

Context engineering ומערכות זיכרון ל-agents: מ-CLAUDE.md ועד dreaming

איך Anthropic חושבת על זיכרון של agents בפרודקשן: קבצים, guardrails ותהליך רקע
שמתחזק את הזיכרון מחוץ ל-session

AI DevCon · Lamis (Anthropic, Applied AI) (כנס של Tessel), דרך repost ב-X של החשבון ultima · תאריך
ההרצאה המקורית לא ידוע

משך: כ-30 דקות · X

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה של כחצי שעה מכנס AI DevCon (לפי הלוגו בשקפים הכנס מאורגן על ידי Tessel). המרצה היא, Lamis, member of technical staff בצוות Anthropic של Applied AI. הצוות הזה יושב בין מחקר, מוצר ו-go-to-market, והיא עצמה עובדת עם סטארטאפים ומייסדים שבונים agents. הנושא הוא context engineering, ובעיקר זיכרון: איך לוקחים מודל חכם וגורמים לו להשתפר עם הזמן בתוך ארגון מסוים.

הטענה המרכזית: אינטליגנציה של מודל לא מצטברת מעצמה. מה שמצטבר הוא context מתוחזק. היא עוברת על מה שעבד בשנה האחרונה (CLAUDE.md, memory tool, skills, ובסוף זיכרון שבנוי כ-file system), מציגה ארבעה עקרונות לפרודקשן (versioning, concurrency, permissioning, portability), ומסבירה למה זיכרון in-band, כלומר בתוך ה-session, נתקע. הפתרון שהיא מציעה הוא תהליך out-of-band בשם dreaming. חשוב לדעת: הכיתוב של ה-repost ב- («we don't write prompts anymore, we build loops») X לא מופיע בתמליל. ההרצאה עוסקת בזיכרון, לא בלולאות prompting.

איך לקרוא את המסמך

- 1. **תקציר מנהלים:** הטענות המרכזיות עם חותמות זמן, ומה לא נאמר.
- 2. **למה אינטליגנציה לבד לא מצטברת:** הבעיה שההרצאה באה לפתור.
- 3. **האבולוציה של זיכרון agents:** מ-CLAUDE.md, דרך memory tool ו-skills, ועד זיכרון כ-file system.
- 4. **מתיאוריה לפרודקשן:** ארבעה עקרונות הנדסיים ל-multi-agent.
- 5. **גבולות ה-in-band ו-dreaming:** הארכיטקטורה של תהליך הרקע שמתחזק את הזיכרון.
- 6. **שאלות מהקהל:** Claude Managed Agents, הרשאות ב-dreaming, ו«האם אנחנו ממציאים מחדש את בסיס הנתונים».
- 7. **מה לעשות עכשיו:** צעדים מעשיים וטבלת שאלות פתוחות.
- 8. **מילון מונחים:** המונחים באנגלית שמופיעים במסמך.

הערות מקור המסמך הופק אוטומטית מהתמליל ומהשקפים של ההקלטה. השמות והמונחים עברו דרך זיהוי דיבור (ASR), ולכן ייתכנו שיבושים. שם המשפחה של המרצה לא קריא בשקפים, והשם Lamis לקוח מהתמליל. הציטוטים מועתקים מילה במילה ונבדקו מול חותמת הזמן שלהם. לא נוסף שום מידע ממקורות חיצוניים.

