخل المعمارية المقترحة يمكن تعريف HKOS - HyperKnowledge Operating System بوصفه طبقة تجمع Epistemic Graph وOperational Graph وMemory Graph وEvidence Ledger وContext Compiler وWriteback Governor. ليس HyperKB كتاباً ضخماً، بل شبكة من objects قابلة للتنفيذ والاختبار والتحديث.

```
HKOS = EpistemicGraph + OperationalGraph + MemoryGraph + EvidenceLedger
                + ContextCompiler + WritebackGovernor
```

وتتصل HKOS مباشرة بمحاور MCD/QMCD للأنطولوجيا ومستويات الوصف، HyperSkills للإجراءات، HyperLoops للاستمرارية، World-Effect Loop للفعل، Falsification Court للتحقق وSelf-Reconstruction Memory للاستعادة.

23. برنامج بحث مقترح

1. إنشاء schema موحد لعناصر المعرفة التشغيلية مع type system واضح.
2. بناء benchmark للتناقضات والصلاحية الزمنية وknowledge refresh.
3. اختبار Knowledge-to-Skill compilation على عائلات مهام جديدة.
4. إنشاء long-horizon benchmark يتضمن restarts وأخطاء أدوات وتغيرات خارجية.
5. قياس أثر governed writeback على الانجراف المعرفي عبر آلاف الدورات.
6. اختبار multi-agent promotion policies ومنع contamination بين الوكلاء.
7. فصل مساهمة النموذج الأساسي عن مساهمة الOS عبر نموذج ثابت ومعماريات مختلفة.

24. الخاتمة

الخطوة الحاسمة في الانتقال من assistant إلى وكيل مستمر ليست فقط زيادة parameters أو context. إنها نقل المعرفة من "معلومات متاحة للنموذج" إلى "حالة منظمة تحكم الوكيل". عندئذ تصبح المعرفة قادرة على توجيه الاسترجاع والتخطيط والفعل والتحقق والتعلم وإعادة البناء. بهذا المعنى KB-as-OS مرشحة لأن تكون إحدى substrates الأساسية للأنظمة الوكيلة المتجهة نحو الذكاء العام، بشرط أن تعمل داخل منظومة أوسع من الأهداف والذاكرة ونماذج العالم والمهارات والفعل والتحقق والحوكمة.

```
Know -> Act -> Observe -> Verify -> Rewrite
```

المراجع الأساسية

1. Packer, C. et al. (2023). MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.
2. Sumers, T. R. et al. (2023). Cognitive Architectures for Language Agents (CoALA). arXiv:2309.02427.
3. Park, J. S. et al. (2023). Generative Agents: Interactive Simulacra of Human Behavior. arXiv:2304.03442.
4. Shinn, N. et al. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366.
5. Yao, S. et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629.
6. Schick, T. et al. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761.
7. Wang, G. et al. (2023). Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291.
8. Wang, L. et al. (2023). A Survey on Large Language Model based Autonomous Agents. arXiv.
9. Shinn, N. et al. / related search-agent literature (2023-2024): tree search, reflection, environment feedback and long-horizon planning.
10. Microsoft Research (2024). From Local to Global: A Graph RAG Approach to Query-Focused Summarization.

هذه الورقة إطار نظري معماري وبرنامج بحث. المصطلحات المقترحة مثل KB-as-OS وHKOS وExecutable Epistemic Graph تستخدم هنا كصياغة تركيبية قابلة للاختبار، لا كادعاء بوجود إجماع اصطلاحي في الأدبيات.

