



AI Graphs: Stop Making AI Wait for Work That Could Happen at the Same Time



Here's something we're going to discover as we build more AI workflows.

A lot of them are unnecessarily slow.
Not because the AI model is slow.
Not because we need more computing power.
Because we've designed the work badly.

We take a process that looks like this:

- ❑ Research Company A.
- ❑ Then research Company B.
- ❑ Then research Company C.
- ❑ Then compare them.

And we faithfully tell AI to execute those instructions in exactly that order. But think about it.

Why does the research on Company B need to wait for Company A?

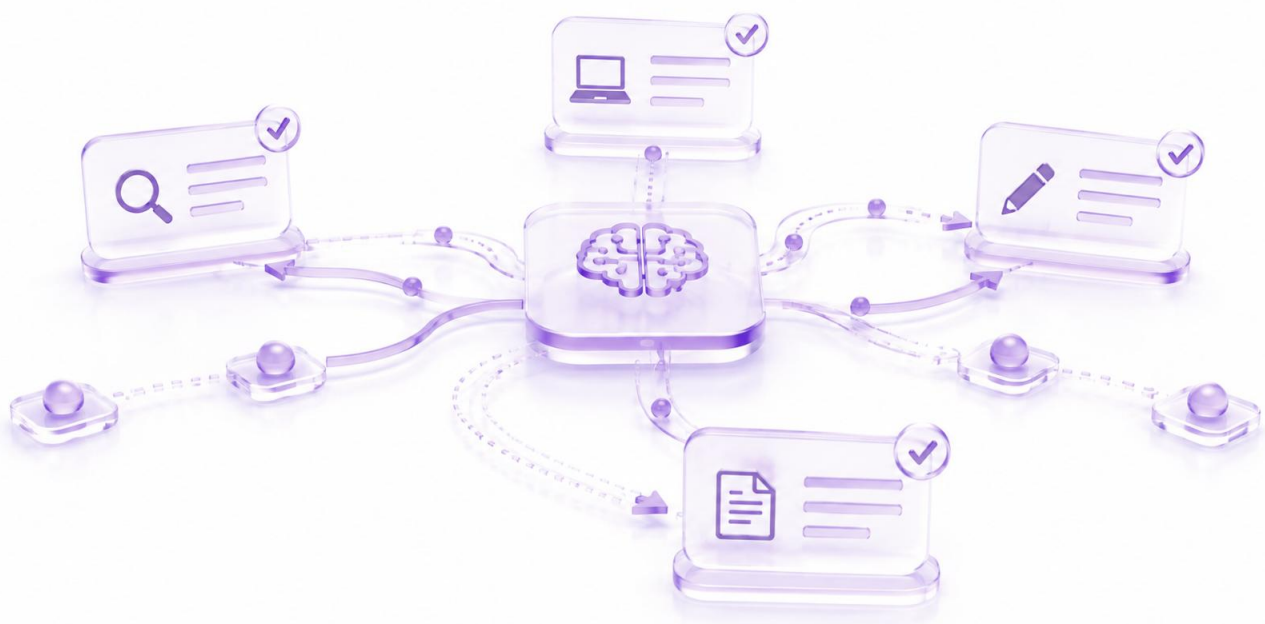
It doesn't. : **Company C doesn't need either of them.**

All three pieces of work could start at the same time.

Only the final comparison needs to wait.

That simple idea is the basis of an AI Graph.

And once you see it, you start seeing unnecessary waiting everywhere.



First, forget the word "graph"

When people hear "graph," they often imagine charts, dashboards or mathematical diagrams.

That's not what we're talking about here.

An AI Graph is simply:

A map of the work that needs to happen and what each piece of work actually depends on.

It contains two basic things.

Nodes

A node is one piece of work.

Edges

An edge means: "This task cannot start until that task produces something it needs."

That's really it.

For example:

Research → Write report

That's a real dependency.

You can't write a meaningful report before you have the research.

But:

Research Company A → Research Company B

probably isn't.

Company B's research doesn't need Company A's results.

We put the arrow there only because that's the order in which we happened to write the tasks down.

And that's the mistake.



Sequence is not dependency

This might be the most important sentence in the whole article.
Sequence is not dependency.

Something happening second doesn't mean it needs the first thing to finish.

Think about how humans run projects.

If three people need to review three separate contracts, we don't normally say:

"Priya, you review yours first."

"Once you're completely finished, Mohammed can start."

"After Mohammed finishes, Anu can begin."

That would be ridiculous.

We give each person a contract and ask them to work simultaneously.