1. תקציר מנהלים

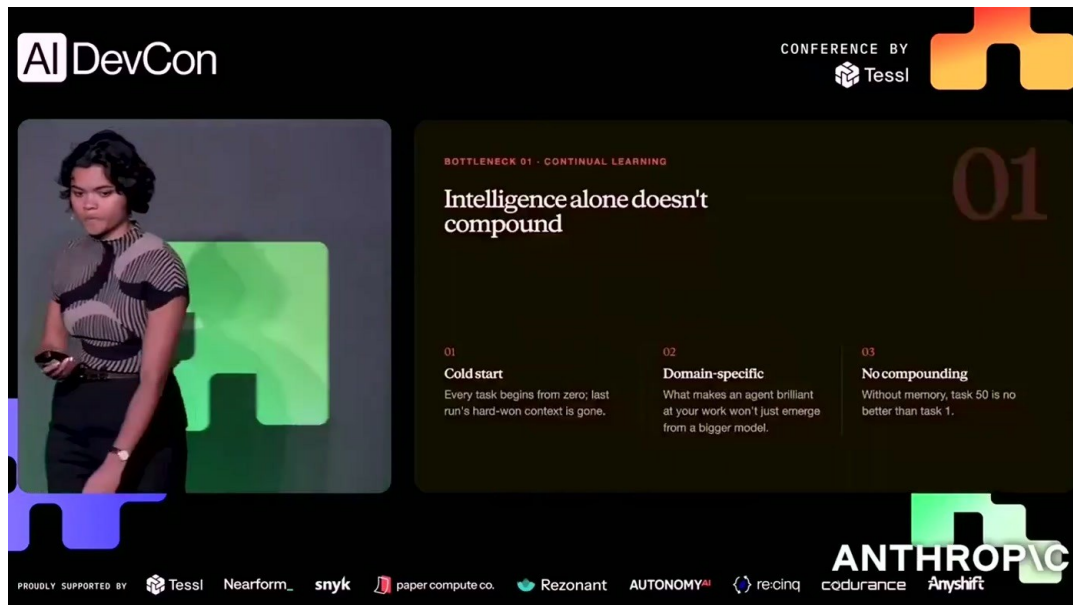
ההרצאה מתארת מסלול: קודם קובץ markdown פשוט, אחר כך זיכרון שה-agent מנהל בעצמו, ובסוף תהליך אסינכרוני שעובר על transcripts של כל ה-fleet ומציע תיקונים לזיכרון. הנקודה היא שכל שלב פתר את המגבלה של השלב הקודם, ושברודקשן עם הרבה agents צריך להחזיר עקרונות קלאסיים של הנדסת תוכנה.

- **Context הוא מכפיל של אינטליגנציה:** מודל חדש לא יודע מה נדרש כדי להצליח בארגון שלך. לכן השקעה ב-context מכפילה את האינטליגנציה גם כשהמודלים משתפרים [1:30].
- **CLAUDE.md עבד "unreasonably effective":** קובץ markdown שמזרק בתחילת ה-session מכון את ה-agent היטב. הבעיה שלו היא context bloat כשהקובץ גדל [3:01].
- **Skills פותרים את הגדילה בעזרת progressive disclosure:** ה-agent קורא רק את ה-front matter וטוען את גוף הקובץ רק כשצריך. המגבלה: עדיין בני אדם מחליטים על מה לכתוב [4:37].
- **ה-state of the art הוא זיכרון כ-file system:** קבצי markdown, כלים רגילים כמו Bash ו-grep, ואוטונומיה מלאה לכתובה. אין צורך בכלי זיכרון ייעודיים [6:09].
- **בפרודקשן צריך ארבעה guardrails:** versioning עם rollback ותיעוד של מי שינה ועל סמך איזה transcript, concurrency מבוסס hash, permissioning בשכבות ו-portability דרך API נקי [9:15].
- **זיכרון in-band נתקע משני טעמים:** ה-agent מתחלק בין המשימה לבין טיפוח הזיכרון, ואין לו נראות לדפוסים שחוזרים בין sessions ובין agents [13:55].
- **Dreaming הוא תהליך batch אסינכרוני:** orchestrator מפעיל sub-agents על transcripts כולל tool calls, ומציע שינויים לזיכרון עם דוגמאות וסטטיסטיקת שכיחות. האדם מאשר או דוחה [20:04].
- **העלות מחזירה את עצמה, לפי הטענה:** agents עם זיכרון טוב עושים one-shot ביתר קלות ושורפים פחות tokens. לא הוצגו מספרים [23:11].
- **זה זמין כמוצר:** בשאלות מהקהל היא אומרת ש-versioning, hashing, ו-dreaming זמינים ב-memory and API dreaming של Claude Managed Agents [26:13].