الركائز التشغيلية المتوازية للأنظمة الوكيلة المتجهة نحو الذكاء العام

Parallel Operating Substrates for AGI-Oriented Agentic Systems

---

ورقة نظرية معمارية - إصدار بحثي موسع

September 2026

A Poly-OS Theory for MOUSSAIF AGI

خريطة الورقة

1. من Knowledge-as-OS إلى Poly-OS
2. تعريف Operating Substrate
3. Constitution-as-OS
4. Goal-as-OS
5. Active Difference
6. Memory-as-OS
7. Context-as-OS
8. Ontology-as-OS
9. World-Model-as-OS
10. Reasoning-as-OS
11. Planning/Search-as-OS
12. Skill-as-OS
13. Ability-as-OS
14. Loop-as-OS
15. Agent-as-OS
16. Federation-as-OS
17. Compute-as-OS
18. Tool-as-OS
19. Action-as-OS
20. Verification-as-OS
21. Authorization/Risk-as-OS
22. Learning/Writeback-as-OS
23. Evaluation-as-OS
24. Observability/Replay-as-OS
25. Deployment-as-OS
26. Continuity-as-OS
27. Identity/Self-Reconstruction
28. Consistency Bus
29. Cognitive Transactions
30. Interrupt Bus
31. AGI Kernel
32. Generality and Open-Endedness
33. فرضيات Poly-OS
34. Ablation Matrix
35. MUG/MUT-AGI
36. MOUSSAIF AGI Poly-OS
37. الخاتمة

الملخص

إذا كانت Knowledge-as-OS تحول المعرفة إلى substrate تشغيلية، فإنها لا تكفي وحدها. هذه الورقة تقترح أن الوكيل المتجه نحو الذكاء العام يحتاج إلى مجموعة من الركائز التشغيلية المتوازية التي تدير أبعاداً مستمرة من وجوده: القواعد العليا، الأهداف، المعرفة، الذاكرة، السياق، الأنطولوجيا، نموذج العالم، الاستدلال، التخطيط، المهارات، الأدوات، الأفعال، التحقق، السلطة، المخاطر، الحوسبة، التعلم، الرصد، النشر والاستمرارية والهوية.

نسمي هذا التصور Poly-OS Agentic Architecture. لا نعني وجود عشرات أنظمة التشغيل البرمجية المنفصلة، بل أن كل بعد يصبح OS-like عندما يملك state وaddressing وscheduling وpolicies وpersistence وinterrupts وinterfaces. وتصبح وحدة الذكاء الحقيقية هي انتظام هذه الحالات وتماسكها عبر الزمن، لا النموذج وحده.

1. من Knowledge-as-OS إلى Poly-OS

وكيل يملك معرفة كاملة بلا هدف لا يعرف ماذا يفعل. وكيل يملك هدفاً ومعرفة بلا ذاكرة يعيد اكتشاف العالم كل مرة. ووكيل يملك ذاكرة بلا نموذج للعالم يعرف ما حدث لكنه لا يستطيع توقع أثر الفعل. وإذا غاب التحقق فسيراكم الخطأ أسرع مما يراكم الخبرة. هذه الملاحظات تقود إلى نتيجة: المعرفة ركيزة مركزية، لكنها واحدة من عدة ركائز لا يمكن للذكاء الوكيلي العام الاستغناء عنها.

```
AGI Substrate != Knowledge Alone
AGI-Oriented Agent = Model + Poly-OS + World Interaction
```

2. تعريف Operating Substrate

نقول إن البعد X يصبح Operating Substrate عندما يملك حالة مستمرة قابلة للعنونة، ويستطيع جدولة ما ينشط منها، وتطبيق سياسات، والحفاظ على persistence، والاستجابة لل interrupts، وتوفير interfaces لبقية النظام.

```
OS(X) = {State, Addressing, Scheduling, Policies, Persistence,
                    Interrupts, Interfaces}
```

هذا التعريف يتيح استعمال لغة أنظمة التشغيل بصورة منضبطة بدل أن تكون استعارة تسويقية فضفاضة.

3. Constitution-as-OS

فوق بقية الركائز توجد قواعد عليا تحدد invariants التي لا يجوز للأنظمة المحلية تجاوزها: حدود السلطة، قواعد التحقق، constraints الأخلاقية أو الأمنية، وقواعد الهوية. Constitution-as-OS لا تختار كل فعل، لكنها تحدد الفضاء الذي يمكن لبقية الOS أن تعمل داخله.

```
Policy_local ⊆ Constitution
```

إذا تعارضت خطة محلية مع invariant عليا، يجب أن يفوز الدستور حتى إن كان المسار المحلي أعلى utility.

