ה-sub-agents שלכם הם מערכת מבוזרת. תתייחסו אליהם ככה.

Girish Konda ממפה את התקלות של orchestrator ו-sub-agents לרעיונות מוכרים ממערכות מבוזרות, ומציע מודל עקביות בשם SAFE hand-offs

Girish Konda (Microsoft) · AI Engineer · 10.11.2026

משך: כ-23 דקות · YouTube

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה קצרה של כ-23 דקות מערוץ AI Engineer. מעביר אותה Girish Konda, principal software engineer ב-Microsoft, שמתחזק גם כלי קוד פתוח ל-agents, ביניהם MCP Pacemaker, שהוא bridge מפוקח שמחזיק שרתי MCP בחיים. הוא מציין בתחילת ההרצאה שהדעות בה הן שלו ולא של המעסיק.

הטענה המרכזית: כש-orchestrator מפצל משימה בין sub-agents שרצים במקביל, קוראים מאותו state משותף ומבצעים tool calls מול מערכות אמיתיות, נוצרת מערכת מבוזרת לכל דבר. לכן מקבלים גם את הכשלים הקלאסיים שלה: partial failure שמדווח כהצלחה, dirty reads, fan-out בלי גבול ו-retries שמריצים side effects פעמיים. לכל כשל הוא מצמיד את המושג הקיים שמסביר אותו, ומרכיב מהם מודל עקביות שהוא מכנה SAFE hand-offs. ההרצאה מסתיימת בפרק מפורש על מה שהמודל לא עושה ועל המחיר שלו.

איך לקרוא את המסמך

- **1. תקציר מנהלים:** הטענות המרכזיות עם חותמות זמן, ומה לא נאמר.
- **2. הארכיטקטורה:** למה orchestrator עם sub-agents הוא מערכת מבוזרת, וארבעת הכשלים.
- **3. מודל הכשל:** תשובות חצי-גמורות, timeouts כחשד ו-failure כערך.
- **4. Dirty reads ובחירת מודל עקביות:** למה quorum של agents לא עובד, ואיך הגיע ל-SAFE.
- **5. S ו-A:** Sourced reads ו-At-most-once effects, כולל הדוגמאות מ-MCP Pacemaker.
- **6. F ו-E:** Fan-out budgets ו-Explicit outcomes.
- **7. גבולות ומחיר:** מה SAFE לא עושה, ואיפה הוא לא מתאים.
- **8. מה עושים עכשיו:** צעדים מעשיים וטבלת שאלות פתוחות.
- **9. מילון מונחים:** המונחים באנגלית שמופיעים במסמך.

הערת מקור המסמך נוצר אוטומטית מהתמליל ומהשקפים של ההקלטה. שמות ומונחים עלולים להשתבש בזיהוי הדיבור (ASR): שם הדובר מופיע בתמליל כ-Grish, וכאן הוא מתוקן לפי השקף ל-Girish Konda. שם המודל מופיע בתמליל כ-SAVE HANDOFFS ומתוקן לפי השקף ל-SAFE. ציטוטים מובאים מילה במילה ונבדקו מול חותמת הזמן שלהם. לא נוסף שום מידע ממקורות חיצוניים.

1. תקציר מנהלים

Konda לא מציע framework חדש. הוא מציע מסגרת חשיבה: מה ש-sub-agent מחזיר הוא תוצאה של network call לא אמין, ומה שהוא כותב ל-context משותף הוא כתיבה שעוד לא עברה commit. מהנקודה הזו נובעים ארבעה כללים, אחד לכל אות ב-SAFE, וכולם נאכפים בקוד של ה-orchestrator ולא ב-prompt.