מה לא נאמר

- **אין מספרים:** השיפור בדיוק, בעלות ובזמן תגובה מוצג באופן איכותי בלבד, בלי benchmark, בלי אחוזים ובלי עלות של ריצת dreaming.
- **אין סף שכיחות:** לא נאמר מה נחשב «prevalent enough» כדי שה-orchestrator יציע שינוי, ולא איך מונעים מ-dreaming להכניס טעויות משלו.
- **Prompt injection לזיכרון:** הסיכון הוזכר, אבל ההגנה היחידה שתוארה היא permissioning ואישור אנושי.
- **הכיתוב של ה-we don't write prompts anymore, we build loops:** «**repost:**» וגם «42 דקות» לא מופיעים בתמליל. ההקלטה עצמה כ-30 דקות.

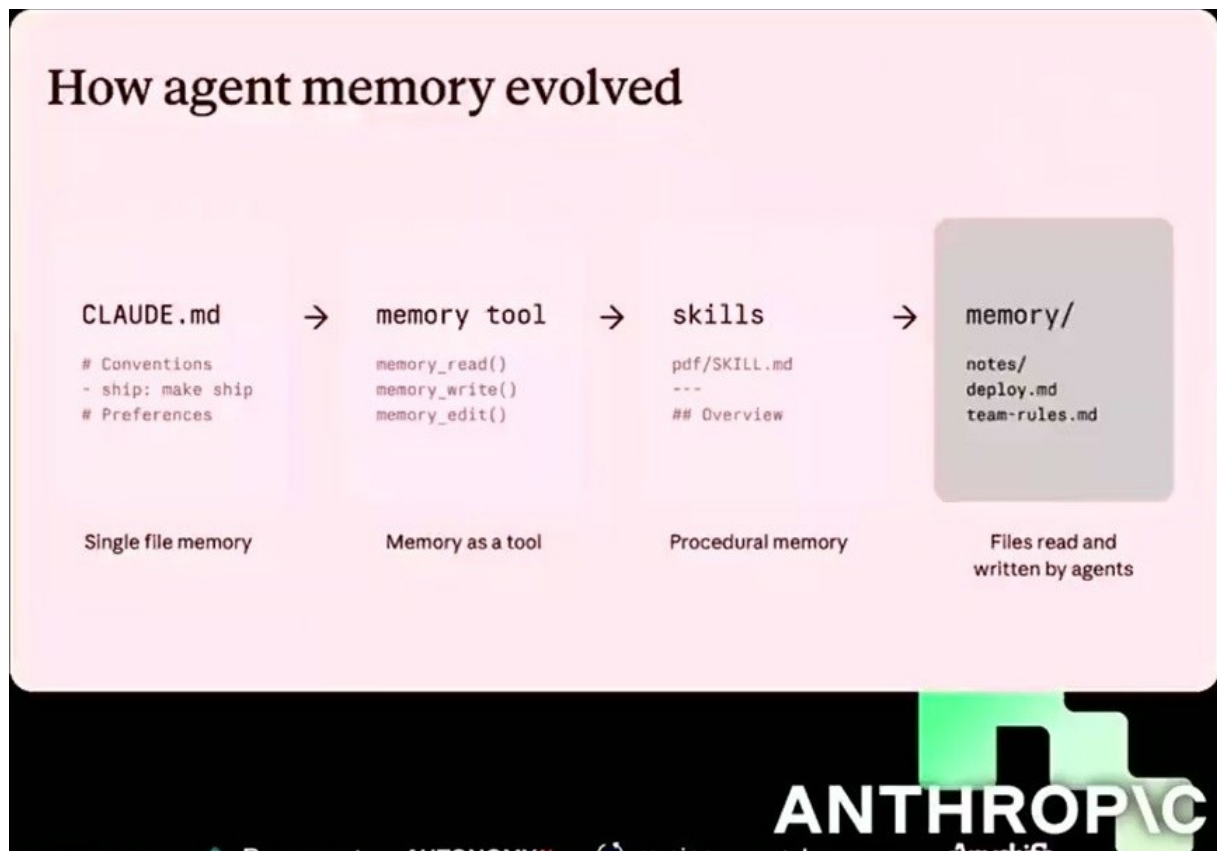
2. למה אינטליגנציה לבד לא מצטברת



שקף הבעיה: *Intelligence alone doesn't compound*, עם שלוש עמודות: *Cold start, Domain-specific, No compounding*

