# Dependency Snapshot

Facilitator guide

**Your job:** trace what the stuck work is really waiting on — node by node, to the far end of the chain — and leave with the source blocker circled and one move aimed at it. You are not solving the blocker in the room; you are finding it.

**The one idea:** the thing that's stuck usually isn't the thing that's blocking you. When work stalls, the team pushes on the most visible stuck task — but that task is waiting on something, which is waiting on something else. The real blocker sits at the far end of a "waiting on" chain. Fix the source, not the symptom.

· WHEN DO I USE THIS?

## Use this when

### GOOD FIT

Work has stalled and the cause isn't obvious. Several things are stuck, people keep chasing the task in front of them, and clearing it doesn't help. You can point to real, stalled work — not a hypothetical.

### WRONG TOOL

The blocker is already obvious — just clear it. The question is where time is lost inside one piece of work › Constraint Snapshot. Nothing's stuck yet and you're protecting a deadline › Protect the Critical Path. Too much work is entering to begin with › Execution Queue Control.

**The one test before you start:** is it already stuck, or are you protecting it from getting stuck? This is for work that has already stalled — "what is blocking us right now?" If nothing's stuck, there's nothing to trace.

· WHAT WILL THE TEAM LEAVE WITH?

## What you'll leave with

### WHAT YOU GET

A map of what's waiting on what, the source blocker at the far end of the chain circled, and one owned move aimed at it — not the visible task that was never the blocker.

### WHY IT MATTERS

The team sees the blocker is rarely the task it was focused on — often nowhere near it. Effort stops going into the wrong thing, and the work starts moving once the real source is cleared.

© Witmer

# PART 1 BEFORE THE ACTIVITY

get ready

1 · WHAT IS MY JOB HERE?

## Your job

You are running this to one outcome: the source blocker circled, with one owned move aimed at it. Your job is to:

- push the team to **trace to the far end** — the first "waiting on" is almost never the real blocker
- run the **source tests** on each node before stopping — don't rely on judgement
- keep it **light** — nodes, arrows, one circle; no scoring or symbols
- aim the one move at the **source, not the visible task** — and keep it the smallest action that clears it

THE ONE IDEA

**Fix the source, not the symptom.** The whole tool turns on tracing the "waiting on" chain to the node that waits on nothing further. Stop one link in and you fix a symptom; trace to the far end and you find the real blocker. Drawing the chain makes the source visible, so effort goes where it actually unblocks the work.

2 · WHO DOES WHAT?

## Roles

| Role | What they do |
|---|---|
| You (facilitator) | Run the flow, keep the team tracing to the far end (not stopping at the first link), run the source tests, and capture the circled source and the one owned move. You find the blocker; you don't solve it in the room. |
| Participants (a team) | The people who know the work. They name the dependencies, trace the chain, and agree the one move. 15–25 minutes. |

3 · HOW DO I OPEN?

## Opening script — 2–3 minutes

In your own words, never read aloud. Don't assume anyone read the app page.

SAY — IN YOUR OWN WORDS

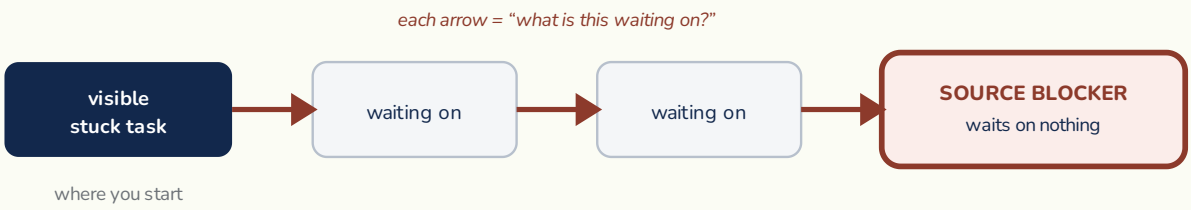
"Something's stuck, and we keep pushing on the thing in front of us without it helping. We're going to take the stuck work and trace what it's actually waiting on — and what that's waiting on — until we hit the thing at the far end that's waiting on nothing. That's usually the real blocker, and it's often nowhere near where we've been putting our effort. Then we pick one move aimed at it."