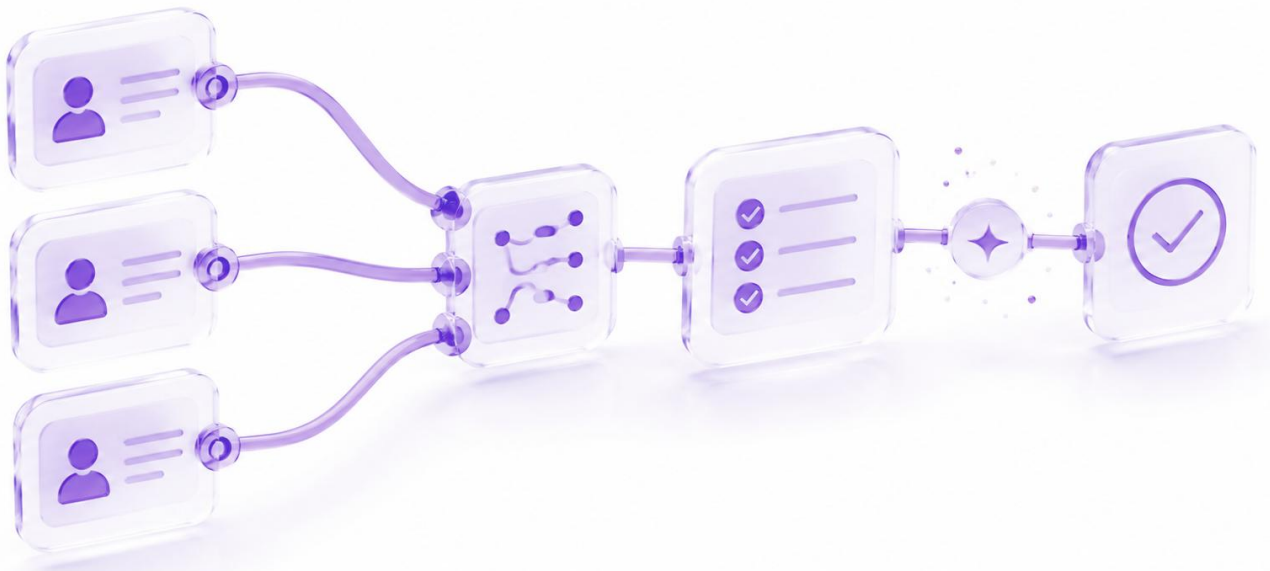
Yet we routinely design AI workflows exactly like that.

- ☐ One task.
- ☐ Wait.
- ☐ Next task.
- ☐ Wait.
- ☐ Next task.
- ☐ Wait.
- ☐ Then combine everything.

AI Graphs force us to ask a better question:

What genuinely needs to wait?

Everything else becomes a candidate for parallel work.



The five-minute "fake edge" test

There's a very simple exercise.

Take any business workflow.

Write every task in a box.

Connect them in the order in which they happen today.

Then examine every arrow.

Ask one question:

Does the output from this task actually become an input to the next task?

If yes, keep the arrow.

That's a real dependency.

If no, delete it.

That's a fake edge.

And once you delete fake edges, something interesting happens.

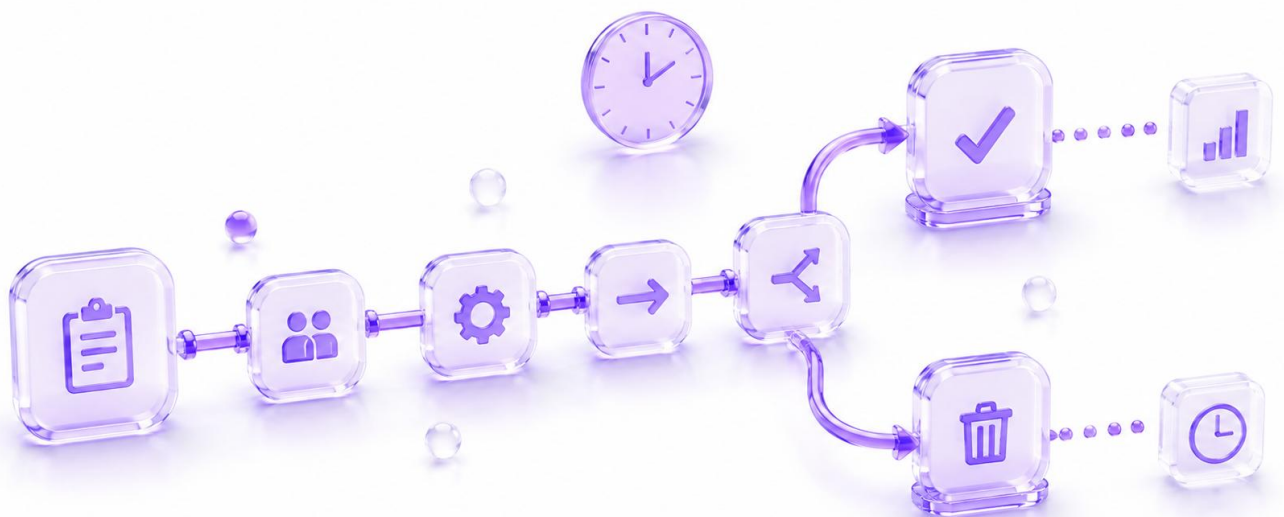
Tasks that looked sequential suddenly become independent.