נקודת הפתיחה פשוטה. כל מודל חדש חכם יותר, אבל כשמכניסים אותו ל-agent בתוך ארגון "the intelligence alone is not going to compound because they need this context" [1:30] עם ה-codebase, העדפות של המשתמש והקשר ארגוני, הוא אורתוגונלי לאינטליגנציה של המודל. הסימפטומים מוכרים: agent שלא מתמצא ב-codebase, שלא מכיר את ההעדפות שלך ושחוזר על אותה טעות בפעם הבאה. אין continual learning. לכן, לטענתה, השקעה ב-context engineering משתלמת לאורך זמן גם כשהמודלים משתפרים.

3. האבולוציה של זיכרון agents: מ-CLAUDE.md ל-file system



How agent memory evolved: ציר מ-CLAUDE.md (single file memory), דרך (memory as a tool) memory tool ו-skills (procedural memory), ועד ספריית /memory/ שה-agents קוראים וכותבים בה

At Anthropic, we like to say, do the simple thing" לפי הכלל "do the simple thing that works." [3:01]

CLAUDE.md, Claude Code עם Claude Code, היה יעיל במפתיע: כמה הוראות שמוזרקות בתחילת ה-session מיישרות את ה-agent עם מה שחשוב. הבעיה היא context bloat. מה עושים כשהקובץ הזה נעשה ארוך מאוד? Memory tool הוסיף אוטונומיה: ה-agent מחליט in-band מתי לקרוא, לכתוב ולעדכן זיכרון. לדבריה, האוטונומיה עובדת טוב.

Skills פתרו את הגדילה בעזרת progressive disclosure. ה-agent סורק רק את ה-front matter ויכול לטעון כמה פירוט שצריך מגוף הקובץ. האנלוגיה שלה היא מדף ספרים: אם משהו פונה אליה בצרפתית, היא שולפת מילון צרפתי, ולא צריכה להחזיק בראש שבע שנות לימודי צרפת. המגבלה היא שעדיין בני אדם מחליטים מה ראוי להיות skill.

השלב האחרון, שהיא מגדירה כ-best practice, הוא לבנות את הזיכרון כ-file system: למלא אותו ב-markdown ולתת ל-agent לחפש בו עם Bash ו-grep, במקום להכתיב לו כלים ייעודיים. החיפוש עצמו ממלא את התפקיד של progressive disclosure.

Lamis [7:41] For reading memories, allow the memories to grow large, but give agents tools to — quickly index and search for what is relevant. Finally, give agents autonomy when they are writing to .memories

4. מתיאוריה לפרודקשן: ארבעה עקרונות ל-multi-agent

SCALING TO PRODUCTION

Versioning & concurrency

01 Versioning
Every write is attributed to an author, a session, a time — full history to inspect and roll back.

02 Concurrency
A content-hash precondition before each write — living in the harness, not the model.

team/deploy.md

version: v1 v2 v3 **v4 · head**

written by: **sesn_011Ca7x...**

precondition: **content_sha256: 9f2c...**

content: **Deploy via `make ship`, not `./deploy.sh`.
Reason: user corrected this on PRs #412, #418, #431.**

ANTHROPIC

Scaling to production, Versioning & concurrency: היסטוריית גרסאות של קובץ `team/deploy.md` עם מחבר ובסיס לכל עדכון, ולצידה עיקרון ה-concurrency

המערכת שתוארה בפרק הקודם עובדת היטב ל-agent בודד. בפרודקשן, עם הרבה agents במקביל, ריצות ארוכות ו-codebase מסובך, היא נשברת. התרחישים שהיא מתארת: כמה agents שכותבים לאותו קובץ בו זמנית; agent אחד שכותב טעות ל-context הארגוני שכל השאר קוראים ממנו, והטעות מתפשטת לכל ה-fleet; בני אדם ו-agents שעוברים יחד בלי מעקב; זיכרונות שהתיישנו, נכתבו בטעות או הוזרקו בזדון דרך prompt injection.