- **Partial failure נראה כמו הצלחה:** sub-agent שעשה חצי עבודה לא זורק שגיאה. הוא מחזיר טקסט רהוט עם אותו type signature כמו הצלחה. בדוגמת ה-alert queue, שלוש התראות סווגו benign באותה רמת ביטחון, למרות שרק באחת מהן נעשתה בדיקה בפועל.
- **Timeout הוא חשד ולא עובדה:** ב-MCP Pacemaker שרת איטי נראה מת בגלל timeout של 30 שניות, ובמקרה ההפוך timeout דרס את exit code שהסביר קריסה. הכלל שלו: לרשום unknown ולשמור את הסיבה האמיתית.
- **הסכמה בין agents היא לא quorum:** שלושה sub-agents שקוראים את אותו context משותף הם לא עותקים בלתי תלויים, ולכן ההסכמה שלהם היא מקור אחד שנספר שלוש פעמים.
- **S, Sourced reads:** כל עובדה היא רשומה עם מקור, גרסה ושלוש חותמות זמן, והטריות שלה נבדקת ברגע השימוש. החלטה ב-9:15 על נתון שנקרא ב-9:06, עם גבול של 5 דקות, נדחית.
- **A, At-most-once effects:** לכל side effect יש זהות ואישור חד-פעמי שמשמש כמו fencing token. אחרי קריאה שהתשובה עליה אבדה בודקים את התוצאה לפי הזהות ולא מריצים מחדש.
- **F, Fan-out budgets:** ב-Pacemaker, cold starts מקבילים דרך NPX שברו את ה-package cache. הפתרון היה שער של 2 במקביל, המתנה של עד 15 שניות, ותקרת sessions שבה סירוב נספר ככשל.
- **E, Explicit outcomes:** פלט של agent הוא קלט לא מהימן. ה-orchestrator בודק בקוד schema, ראיות, טריות וחישוב מחדש של מספרים. הטענה "error rate ירד" (10/1000 מול 6/600) נדחית כי בשני המקרים השיעור הוא 1%.
- **למודל יש מחיר והוא נאמר בגלוי:** SAFE בוחר עקביות על פני זמינות בכתיבות, מוסיף latency לכל בדיקת טריות, ולא מתאים למערכת עם כמה orchestrators או ל-agents שרק מנסחים טקסט.

מה לא נאמר

- **אין מדידות:** אין נתונים על תדירות הכשלים, על עלות ה-latency של SAFE או על השיפור שהמודל הביא. הדוגמאות על triage ועל error rate מוצגות במפורש כ-made-up examples.
- **אין קוד או API:** לא מוצג איך נראה בפועל ה-schema של תוצאת sub-agent, ואין ספרייה מוכנה מלבד ה-bridge עצמו.
- **Multi-orchestrator ו-peer agents:** הוא מציין שהמודל לא מתאים להם, אבל לא מציע חלופה.
- **Compensation:** הוא מפנה ל-Sagas לגבי ביטול משימה חלקית, אבל לא מראה איך לתכנן את זה.
- **Internet draft:** הוא מזכיר internet draft אישי על evaluation states בלי שם או מספר, ולכן אי אפשר לאתר אותו מתוך ההרצאה.

2. הארכיטקטורה: sub-agents הם מערכת מבוזרת



שקף הפתיחה של חלק הארכיטקטורה: "Sub-agents are a distributed system"

התרחיש: orchestrator מפצל משימה לשלושה sub-agents שרצים במקביל. כולם קוראים מאותו state משותף, כמו conversation, קובצי memory, ה-repo, working trees, issues, ו-vector stores. הם גם מבצעים tool calls מול מערכות אמיתיות: databases, APIs, CI ו-deployment pipelines. בסוף ה-orchestrator ממזג את התוצאות לתשובה אחת. בניסוח שלו, זה shared mutable state ומיזוג של תוצאות מהרצות מקביליות, כלומר מערכת מבוזרת קלאסית.

ההבדל לעומת מערכות מבוזרות רגילות הוא שאין כאן אף אחד מהכלים המוכרים. "What you get is a function that takes a string and returns a string.". ארבעת הכשלים שהוא מונה מוכרים כולם: partial failure שחוזר כטקסט שנראה כמו הצלחה, dirty reads מ-context משותף שבו גם ניחושים של worker אחר קריאים, fan-out בלי גבול שבו ה-orchestrator לא רואה 429 או 503, ו-retries שמריצים מחדש משימה שלמה כולל ה-side effects שלה.