© Witmer

4 · HOW DO I PREPARE THE SPACE?

## Setup and materials

THE “WAITING ON” CHAIN — TRACE TO THE FAR END




**The thing that’s stuck usually isn’t the thing that’s blocking you.**

*The real blocker sits at the far end — often nowhere near where you started.*

Branches? If a node waits on two things, make two notes and follow each until it ends.

| Component | What it is, and when you use it |
|---|---|
| Your stuck work | The thing that has visibly stalled. If nothing is actually stuck, there’s nothing to trace. The first node, on the left. |
| A board or wall | Room to put up nodes and draw “waiting on” arrows between them, left to right. |
| Sticky notes + marker | One note per item — the stuck work and each thing it waits on. One dependency per note. The marker draws the arrows and circles the source. |

**One dependency per note.** If a node is waiting on two things, make two notes and trace each — a note that says “waiting on QA and Product” hides a branch and weakens the chain.

## PART 2 DURING THE ACTIVITY

run it

**Before you begin — 30-second check**

- ✓ There’s one real piece of stalled work to trace — not a hypothetical
- ✓ The people who know the dependencies are in the room
- ✓ The board’s ready to run left-to-right, one note per dependency
- ✓ You’re ready to hold the team to the far end — and run the source tests

5 · HOW DO I RUN IT?

## How to run it

Brief the team (2–3 min), then run these in order. The bold part is what you say or do; the note under it is where you add the most.

© Witmer

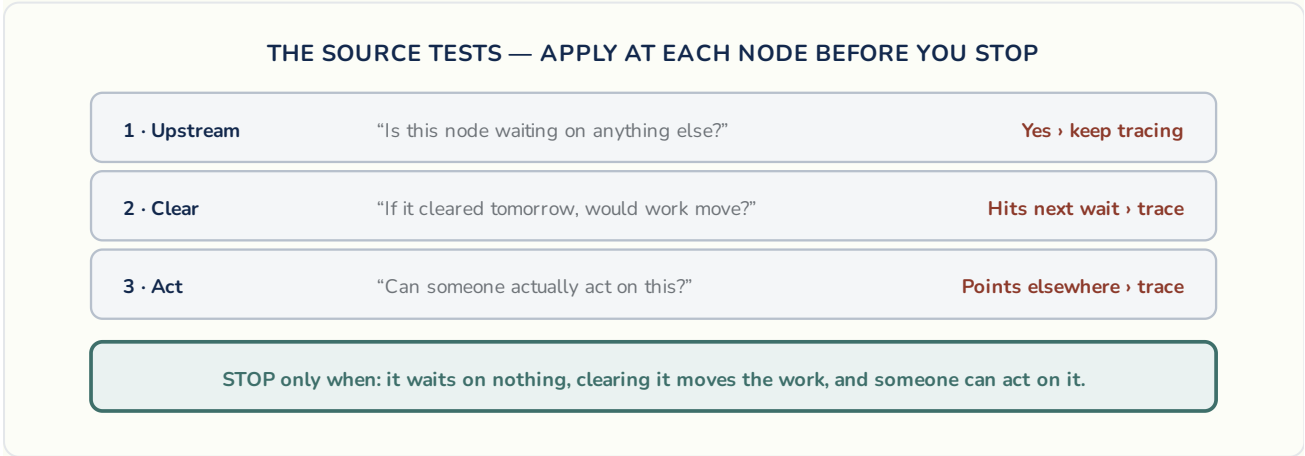
| | | |
|---|---|---|
| **1. Put up the stuck work** | **2–3 min** | *"What's visibly stopped moving?"* Write it on a note and put it up as the first node, on the left. |
| **2. Ask what it's waiting on** | **3–4 min** | *"What has to happen before this can move?"* Write that as a new node and draw an arrow to it. One dependency per note — if it's waiting on two things, make two notes. |
| **3. Keep tracing** | **4–6 min** | *"And what is that waiting on?"* Ask it again of each new node. Don't stop at the first dependency. If a node waits on several things, follow each branch separately until it ends. Run the source tests below before you stop. |
| **4. Find where the chain ends** | **3–4 min** | *"Does this wait on anything else?"* The node with no further dependency — waiting on nothing else — is the source blocker. It's often nowhere near where you started. Circle it, and let that distance land. |
| **5. Make one move at the source** | **3–4 min** | *"What's the smallest action that would clear the circled source?"* Aim it at the source, not the visible task. Give it an owner and a rough when. Photograph the board. |

**These times are indicative** and will depend on the nature of the problem and the size of the team; the whole activity runs 15–25 minutes. This is a five-minute trace, not a workshop. Step 3 (trace to the far end) is where the value is — don't let the team stop one link in. Step 5 (one move at the source) is what gets skipped; protect it. A found blocker with no move is just a diagram.