הפתרון ל-concurrency הוא מנגנון hash. ה-agent לוקח hash, מנסח את העריכה, ולפני הכתיבה לוקח hash שוב. "If those two things do not match, then the agent cannot write it because it means that some update was made in the meantime." [10:48] `commit`

SCALING TO PRODUCTION

Built for multi-agent systems

03 Permissioning

Read-only access to shared org knowledge; read-write to an agent's own working memory.

04 Portability

It's just files, with a standalone API on top — memory goes where you go.

Agent A session sess_01

Agent B session sess_02

Agent C session sess_03

read-only

org-conventions

notes/

style-guide.md

security.md

oncall.md

read-write

team-memory

deploy.md

flaky-tests.md

repo-layout.md

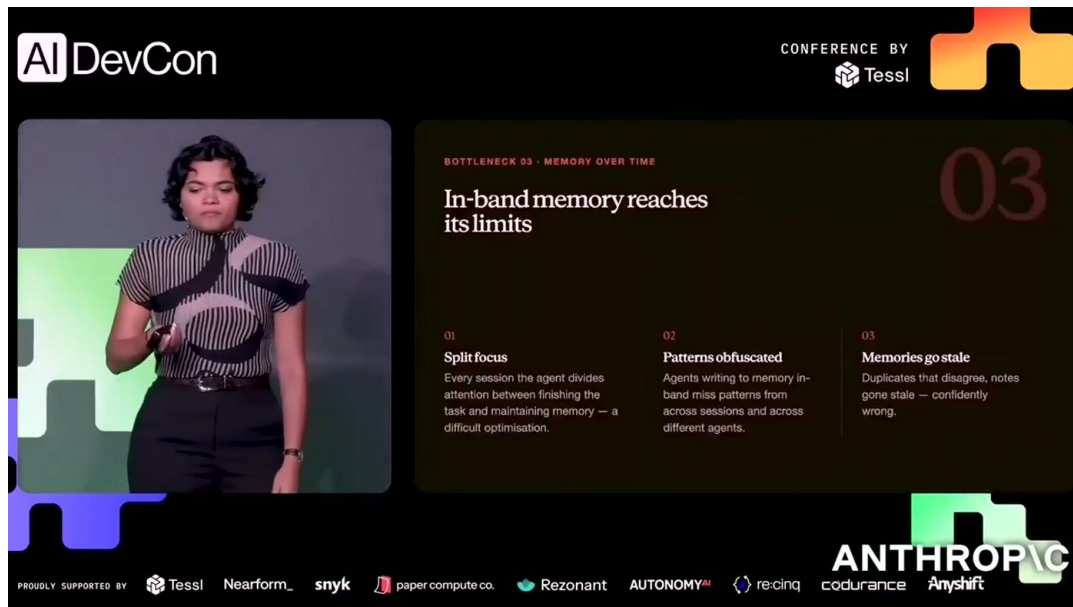
ANTHROPIC

Built for multi-agent systems: Permissioning (קריאה בלבד לידע הארגוני, קריאה וכתובה לזיכרון העבודה של ה-agent) ו-Portability (קבצים עם API עצמאי מעליהם). בתרשים: sessions של agents מחוברים ל-org-conventions ול-team-memory

עיקרון	מה זה אומר בפועל
Versioning	לשמור גרסאות ולאפשר rollback, ולתעד איזה session או transcript הוביל לעדכון ומי ביצע אותו, agent או אדם
Concurrency	בדיקת hash לפני ואחרי ניסוח העריכה. אם אין התאמה, מושכים מחדש ומנסים שוב (optimistic concurrency)
Permissioning	היררכיה מהידע הארגוני המתוחזק (read-only) ועד scratch pad אישי של agent (read-write), עם שכבות ביניים לצוותים
Portability	הזיכרון הוא נכס ארגוני, ולכן צריך API נקי שמאפשר גישה מכמה משטחי מוצר ומכמה מערכות

התוצאות שהיא מדווחת מפריסות כאלה הן איכותיות בלבד: דיוק טוב יותר בפעם השנייה שה-agent עושה משימה, ובעקבות זה פחות tokens, יותר one-shot ועלות נמוכה יותר. היתרון השלישי הוא שצוות המוצר מתפנה לעבודה על המוצר בזמן שלולאת הלמידה רצה ברקע [12:22].