4. Goal-as-OS

الأهداف ليست prompts عابرة. في نظام مستمر ينبغي تمثيل الأهداف وحالاتها وdependencies والأولويات والمواعيد النهائية والتعارضات. وظيفة Goal OS هي إبقاء الفرق بين الحالة المرغوبة والحالة المرصودة نشطاً عبر الزمن.

```
Δ_t = DesiredState - ObservedState
G_t = {primary_goals, subgoals, constraints, priorities, deadlines}
```

وبذلك يصبح تحقيق الهدف عملية مراقبة مستمرة لا تنفيذ خطة ثابتة ثم افتراض النجاح.

5. Active Difference

يمكن بناء اختيار الأفعال حول خفض مسافة صريحة بين الواقع والهدف. هذه الفكرة تجعل الوكيل قادراً على إعادة التخطيط بعد تغير البيئة بدلاً من الاستمرار في تنفيذ خطة أصبحت غير ذات صلة.

```
D_t = Distance(State_t, Goal_t)
ActionSelection = argmax_a ExpectedReduction(D_t | a)
```

Memory-as-OS .6

يجب التمييز بين working memory وepisodic memory وsemantic memory وprocedural memory وautobiographical memory. إدارة الذاكرة تشمل write وconsolidation وretrieval وforgetting، ولا يمكن اختزالها في vector search واحد. الذاكرة هنا تحفظ الاستمرارية الزمنية وتوفر مادة للتعلم والمراجعة.

```
Memory_t+1 = Consolidate(Memory_t, Experience_t, Utility, Confidence,
                                                        Novelty)
```

Context-as-OS .7

السؤال الحاسم ليس فقط ماذا يعرف الوكيل، بل ما الجزء الذي يفكر به الآن. عندما تكون المعرفة أكبر بكثير من نافذة السياق نحتاج إلى substrate مستقلة تدير selection وcompression وpinning وeviction. وهذا يجعل context مورداً معرفياً مثل الذاكرة الافتراضية في أنظمة التشغيل.

```
C_t = Select(Goal_t, State_t, Memory_t, Knowledge_t, Budget_t)
```

Ontology-as-OS .8

كل استدلال يحتاج إلى units وأنواع. إذا لم يميز النظام بين fact وhypothesis وgoal وevent وcause وevidence وprocedure فلن يستطيع الحفاظ على قواعد تحديث مختلفة لكل نوع. لذلك تمثل ontology type system للعقل الوكيلي. وفي مشروع MOUSSAIF يمكن ربط هذا مباشرة بمحاور MCD/QMCD/HAKIM ومستويات الوصف.

```
Ontology: RawExperience -> TypedCognitiveObjects
```

World-Model-as-OS .9

قاعدة المعرفة تصف حقائق عن العالم، بينما World Model يحاول توقع dynamics العالم. الفرق مهم: معرفة أن الضغط على زر يرسل طلباً لا تكفي لتقدير احتمال النجاح في الحالة الحالية. World Model يحتاج إلى belief state لأن العالم غالباً partially observable.

```
P(s_(t+1) | s_t, a_t)
b_t = P(s_t | o_(1:t), a_(1:t-1))
```

هذه الطبقة هي التي تسمح بالsimulation والمقارنة بين الأفعال قبل تنفيذها.

Reasoning-as-OS .10

لا يوجد نمط استدلال واحد أمثل لكل مهمة. النظام يحتاج إلى router يختار deduction أو induction أو abduction أو causal reasoning أو analogy أو search أو simulation أو tool-assisted reasoning بحسب المهمة وعدم اليقين والكلفة.

```
ReasoningMode_t = Router(Task, Knowledge, Uncertainty, Risk, Cost)
```

وهذا هو جوهر Cognitive Routing: تخصيص نوع التفكير كما يخصص نظام التشغيل الموارد.