3. מודל הכשל: מה שנראה כמו הצלחה

הוא מתחיל מסולם הכשלים. crash-stop הוא המקרה הנוח, כי התהליך עוצר ונשאר עצור. no reply קשה יותר. המקרה הקשה ביותר מגיע מבעיית Byzantine Generals (ב-"ASR: "Byzantine General Cyber"): רכיב שעונה, והתשובה שגויה. sub-agent שגמר חצי עבודה לא זדוני, אבל מבחוץ הוא נראה בדיוק כך.

הדוגמה היא agent שהופנה לתור התראות. בהתראה הראשונה הוא משך telemetry והשווה מול סט benign ידוע. בשנייה הוא זיהה כותרת שכבר ראה ונתן את אותה תשובה. בשלישית הוא לא עשה כלום. שלושתן סווגו benign באותה רמת ביטחון.

> **Girish Konda** The orchestrator can read the conclusion. It cannot read how much work was behind it. —

לגבי timeouts הוא מסתמך על FLP מ-1985: במערכת אסינכרונית שיש בה אפילו תהליך אחד תקול, ייתכן שהסכמה לא תושג לעולם. התשובה מ-1996 היא failure detectors שמותר להם לטעות (ב-"ASR: "Chandra and Toueg), כנראה Chandra ו-2X". ב-MCP Pacemaker הוא נתקל בשני הכיוונים. שרת שמוריד את עצמו בהרצה הראשונה עבר את ה-timeout של 30 שניות ונראה שבור, אבל היה רק איטי. התיקון היה timeout נפרד ל-initialize. במקרה ההפוך שרת קרס באמצע קריאה, ה-timeout שהגיע אחר כך דרס את ה-exit code שהסביר את הקריסה, והתיקון היה לרשום timeout רק כשהתהליך עדיין חי. המסקנה: "Record it as unknown and keep the real cause.".

מכאן הוא מגיע ל-failure כערך. sub-agent צריך להחזיר status, מה עשה, מה לא עשה, ראיות ו-side effects, ו-partial היא תשובה לגיטימית. בפרוטוקולים כבר יש לזה מקום: ב-A2A משימה מסתיימת כ-completed, failed, cancelled או rejected, או נעצרת ומבקשת input. ב-MCP תוצאת tool יכולה לסמן isError true. הבעיה היא ש-sub-agent עדיין יכול לדווח completed כשהטקסט אומר משהו אחר, ולולאה שקוראת רק את הטקסט לא תשים לב. עוד הבחנה שהוא מדגיש: "בדקתי והתשובה היא לא" ו"לא הצלחתי לבדוק" הן שתי עובדות שונות. מי שמאחד אותן משנה מה ינסו שוב, את מי יזעיקו ומה יופיע בדוחות אחר כך.

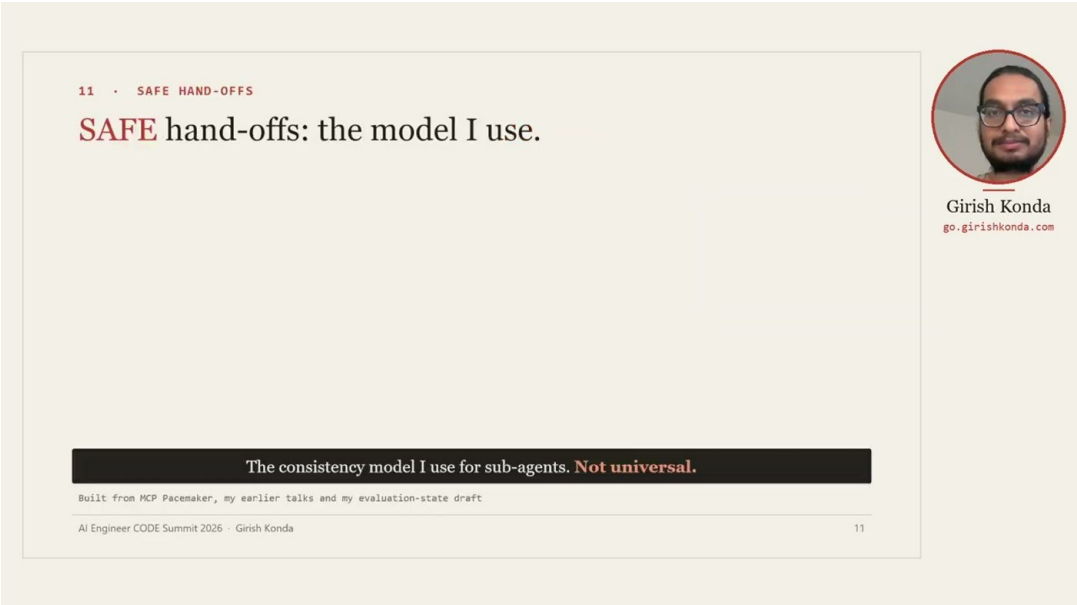
4. Dirty reads, quorum מדומה ובחירת מודל עקביות