5. גבולות ה-in-band, ו-dreaming כפתרון

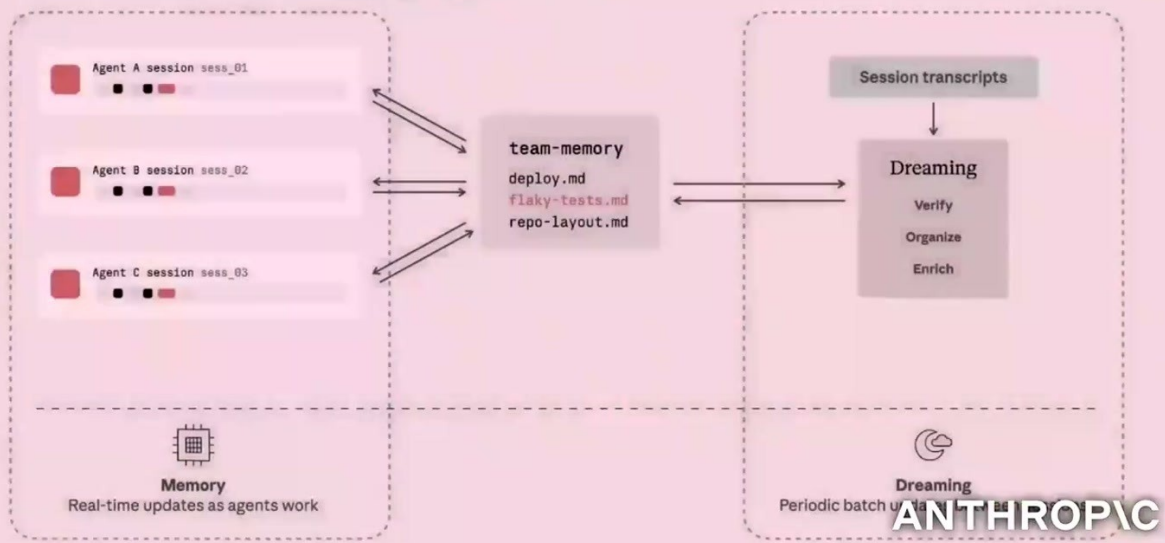


In-band memory reaches its limits: שלוש מגבלות, Split focus, Patterns obfuscated, ו-Memories go stale

זיכרון in-band, כלומר כתיבה וקריאה בתוך ה-session, נתקע מסיבות ארכיטקטוניות. הראשונה היא פיצול משאבים: "how much capacity should an agent put into helping future versions of itself, versus doing the task" [13:55] "that you actually asked it to do?". השנייה היא נראות: ה-agent רואה רק את ה-session שלו, ולכן לא מזהה טעות שחוזרת בין sessions או כשלים של agents אחרים ב-fleet. השלישית, שהיא שכחה ואז הוסיפה, היא שזיכרונות מתיישנים ומישהו צריך לבדוק אותם.