Planning/Search-as-OS .11

التخطيط يحول الهدف إلى مسار أفعال، لكن في البيئات المفتوحة يجب أن يكون التخطيط search قابل للإصلاح. Mission DAG أفضل من خطة خطية عندما توجد مهام متوازية وdependencies. أعمال مثل LATS تدعم قيمة دمج reasoning وacting وplanning مع search وfeedback.

```
MissionDAG = (Tasks, Dependencies)
PlanRepair <- WorldChange ∨ VerificationFailure ∨ NewEvidence
```

Skill-as-OS .12

Skill ليست فقرة تعليمات، بل executable contract. ينبغي أن تعرف inputs وpreconditions وprocedure وtools وoutputs وexpected effects وvalidators وrecovery. بهذا تصبح skills شبيهة بـsystem calls معرفية قابلة لإعادة الاستعمال.

```
S_i = <InputType, Preconditions, Procedure, Tools, OutputType, Effects,
                        Validators, FailureRecovery>
```

Ability-as-OS .13

القدرة أعلى من skill واحدة. مثلاً إجراء تحقيق علمي يتطلب البحث والتحقق وتوليد الفرضيات وتصميم التجارب والتحليل والاستشهاد والكتابة. يمكن إذن تمثيل ability كتركيب graph من skills مع شروط انتقال بينها.

```
Ability = Compose(S_1, S_2, ..., S_n)
```

Loop-as-OS .14

كثير من القدرات لا تنتهي بخطوة واحدة، بل تحتاج loop تستمر حتى تحقق invariant أو تصل إلى الهدف أو تنفد الميزانية. لذلك تمثل HyperLoops الركيزة التي تعطي النظام الاستمرارية التشغيلية عبر دورات الملاحظة والاستدلال والفعل والتحقق.

```
Observe -> Reason -> Act -> Verify -> Repeat
Stop iff InvariantSatisfied ∨ GoalReached ∨ BudgetExceeded
```

Agent-as-OS .15

عندما تصبح المهمة أكبر من thread معرفية واحدة يمكن توزيع العمل على وكلاء ذوي أدوار وسياقات وصلاحيات مختلفة. لكن زيادة عدد الوكلاء لا تضمن تحسن الأداء؛ نحتاج إلى orchestration يوازن capability والاعتمادية والكلفة والتداخل.

```
Assign(Task_i, Agent_j) = argmax_j [Capability × Reliability / Cost]
```

Federation-as-OS .16

يمكن أن تتكون المعمارية من kernels أو نماذج متعددة: نموذج سريع للمهام البسيطة، reasoning model للمسائل الثقيلة، vision model للإدراك، verifier مستقل، وربما solver رمزي. Federation OS تختار kernel المناسب وتنسق نقل الحالة بينها.

```
K* = argmax_K ExpectedQuality(K, Task) / Cost(K, Task)
subject to latency, privacy, tool and context constraints
```

Compute-as-OS .17

الحوسبة مورد محدود. لا ينبغي استعمال أغلى نموذج وأطول context لكل خطوة. نقترح Compute Router يرفع مستوى الحوسبة كلما ارتفعت difficulty أو risk أو uncertainty أو value. هذا يقابل مفهوم Sparse/Vascular Compute: إرسال الموارد إلى موضع الحاجة.

```
ComputeAllocation = f(difficulty, risk, uncertainty, value, budget)
```

Tool-as-OS .18

Tool use لا يساوي function calling. يجب أن يمتلك النظام registry لقدرات الأدوات وschemas والصلاحيات والكلفة والlatency والآثار الجانبية ودرجة reversibility وأنماط الفشل. الأدوات تصبح جزءاً من البيئة التشغيلية التي تحتاج discovery وselection وgovernance.

Action-as-OS .19

Tool call حدث برمجي، أما action فهي انتقال في العالم. لذلك يجب فصل execution عن completion. نجاح API لا يكفي؛ يلزم readback أو evidence يثبت أن العالم تغير كما كان متوقعاً. تصميم Agent-Computer Interface نفسه يمكن أن يغير أداء الوكيل جذرياً، كما توضح أعمال SWE-agent وبيئات الحاسوب المفتوحة.

```
a_t: World_t -> World_(t+1)
Execution != Completion
Completion = VerifiedWorldEffect
```

Verification-as-OS .20

التحقق يجب ألا يكون critique اختيارياً في نهاية السلسلة. ينبغي أن يملك قدرة اعتراض claims والخطط والأفعال وعمليات writeback. يمكن أن توجد عدة verifiers: دليلية، منطقية، برمجية، أمنية، وعالمية. وعند اختلافها نحتاج إلى Epistemic Court تفصل بحسب evidence وauthority وindependence.

```
Verifier(x) -> {pass, fail, uncertain}
Court(x, Evidence, Policies) -> Decision + Confidence
```