הדוגמה, שגם היא מוגדרת made-up: sub-agent של triage כותב ל-context המשותף ש-error rate ירד ב-80%. הוא לא מדד את זה אלא הסיק. sub-agent שני מעתיק את המשפט ל-review note, ושלישי משתמש בו בתור נימוק להמשיך ב-runbook. ה-orchestrator ממזג שלושה ארטיפקטים שמסכימים זה עם זה, אבל בפועל מדובר במשפט אחד לא מבוסס שנקרא שלוש פעמים. לפי הביקורת על SQL isolation levels מ-1995, dirty read הוא קריאה של ערך לפני שהכותב עשה commit או rollback. ב-context משותף ניחוש קריא מהרגע שנכתב, ואין שום סימון שהוא לא נבדק.

הפתרון האינטואיטיבי, להריץ שלושה agents ולקחת quorum, לא עובד כאן. ב-weighted voting מ-1979 ההבטחה נובעת מכך ש-read quorum ועוד write quorum גדולים מסך הכול, כך שכל קריאה חופפת לכתיבה האחרונה. החפיפה היא מה שנותן את ההבטחה.

> **Girish Konda** — They are not independent copies. They read the same guess. Their agreement is one source counted three different times.

אחר כך הוא עובר על סולם מודלי העקביות: read your writes, monotonic reads) linearizable, sequential, causal, session guarantees) ו-eventual. לטענתו context משותף לא מבטיח אף אחד מהם. גם לא צריך את המודלים החזקים, שלא יכולים להישאר זמינים כשהרשת מתפצלת. causal ו-session guarantees כן יכולים, בתנאי שכל client נצמד למקום אחד, ובמקרה של sub-agents המקום הזה הוא ה-orchestrator.



"SAFE hand-offs: the model I use", ובתחתית ההסתייגות: The consistency model I use for sub-agents. Not universal.

אות	שם	הכלל
S	Sourced reads	כל עובדה שעוברת בין workers נושאת את גרסת המקור שלה (זמן ו-revision), והחלטות נשענות רק על עובדות טריות
A	At-most-once effects	לכל כתיבה יש זהות ואישור חד-פעמי לאותה פעולה בדיוק. את התוצאה קוראים בחזרה ולא מריצים שוב
F	Fan-out budgets	לכל downstream יש שער והמתנה חסומה בזמן, וסירוב נספר ככשל

אות	שם	הכלל
E	Explicit outcomes	כל hand-off מסתיים במצב בעל שם. timeout נרשם כ-unknown, והסיבה נשמרת

5. S ו-A: מקור לכל עובדה, זהות לכל side effect



שקף מ-1978 Lamport של happened-before דוגמה להמחשה המבוססת על "Causality: "Provenance is causal metadata".