האנלוגיה שלה היא בית ספר: תלמידים מגישים עבודות, מורים בודקים, ומנהלת רואה את התמונה הכוללת ומעדכנת את תוכנית הלימודים. מכאן dreaming, "a process that runs in batch and asynchronously with its own allocated resources to ensure that those memories themselves are effective, up-to-date, and helping the agents learn over time." [17:02]

A unified memory system



A unified memory system: משמאל agent sessions שמעדכנים בזמן אמת (deploy.md, flaky-tests.md, repo-) אמת team-memory במרכז, ומימין dreaming כ-batch תקופתי על session transcripts עם השלבים Verify, Organize ו-Enrich (layout.md)

התמונה הסופית מורכבת משני תהליכים שרצים במקביל. Memory in-band נותן שיפור כבר ב-session הבא, אבל מוגבל במשאבים ובנראות. Dreaming out-of-band מקבל נראות רחבה ו-token spend ייעודי. השאלה הצפויה היא אם זה לא יקר מדי. התשובה שלה היא שזיכרון טוב מוריד עלויות, כי agents מצליחים ב-one-shot. היא גם מדגישה שזה לא ספציפי לקוד: "[23:11] this by no means is just specific to coding". היא עצמה משתמשת בזיכרון כשהיא מכינה מצגות.

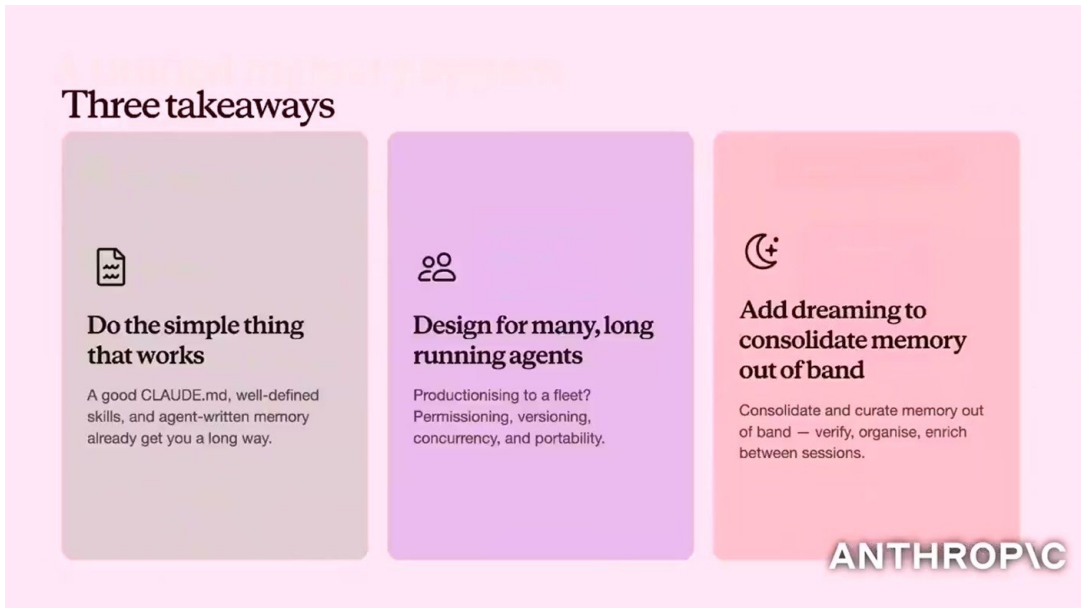
6. שאלות מהקהל

פתרון enterprise מוכן: היא אומרת שהשתדלה לא לקדם מוצר, אבל ש-versioning, hashing ו-dreaming כולם "available within our memory and dreaming API through Claude Managed Agents" [26:13]. ההרצאה מתארת את הארכיטקטורה של התשתית המנוהלת של Anthropic.

הרשאות ב-dreaming: שואל אחד הזכיר את «code code leak», כנראה דליפת הקוד של Claude Code שבה הופיע מנגנון dreaming. הניסוח משובש ב-ASR ולא אומת. השאלה הייתה איך dreaming מכבד הרשאות של מאות משתמשים. התשובה: בכל עבודת dreaming בוחרים במפורש אילו transcripts לצרף, ולכן אפשר לסנן לפי אותו permission set של ה-memory store [27:44].

השאלה החדה ביותר: "[27:44] At what point are we reinventing databases from first principles again?". התשובה שלה היא שהמטרה היא למצוא את הגבול בין מה שה-agent עושה באוטונומיה לבין מה שנכתב בקוד בתוך harness-ה. אחרי שראו אילו primitives עובדים, הם מקבעים אותם: "We are kind of going back to the software engineering principles that we've seen work well in the past" [29:14].