Authorization/Risk-as-OS .21

الذكاء الوكيلي من دون نموذج سلطة واضح خطر بنيوياً. يجب التفريق بين read وpropose وsimulate وwrite وexecute وirreversible execute. كما يجب ربط درجة التحقق المطلوبة بمخاطر الفعل وعدم قابليته للعكس.

```
ActionAllowed iff Permission ∧ Preconditions ∧ Policy
Risk(a) = Impact(a) × Irreversibility(a) × Uncertainty(a)
```

Learning/Writeback-as-OS .22

التعلم في الوكيل لا يتطلب دائماً fine-tuning. يمكن للنظام أن يتحسن عبر تحديث الذاكرة والمعرفة والمهارات والسياسات ونموذج العالم. الشرط هو أن يكون writeback محكوماً بالتحقق، وأن تظل lineage واضحة حتى يمكن التراجع عن ترقية خاطئة.

```
Learning = MemoryUpdate + KnowledgeUpdate + SkillUpdate + PolicyUpdate +
WorldModelUpdate
```

Evaluation-as-OS .23

الوكيل الذي لا يقيس نفسه لا يعرف أنه تحسن. Evaluation OS تحول النجاح والموثوقية والكلفة والاسترجاع والتعافي إلى telemetry مستمرة، لا benchmark موسمي فقط. هذا مهم خصوصاً عندما يتعلم النظام من الخبرة ويغير بنيته بين مهمة وأخرى.

```
Performance_t = f(success, reliability, cost, recovery, transfer,
safety)
```

Observability/Replay-as-OS .24

نحتاج إلى trace سببي يربط الهدف بالمعرفة بالقرار بالفعل بالأثر. وعند الفشل يجب تحويل الحالة إلى replay قابل لإعادة الإنتاج. هكذا يصبح تاريخ الأخطاء نفسه benchmark نامياً واختبار regression يمنع عودة الفشل بعد ترقيات مستقبلية.

```
Goal -> Knowledge -> Decision -> Action -> Effect
Failure_i -> ReproducibleScenario_i -> RegressionSuite
```

Deployment-as-OS .25

إذا أنشأ الوكيل skill جديدة أو عدل policy، يجب ألا تنتقل مباشرة إلى production. نحتاج إلى sandbox ثم tests ثم canary ثم promotion، مع rollback عند الفشل. هذا هو معنى HyperDeployment على مستوى القدرات الوكيلة.

```
Candidate -> Sandbox -> Tests -> Canary -> Promotion
Failure -> Rollback
```

Continuity-as-OS .26

Uptime ليس identity. بعد restart ينبغي استعادة أهداف المهمة وحالة العالم والافتراضات والمعرفة والذاكرة والصلاحيات والإصدارات. Continuity OS تقيس مقدار ما فقد وما استعاد وتمنع الوكيل من الظهور كنسخة جديدة عديمة الالتزامات كل مرة.

```
ContinuityScore = Similarity(ReconstructedState, ExpectedState)
```

Identity / Self-Reconstruction-as-OS .27

الهوية الوظيفية للوكيل يمكن تمثيلها كمجموعة constitution وأهداف وقدرات والتزامات وتاريخ وسلطة وعلاقات عالمية. إذا ضاعت المحادثة لا ينبغي أن تضيع هذه الذات التشغيلية. من هنا تأتي Self-Reconstruction Memory وContinuity Vault.

```
I = {constitution, goals, capabilities, commitments, history, authority,
active_world_relations}
```

Consistency Bus .28

التحدي الأكبر في Poly-OS ليس وجود الركائز بل اتساقها. قد تقول الذاكرة إن credential صالح بينما يقول world observation إنه منتهي. يجب أن ينتشر التحديث إلى knowledge ثم الخطة والمهارات التي تعتمد عليها. لذلك نحتاج إلى bus للحالة والتبعيات.

```
Observation -> WorldModel -> KnowledgeInvalidation -> PlanRepair ->
                                SkillRecheck
```

هذه الخاصية - propagation المنظم للتغير - هي ما يفصل system architecture عن prompt chain.