Sourced reads הוא המענה ל-dirty reads. עובדה צריכה להיות רשומה ולא משפט, והוא מסתמך על המאמר של Lamport (ב-"Lampard :ASR") מ-1978 על happened-before. הרשומה כוללת את הטענה וה-revision שלה, את המקור והגרסה שלו, את ה-scope ושלוש חותמות זמן: מתי הנתון היה עדכני, מתי נקרא ומתי נעשה בו שימוש. הטריות נבדקת בזמן השימוש. בדוגמה הנתונים מסתיימים ב-9:05, נקראו ב-9:06, וההחלטה ב-9:07 תקינה תחת גבול של 5 דקות. החלטה שנייה ב-9:15 נדחית ודורשת קריאה מחדש.

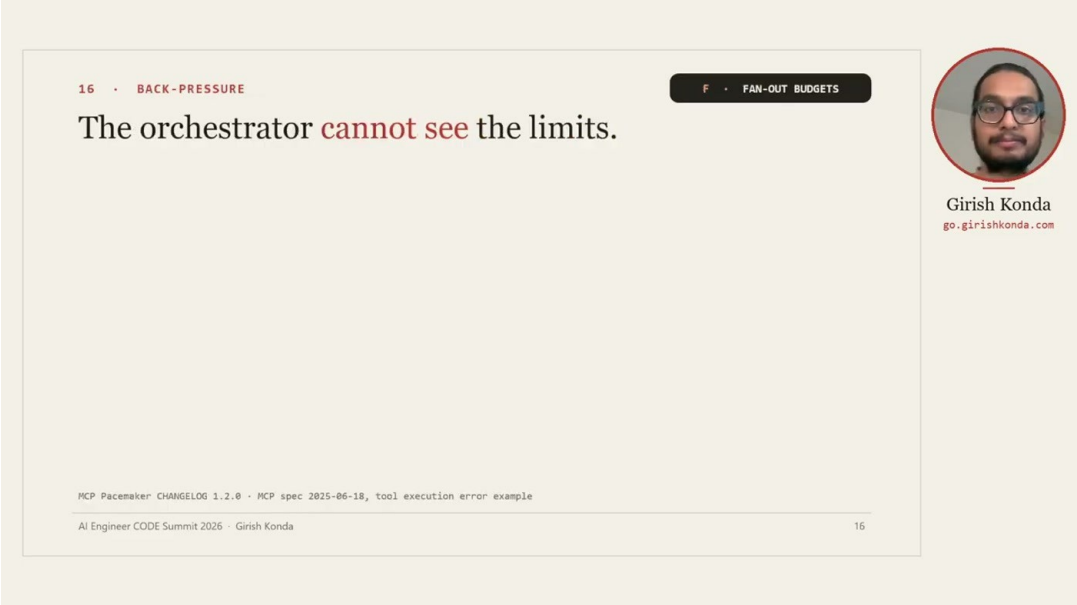
> **Girish Konda** — This is how you separate an observation from an inference. An observation has a source and a read time. An inference has none.

At-most-once effects עוסק ב-retries. ה-orchestrator מאציל משימה, ה-sub-agent קורא ל-tool שמבצע כתיבה אמיתית, למשל פתיחת PR, והתשובה אובדת. מה שה-orchestrator יודע הוא רק שהתוצאה לא ידועה, ולכן הוא מריץ את כל המשימה מחדש וה-PR נפתח פעמיים. מאמר ה-RPC מ-1984 כבר קבע שכשמתקבל exception, הפרוצדורה רצה פעם אחת או לא רצה בכלל, ואין דרך לדעת איזה מהשניים. לכן צריך לבחור delivery semantics: at most once, at least once, או exactly once באמצעות deduplication עם idempotency key. ב-MCP, idempotentHint הוא רק רמז וברירת המחדל שלו היא false, כך שמי שצריך idempotency צריך לבנות אותה.

בתכנון של ה-bridge, כש-client חוזר אחרי restart, ה-initialize שלו משוחזר מול שרת חדש. זה בטוח כי מדובר רק ב-handshake, והוא לא משחזר tool call. tool state שהתוצאה שלו לא ודאית לא משוחזר לעולם: כששרת משותף ממוחזר, קריאות שלא נענו עד ה-deadline נכשלות כ-potentially executed (ב-"potentially excluded :ASR"), סטטוס שאומר ל-caller ללכת ולבדוק. הכלל: לכל side effect יש זהות, האישור קשור לפעולה הספציפית ומשמש פעם אחת, ואחרי הקריאה בודקים את התוצאה לפי הזהות. "Submitted is not confirmed. Confirm means you observe their effect."