7. מה לעשות עכשיו



Three takeaways: Do the simple thing that works, Design for many long-running agents, Add dreaming to consolidate memory out of band

- **להתחיל פשוט:** CLAUDE.md מתוחזק, skills מוגדרים היטב וזיכרון שה-agent כותב בעצמו. לדבריה זה לבד "goes a really long way".
- **לבנות זיכרון כספריית markdown:** לתת ל-agent לחפש בה עם grep ו-Bash, ולא לכתוב כלי זיכרון ייעודיים.
- **לפני שמרחיבים ל-fleet:** להוסיף versioning עם provenance, כלומר איזה transcript ומי ביצע, concurrency מבוסס hash, והרשאות בשכבות: ידע ארגוני read-only ו-scratch pad בקריאה וכתיבה.
- **להריץ תהליך consolidation תקופתי:** לעבור על transcripts כולל tool calls, לזהות כשלים חוזרים ולהציע diff לזיכרון עם דוגמאות ושכיחות, ובאישור אנושי.
- **לסכן transcripts לפי הרשאות:** בכל ריצת dreaming לצרף רק transcripts מאותו permission set של ה-memory store.

שאלה פתוחה	למה היא חשובה
כמה עולה ריצת dreaming ביחס לחיסכון ב-tokens?	הטענה שהעלות מחזירה את עצמה לא גובתה במספרים
מה סף השכיחות שמצדיק שינוי בזיכרון?	בלי סף, dreaming עלול לרדוף אחרי רעש או לקבע טעות
איך מזהים זיכרון שהוזרק בדון?	prompt injection הוזכר כסיכון, אבל לא הוצג מנגנון זיהוי
מתי זיכרון נמחק ולא רק מתעדכן?	Verify ו-Organise מופיעים בשקף, אבל קריטריון להסרה לא פורט
האם כדאי לבנות לבד או להשתמש ב-Claude Managed Agents?	ההרצאה מתארת ארכיטקטורה שזמינה כמוצר, ונשארת שאלה של lock-in

8. מילון מונחים

מונח	משמעות
Context engineering	תכנון וניהול של המידע שנכנס לחלון ההקשר של המודל
Context bloat	התנפחות ההקשר בגלל קבצים או הוראות ארוכים מדי
CLAUDE.md	קובץ markdown עם הוראות שמוזרק לתחילת ה-session ב-Claude Code
Memory tool	כלי שמאפשר ל-agent לקרוא ולכתוב זיכרון בעצמו
Skills	קבצי תהליך שנטענים לפי צורך, כשה-agent קורא קודם רק את ה-front matter
Progressive disclosure	חשיפה הדרגתית: תקציר קודם, ופירוט רק כשצריך
In-band / Out-of-band	בתוך ה-session של המשימה, לעומת תהליך נפרד שרק ברקע

מונח	משמעות
Dreaming	תהליך batch אסינכרוני שעובר על transcripts ומציע תיקונים ל-memory store
Memory store	אוסף הזיכרונות, בדרך כלל ספריית קבצי markdown
Orchestrator / Sub-agents	agent מתאם שמחלק עבודה ל-agents משניים ומאחד את הממצאים שלהם
Versioning	שמירת גרסאות עם אפשרות rollback ותיעוד מקור לכל שינוי
Concurrency	טיפול בכתיבות מקבילות, כאן באמצעות השוואת hash לפני ה-commit
Permissioning	הרשאות קריאה וכתיבה לפי שכבה: ארגון, צוות, agent
Portability	גישה לזיכרון מכמה מערכות ומשטחי מוצר דרך API
Harness	שכבת הקוד שעוטפת את המודל ומריצה אותו כ-agent
One-shot	הצלחה במשימה בניסיון הראשון
Continual learning	שיפור מתמשך של ה-agent מריצה לריצה
Claude Managed Agents	השירות המנוהל של Anthropic שבו זמין ה-memory and-dreaming API, לפי המרצה