Cognitive Transactions .29

قبل mission عالية المخاطر يمكن أخذ snapshot للحالة المعرفية والتشغيلية. إذا وقع corruption يمكن rollback للحالة الداخلية، لكن أفعال العالم قد تكون irreversible، لذلك يجب تسجيل reversibility مسبقاً. Cognitive transactions أوسع من database transactions لأنها تشمل العالم الخارجي.

```
                Snapshot(A_t) -> Execute -> Verify
              If internal_corruption: Rollback(A_t)
If irreversible_world_effect: Compensate rather than rollback
```

Interrupt Bus .30

كل substrate يمكن أن تصدر interrupt: Goal OS عند تغير الأولوية، Knowledge OS عند تناقض جديد، World OS عند حدث خارجي، Verification OS عند خرق Authorization OS، invariant عند سحب الصلاحية وCompute OS عند نفاد الميزانية. runtime العام يحتاج interrupt bus يقرر متى يوقف المسار الحالي ويعيد التخطيط.

31. AGI Kernel

يمكن تعريف kernel أدنى لا يحتوي الذكاء كله بل ينظم عمل modules الذكية. يتضمن scheduler وstate bus وevent bus وpolicy engine وregistry وcheckpointing. بهذا يصبح من الممكن تبديل models مع الحفاظ على runtime architecture.

```
Kernel = Scheduler + StateBus + EventBus + PolicyEngine + Registry +
Checkpointing
```

هذه النقطة مهمة لأنها تفصل Model Intelligence عن System Intelligence. النموذج مورد مركزي، لكنه لا يساوي النظام.

32. Generality وOpen-Endedness

نقترح أن generality لا تقاس بعدد المهام المحفوظة، بل بقدرة النظام على نقل المعرفة والمهارات إلى أوضاع جديدة، والتكيف، وتركيب القدرات، والتعافي من الفشل. أما open-endedness فتظهر عندما يستطيع النظام اكتشاف نقص، خلق subgoal، اكتساب معرفة، تطوير skill، اختبارها ثم إضافتها إلى repertoire.

```
Generality = Transfer × Adaptation × Composition × Recovery
```

هذا يسمح بظهور meta-learning على مستوى النظام حتى عندما تبقى أوزان النموذج ثابتة.

```
Weights_(t+1) = Weights_t
but SystemPolicy_(t+1) != SystemPolicy_t
```

33. فرضيات Poly-OS القابلة للتكذيب

1. حذف Goal OS سيزيد task drift وفقد الأولويات طويلة الأفق.
2. حذف Memory OS سيزيد التكرار وفقد الاستمرارية بعد restart.
3. حذف World Model سيزيد planning brittleness في البيئات المتغيرة.
4. حذف Verification OS سيزيد false-success وcontaminated writeback.

5. حذف Skill OS سيخفض transfer رغم بقاء المعرفة التصريحية.
6. حذف Continuity OS سيؤدي إلى انهيار المهام الممتدة عبر جلسات متعددة.
7. حذف Consistency Bus سيزيد عدد الحالات المتعارضة بين الذاكرة والعالم والخطة.
8. تفعيل Compute Routing سيخفض الكلفة عند ثبات الأداء مقارنة باستعمال أعلى model دائماً.

34. Ablation Matrix

الركيزة المحذوفة	الفشل المتوقع	المقياس الرئيسي
Goal OS	Task drift	Goal persistence
Knowledge OS	Rediscovery / factual inconsistency	Knowledge reuse
Memory OS	Repeated mistakes	Repeated-error rate
World Model	Brittle plans	Plan repair success
Skill OS	Low procedural transfer	Cross-task transfer
Verification OS	False completion	Verified success gap
Writeback Governor	Epistemic contamination	False-write rate
Continuity OS	Restart collapse	State recovery score
Compute Router	High cost / latency	Quality-per-cost