6 · HOW DO I KNOW WHEN TO STOP TRACING?

## The source tests

The first "waiting on" is almost never the real blocker. Don't rely on judgement — run these on each node before you stop.




© Witmer

**The stop rule:** you've found the source only when a node waits on nothing further, clearing it actually moves the work, and someone can act on it. If any test says "keep tracing", draw another arrow. Two chains converging on one node is a strong signal you're there. And don't stop because you've hit a difficult dependency — stop only when there's nothing else it's waiting on.

**When to step in — and when to stay out** · a principle, not a step

STEP IN

They stop at the first "waiting on" → push: what's that waiting on?

They start scoring or ranking links → it's just arrows and a circle.

The move aims at the visible task, not the source → point it at the source.

The chain sprawls → pick the one path to the real hold-up.

STAY OUT

The map looks messy → fine, if it traces to a source.

Someone's surprised the blocker isn't their task → let it land.

They debate which item to start from → any stuck one works.

The trace is quick → quick is the point, keep it short.

**The clearest tell it's time to step in:** the team circles the first thing the stuck task is waiting on and calls it the blocker. That's usually a symptom. Push: "and what is that waiting on?" Keep going until the arrows stop.

## PART 3 AFTER THE ACTIVITY

make it stick

7 · HOW DO I CLOSE?

### Closing script — 2 minutes

Reconstruct the discovery — don't summarise it. Replay where the team started so the distance to the source lands.

SAY — IN YOUR OWN WORDS

"We walked in pushing on *[the visible task]*. But trace the chain and it was waiting on *[link]*, which was waiting on *[the source]* — the thing that's waiting on nothing, that nobody was chasing. That's the real blocker, and it's *[N]* links from where we've been putting our effort. Our one move is aimed there — not at the task that was never the blocker."

© Witmer

8 · DID WE GET A GOOD RESULT?

## What good looks like

### A GOOD RUN

The chain traces to a single circled source, and the one move is aimed at that. The team can say: “we’d been pushing the wrong thing all week.”

### A HOLLOW RUN

They circle the first “waiting on” and call it done, or aim the move back at the visible task. If so, you stopped tracing too early — run the source tests and keep going.

9 · WHY DOES THIS WORK?

## The idea underneath

When work stalls, the chain of who-waits-on-whom is invisible — so the team pushes on the most visible stuck task, which is usually below the real cause in the chain. Drawing the chain makes the source visible, so effort goes where it actually unblocks the work. The value isn’t the map; it’s the distance the map reveals between the task you were focused on and the thing actually holding it up.




**Before you finish — the check**

- ✓ The chain traces to a single node that waits on nothing further
- ✓ The source passed the tests: nothing upstream, clearing it moves the work, someone can act
- ✓ The source blocker is circled — and the team saw how far it was from the visible task
- ✓ One move is aimed at the source, not the visible task — the smallest action that clears it
- ✓ The move has a named owner and a rough when
- ✓ The board has been photographed before anyone leaves

© Witmer

10 · HOW DO I MAKE THIS A HABIT?

## Take it back to work

**THE HABIT**

When something's stuck: "What is this actually waiting on — and what is that waiting on?"

The lasting change: the team stops pushing harder on the visible stuck task and starts tracing the chain to the source. Fix the source, not the symptom.

**After you clear the source:** clearing one blocker often reveals the next — the work moves a step, then waits on something new. That's normal, not a failed run. Keep the photographed map as a record and re-trace it as the blocker moves; the chain you drew makes the next snapshot faster.

## Appendix · Reference

You should not need this while you run the activity. Read it before, or use it if a run goes sideways.