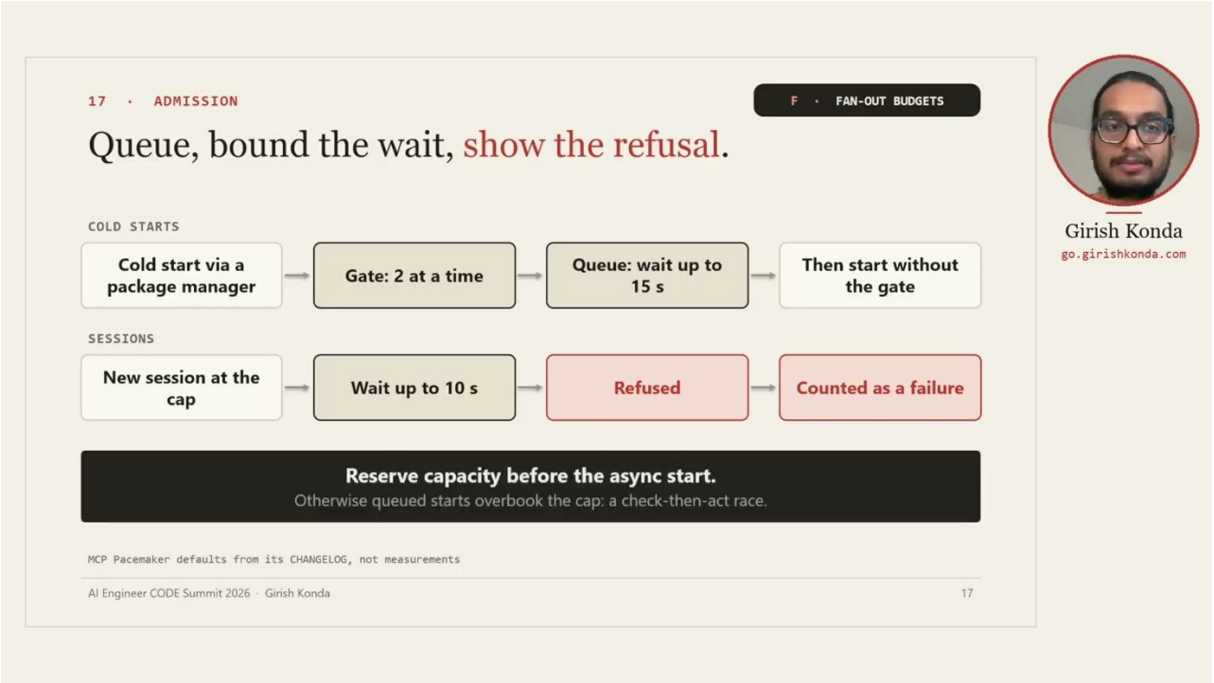
החלק האחרון של A נוגע ל-pause ו-resume. לפי הפוסט של Martin Kleppmann על distributed locking, lease הוא lock עם client. timeout שנעצר עם ה-lease בידיו מתעורר אחרי שמישהו אחר כבר לקח אותו, וכותב בכל זאת. הפתרון הוא fencing token, מספר שעולה בכל הענקה, כשה-storage דוחה כתיבות עם token ישן. אצל sub-agents זה מתורגם לכלל "restore context, not side effects": אישור מתנהג כמו fencing token, כלומר פעולה אחת ושימוש אחד, והוא נבדק בשער ולא נשמר ב-prompt.