Independent work can happen simultaneously.

This doesn't just apply to AI.

It's a useful process-design technique generally.

AI simply makes the benefit much more obvious because digital workers don't have the same scheduling constraints as humans.



Consider competitive research

Suppose you want a comparison of three companies.

Let's call them A, B and C.

You want:

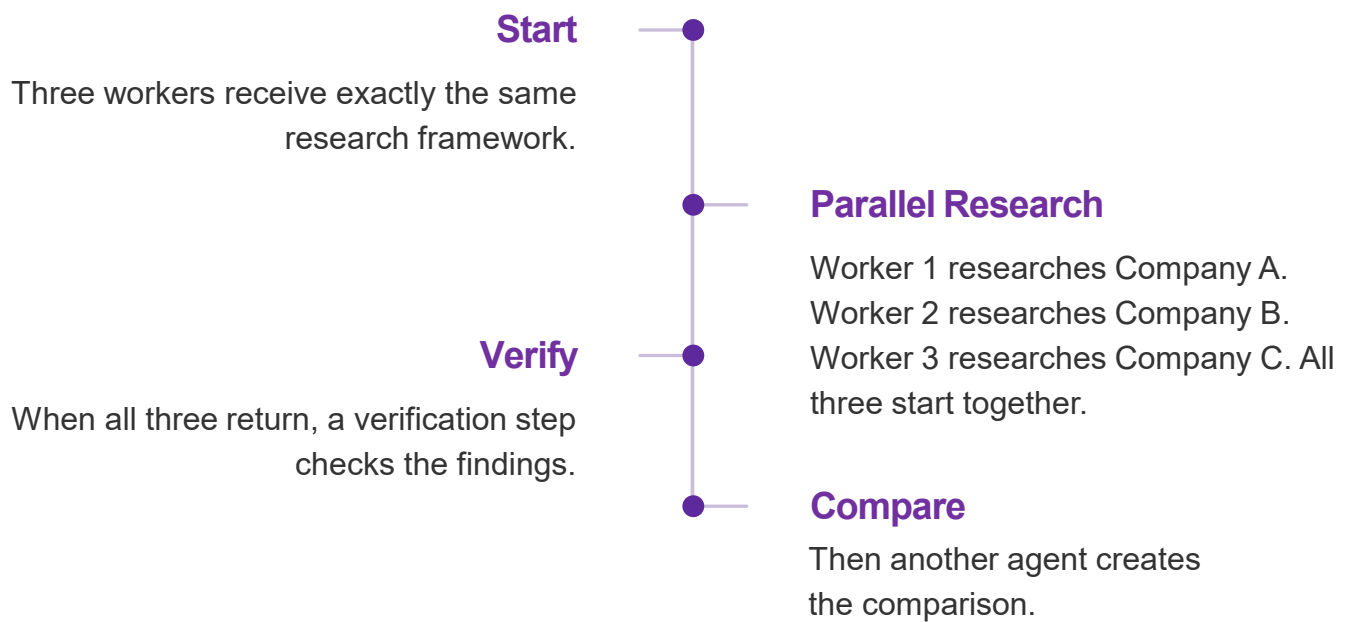
- ☐ Current pricing
- ☐ Product features
- ☐ Recent developments
- ☐ Customer complaints
- ☐ Market positioning

A linear workflow might look like this:

Research A. Then research B. Then research C. Then compare them.

Suppose each research task takes five minutes. The research phase takes roughly fifteen minutes before the comparison even begins.

Now redraw it.



If the slowest researcher takes six minutes, our research stage now takes around six minutes instead of fifteen.