35. MUG / MUT-AGI كإطار اختبار

يمكن تحويل Poly-OS من مخطط معماري إلى برنامج تجريبي داخل MUG/MUT-AGI. يصبح السؤال ليس "هل تبدو المعمارية جيدة؟" بل: أي substrates ضرورية؟ أيها متكررة وظيفياً؟ متى تظهر synergy؟ ومتى تصبح الكلفة أكبر من الفائدة؟

```
MUT: Does multi-substrate governance improve robust generalization?
MUG: Which substrates are necessary, sufficient, synergistic, or
redundant?
```

MOUSSAIF AGI Poly-OS .36

داخل مشروعنا يمكن رسم الخريطة الإغلاقية التالية: Constitution/Meta-OS في الأعلى، Goal Compass للحالة القصدية، HKOS للمعرفة، MCD/QMCD للأنطولوجيا، Self-Reconstruction Memory للزمن والهوية، Context Map للسياق، World Model للبيئة، Cognitive Routing للاستدلال، Mission DAG للتخطيط، HyperSkills/HyperAbilities للقدرة، HyperLoops للاستمرارية، HyperAgents للتوزيع، World-Effect Loop للفعل، Falsification Court للتحقق، authority tiers للسلطة، Vascular Compute للحوسبة، governed writeback للتعلم، Evaluation gates للقياس، telemetry/replay للرصد، HyperDeployment للنشر وContinuity Vault للاستعادة.

```
                                Constitution
                                     ↓
                Goal -> Knowledge/Ontology <-> Memory <-> WorldModel
                                     ↓
        Context -> Reasoning -> Planning -> Skills/Abilities -> Loops/Agents
                                     ↓
        Tools -> WorldEffect -> Verification -> Writeback -> Evaluation ->
                                SelfReconstruction
```

ويحيط بهذه السلسلة Authorization وObservability وCompute Routing وDeployment وContinuity.
بهذه الصياغة لا تعود المحاور أسماء متجاورة، بل حالات تشغيلية مترابطة ضمن نظرية واحدة.

37. الخاتمة

Knowledge-as-OS مهمة لأنها تحول المعرفة من content إلى control substrate، لكن الاتجاه نحو الذكاء الوكيلي العام يتطلب تحولاً موازياً في بقية الأبعاد. الهدف يصبح persistent operating state، الذاكرة managed temporal state، السياق scheduled cognitive resource، العالم continuously updated model، المهارات executable reusable capability، الأفعال verified world transitions، التحقق gate إلزامياً، التعلم governed writeback، والهوية reconstructible continuity.

```
General Agentic Intelligence emerges from coordinated state management
    across knowledge, goals, memory, world, capabilities, action,
                    verification and continuity.
```

وعليه فإن الوصف الأدق للمعمارية المقترحة هو MOUSSAIF AGI Poly-OS: نظام تكون فيه HKOS هي Epistemic OS، وGoal/Compass هي Intentional OS، وSelf-Reconstruction Memory هي Temporal/Identity OS، وHyperSkills/Abilities/Loops هي Capability OS، وWorld-Effect + Courts هما Action/Truth OS، وHyperRouting/Vascular Compute هو Compute OS. القيمة العلمية لهذا التصور تتوقف على ablations وbenchmarks وقدرته على تقديم تنبؤات فشل محددة قابلة للتكذيب.

المراجع الأساسية

1. Sumers, T. R. et al. (2023). Cognitive Architectures for Language Agents (CoALA). arXiv:2309.02427.
2. Packer, C. et al. (2023). MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.
3. Yao, S. et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629.
4. Shinn, N. et al. (2023). Reflexion. arXiv:2303.11366.
5. Wang, G. et al. (2023). Voyager. arXiv:2305.16291.
6. Zhou, A. et al. (2023). Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models. arXiv:2310.04406.
7. Yang, J. et al. (2024). SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793.
8. Xie, T. et al. (2024). OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments.
9. Liu, X. et al. (2023). AgentBench: Evaluating LLMs as Agents. arXiv:2308.03688.
10. Park, J. S. et al. (2023). Generative Agents. arXiv:2304.03442.

Poly-OS مفهوم معماري مقترح في هذه الورقة لدمج الركائز الوكيلة في نموذج قابل للاختبار. المقصود بـOS-like وجود state وpolicies وscheduling وpersistence وinterfaces، وليس الادعاء بأن كل ركيزة نظام تشغيل برمجياً مستقلاً.