6. F ו-E: תקציב fan-out ותוצאות מפורשות



שקף "Back-pressure: "The orchestrator cannot see the limits". השוליים מפנים ל-MCP Pacemaker ולדוגמת tool execution error במפרט MCP

Fan-out בלי גבול הוא בעיית flow control. ב-Pacemaker, לפני שהייתה הגבלה, כמה sessions הפעילו את השרתים שלהם דרך NPX בו-זמנית, וחלקו package cache אחד. כולם נכשלו עם שגיאת lock compromised של NPM. "Nothing in any single request was wrong. The failures was the concurrency itself". בעיה נוספת היא שגם כש-downstream מחזיר back-pressure, הוא עשוי להגיע כטקסט פשוט: בדוגמה ממפרט MCP, rate limit הוא משפט אחד. המודל קורא אותו, הקוד של ה-orchestrator לא, וה-worker הבא מתחיל בכל מקרה.



תרשים Admission: cold start דרך package manager עובר שער של 2 במקביל, ממתין עד 15 שניות ואז מתחיל בלי השער. session שמגיע לתקרה ממתין עד 10 שניות, נדחה, והסירוב נספר ככשל. למטה: Reserve capacity before the async start

הפתרון ב-bridge הוא quota budget. cold starts דרך package manager עוברים שער של שניים במקביל כברירת מחדל, והשאר ממתינים עד 15 שניות. אחרי זה ה-start ממשיך בלי השער, וההחלטה מתועדת: cache פגום הוא סיכון, ו-client request תקוע הוא ודאות. זו בחירה מכוונת בזמינות. בנוסף יש תקרת sessions: session חדש ממתין עד 10 שניות ואז נדחה, והסירוב נספר ככשל ב-health. באג נוסף שמצא: את הקיבולת צריך לשריין לפני

ה-start האסינכרוני ולא אחריו, אחרת starts שממתינים בתור עוברים את התקרה. זה race מסוג check-then-act. לגבי sub-agents ההמלצה היא לתת ל-orchestrator מספר לכל downstream, המתנה חסומה בזמן וסירוב שהוא יכול לראות.

Explicit outcomes סוגר את המודל. כל hand-off מסתיים במצב בעל שם, כמו stale או unknown, ולא בטקסט חופשי. השאלה החשובה היא מי מחליט על המצב: פלט של agent הוא קלט לא מהימן, בדיוק כמו request body שמגיע מהאינטרנט. ה-orchestrator מריץ את הבדיקות בקוד: האם התוצאה תואמת ל-schema, האם יש ראיות, האם הן טריות, והאם המספרים מסתדרים בחישוב מחדש. בדיקה שנכשלת מחזירה סירוב עם סיבה, ולא טקסט עם פסקת אזהרה מתחתיו. בדוגמה, sub-agent טוען ש-error rate ירד מ-10 שגיאות ב-1000 בקשות ל-6 שגיאות ב-600. בחישוב מחדש השיעור הוא 1% בשני המקרים, והטענה נדחית.

> **Girish Konda** If the same model can skip a check and declare it passed, you have a request, not a — boundary.

7. גבולות ומחיר: מה SAFE לא עושה



שקף "Limits: "What SAFE does not do". בשוליים: CAP של Gilbert ו-Lynch מ-2002 ו-PACELC של Abadi מ-2012

- **לא הופך עובדות לנכונות:** source fact הוא עובדה שאפשר לבדוק, וזה לא אומר שהיא נכונה.
- **לא נותן exactly-once delivery:** תשובה שאבדה נשארת unknown עד שבודקים את התוצאה.
- **לא עושה rollback:** ביטול משימה חלקית דורש compensation step שמתכננים בנפרד, בדומה ל-Sagas מ-1987.
- **אין סדר גלובלי בין workers:** יש רק causal order ברמת העובדה הבודדת.
- **מחיר CAP:** ה-bridge בוחר בכוונה בזמינות ל-cold starts, אבל בכתיבות SAFE בוחר עקביות. "It refuses rather than guesses."
- **מחיר PACELC (ב-Backlack "ASR:"):** גם בלי כשלים משלמים latency תמורת עקביות. כל בדיקת טריות היא עוד קריאה, וכל סירוב צריך בעלים.
- **איפה זה לא מתאים:** sub-agents שרק מנסחים טקסט שאדם קורא לפני שקורה משהו, שם רוב זה overhead. גם מערכות עם יותר מ-orchestrator אחד או peer agents בלי נקודת עיגון אחת לא מתאימות.