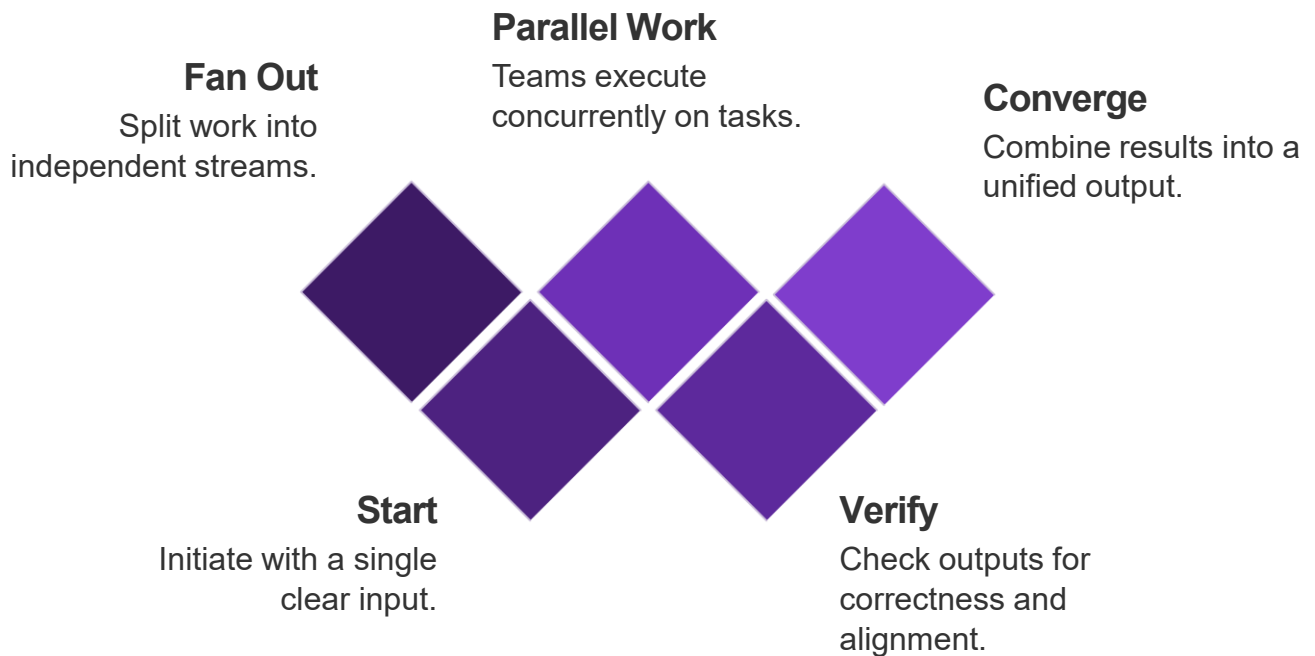
The work didn't disappear. The waiting disappeared. That's the point.

This creates a very common shape

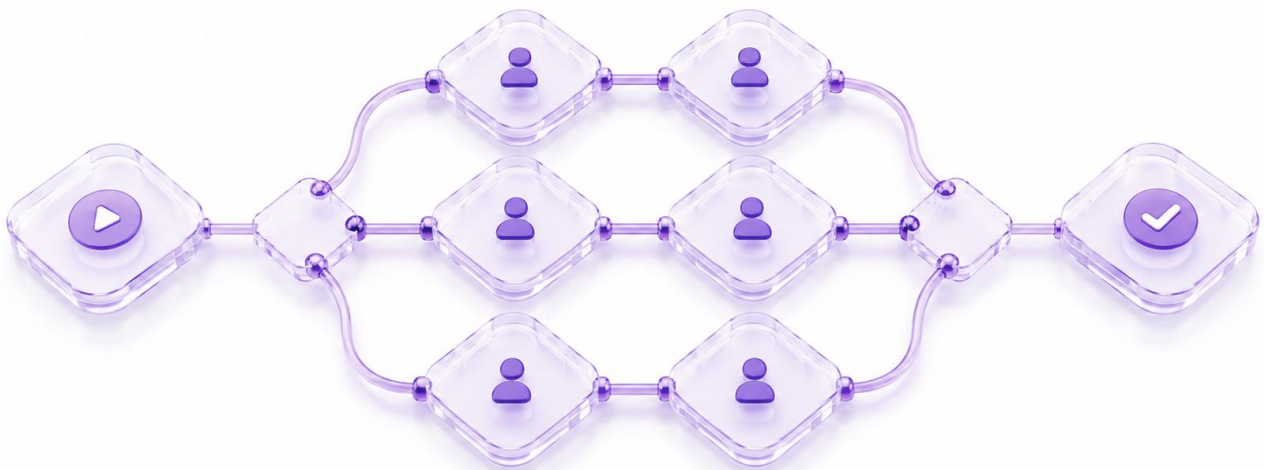
When you remove fake dependencies, many workflows start looking like a diamond.