11 · A WORKED RUN

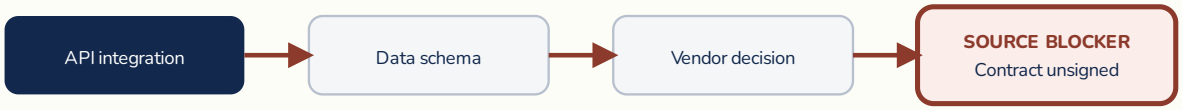
## Watch the facilitator move

A team's "API integration" had been stuck for a week; everyone was pushing hard on the API itself. The moment that found the real blocker:

**THE TRACE**

Team: "The API integration is stuck — we need more hands on it."
Facilitator: "Before that — what is the API waiting on?"
Team: "...the data schema, actually. And the schema's waiting on a vendor decision."
Facilitator: "And the vendor decision — is it waiting on anything?"
Team: "...the contract. It's unsigned. Nobody's chasing it."

FINISHED DEPENDENCY MAP — THREE LINKS FROM THE VISIBLE TASK




**THE MOVE**

Escalate the contract to procurement today — signed by Friday. Owner: team lead. Not "push harder on the API".

*One source circled, one move, one owner — aimed at the far end, not the visible task.*

**What the facilitator did:** (circles "contract unsigned") "That's the source — three links from the API. The move is to chase the contract, not add hands to the API." They didn't argue about the API; they ran the trace and let it lead to the far end. The team had been pushing on a symptom for a week.

© Witmer

12 · WHAT GOES WRONG MOST OFTEN?

## Common mistakes

| Mistake | Why it fails the run |
|---|---|
| **Stopping at the first “waiting on”** | That’s almost always a symptom. Run the source tests — keep tracing until a node waits on nothing. |
| **Aiming the move at the visible task** | The task was never the blocker. Point the move at the circled source. |
| **Two dependencies on one note** | Hides a branch. One dependency per note, and follow each branch to its end. |
| **Scoring or ranking links** | This is nodes, arrows and a circle — no notation. Keep it light and fast. |
| **Mapping the whole system** | Pick the single most painful stuck task and trace just its chain, not the org. |

13 · WHAT IF SOMETHING GOES WRONG?

## Edge cases

| What happens | What to do |
|---|---|
| The chain loops back on itself | A genuine circular dependency (A waits on B waits on A). The loop itself is the blocker — you break it with a decision, not more tracing: pick the one link someone can act on now, and commit to it out of sequence. Usually it’s a call that’s been deferred because each side was waiting for the other. |
| Two chains point to the same node | Strong signal you’ve found the source — that shared node is almost certainly the real blocker. Circle it. |
| The source is outside the team | Common, and useful to see. The move becomes “escalate to / chase whoever owns it”, with an owner on your side to drive it. |
| The team is split on what a node waits on | Don’t debate it abstractly — ask the person who does that work. If still unclear, trace both branches briefly; the real one usually leads somewhere, the other dead-ends. |
| Everything seems to wait on everything | Too many items at once. Pick the single most painful stuck task and trace just its chain. |
| The first “waiting on” really is the blocker | Possible — but make the team prove it with the source tests, not assume it. If it waits on nothing and someone can act, fine. |

© Witmer

14 · WHERE DOES THIS FIT?

## Where this sits in the Gym

| If the question is... | Use this |
|---|---|
| “What are we waiting for?” (work already stuck) | **Dependency Snapshot** — this tool. Trace the stuck work to its source blocker. |
| “Where is time disappearing?” (in one piece of work) | **Constraint Snapshot** — find the stall point where work sits waiting. |
| “What must not be interrupted?” (before the cycle) | **Protect the Critical Path** — protect the pacing work before it’s at risk. |
| “Too much work is entering to begin with” | **Execution Queue Control** — manage what gets in, before it ever stalls. |

**Rule of thumb:** if work has already stalled, you’re finding a blocker (this tool). If nothing’s stalled yet and you’re defending a deadline, you’re protecting a path (Protect the Critical Path). Dependency Snapshot asks *why* is this work waiting; Constraint Snapshot asks *where* is work waiting.

© Witmer