8. מה עושים עכשיו



שלושת ה-takeaways: להתייחס לכל hand-off כאל network call, להחזיר status ולא רק טקסט, ולהתחיל מ-SAFE ולאכוף אותו בקוד

- **להגדיר schema לתוצאה של sub-agent:** status (כולל partial, stale ו-unknown), מה בוצע, מה לא בוצע, ראיות ו-side effects. ה-orchestrator קורא את השדות ולא את הטקסט.
- **להפריד בין observation ל-inference ב-context המשותף:** עובדה בלי מקור ובלי זמן קריאה מסומנת כהסקה ולא משמשת להחלטות.
- **לתת זהות לכל side effect:** אישור חד-פעמי לפעולה ספציפית, ובמקום retry עיוור של משימה שלמה לבדוק את התוצאה לפי הזהות.
- **לרשום timeout כ-unknown:** בלי לדרוס את ה-exit code או את הסיבה האמיתית.
- **לקבוע תקציב לכל downstream:** שער concurrency, המתנה חסומה בזמן וסירוב שה-orchestrator רואה. לשריין קיבולת לפני ה-start.
- **לחשב מחדש מספרים בקוד:** לפני שטענה כמותית של agent עוברת הלאה.

> **Girish Konda** We already know how to build distribute systems. We just have to notice that we are — building one.

שאלה פתוחה	למה היא חשובה
מה עושים כשיש כמה orchestrators או peer agents?	SAFE נשען על נקודת עיגון אחת, וההרצאה לא מציעה חלופה
כמה latency ועלות מוסיפות בדיקות הטריות והסירובים?	לא הוצגו מדידות, ולכן קשה להעריך את ה-trade-off מראש
איך בוחרים גבול טריות (כמו 5 הדקות בדוגמה) לכל סוג נתון?	הגבול בדוגמה נבחר שרירותית, ואין הנחיה איך לקבוע אותו
איך נראה compensation step טוב למשימת agent חלקית?	SAFE לא עושה rollback, ורק מפנה ל-Sagas
האם אפשר לאכוף את המודל בפרוטוקולים עצמם (A2A, MCP)?	כרגע idempotentHint הוא רק רמז, ו-completed יכול לסתור את הטקסט

9. מילון מונחים

מונח	משמעות
Orchestrator	ה-agent שמפצל משימה, מאציל חלקים ל-sub-agents וממזג את התוצאות
Partial failure	כשל באמצע פעולה. אצל agents הוא חוזר כטקסט שנראה כמו הצלחה
Byzantine failure	רכיב שעונה, אבל עם מידע שגוי או סותר

מונח	משמעות
Failure detector	מנגנון שמחשיד רכיב בכשל ועלול לטעות. timeout הוא חשד ולא ודאות
Dirty read	קריאה של ערך לפני שהכותב עשה commit או rollback
Quorum	קבוצת קוראים או כותבים שהחפיפה ביניהם מבטיחה שקריאה תראה את הכתיבה האחרונה
Provenance	מקור העובדה: ממה נקראה ומתי
Idempotency key	מזהה שמאפשר לזהות ולסנן ביצוע כפול של אותה פעולה
Fencing token	מספר שעולה בכל הענקת lock, וה-storage דוחה כתיבות עם מספר ישן
Back-pressure / Flow control	מנגנון שבו שירות עמוס מאט את מי שפונה אליו (429, 503, תורים)
Check-then-act race	מצב שבו בדיקה ופעולה לא אטומיות, ולכן המצב משתנה ביניהן
CAP / PACELC	ה-trade-off בין עקביות לזמינות כשהרשת מתפצלת, ובין עקביות ל-latency כשהיא לא מתפצלת
Saga / Compensation	ביטול של משימה חלקית באמצעות צעדים הפוכים שמתכננים מראש
MCP Pacemaker	ה-bridge המפוקח בקוד פתוח של הדובר, שמחזיק שרתי MCP בחיים