- ❑ One starting point.
- ❑ Work fans out.
- ❑ Several independent workers execute simultaneously.
- ❑ Their outputs converge.
- ❑ Then something combines them into the final result.

Conceptually:



Let's call this the Diamond. It's a surprisingly useful pattern.



Take software code review

Imagine we're reviewing three files:

- ☐ `auth.py`
- ☐ `api.py`
- ☐ `db.py`

We could ask one agent: "Review `auth.py`. Then `api.py`. Then `db.py`."

But unless the reviews depend on each other, why wait?

Worker 1

Reviews authentication.

Worker 2

Reviews the API.

Worker 3

Reviews the database layer.

They run simultaneously.

Then a checker looks across all three findings.

That's important because some problems only become visible when you look across files.

For example:

The authentication reviewer might say an access token is correctly generated. The API reviewer might say requests are correctly processed. But the combined review might discover that one API endpoint doesn't actually validate the token.

The individual reviews happen in parallel. The cross-file reasoning waits because it genuinely needs all three results. That's a real edge.

Then a final step produces a prioritized fix list:

- ☐ **Critical.**
- ☐ **High.**
- ☐ **Medium.**
- ☐ **Low.**

That's a Graph.



Finance has plenty of examples too

Consider investigating an unusual transaction.

Before making a decision, we might need to know:

- ✓ **Data Check**
What does the transaction data say?
- ✓ **Policy Check**
What does company policy say?
- ✓ **Customer History Check**
What happened with this customer previously?
- ✓ **Approval Check**
Is there a related approval?
- ✓ **Control Threshold**
Does the value exceed a control threshold?

There is no reason for an agent to check the transaction first, wait, check the policy next, wait, and then retrieve history.

Those checks may be completely independent.

So run them together.

Then send their outputs to a decision agent.

Now the decision agent has everything it needs.

The Graph doesn't make the decision itself. It organizes how the evidence reaches the decision. That distinction matters.

Nodes need to stay small

There's another useful principle here.

A node should usually represent one clear job.

Bad node:

"Research the competitor, analyse the results, check the sources, compare it with us and prepare a management report."

That's practically an entire workflow hidden inside one box.

Better:

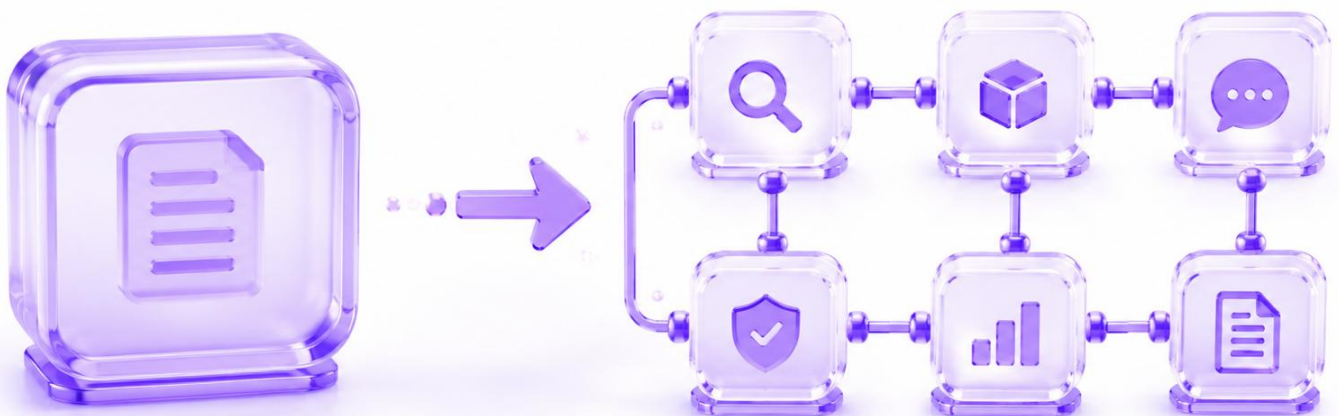
- Node 1: Research pricing.
- Node 2: Research products.
- Node 3: Analyse customer reviews.
- Node 4: Verify claims.
- Node 5: Compare findings.
- Node 6: Prepare report.

Why does this matter?

Because smaller nodes make dependencies visible.

If everything is hidden inside one giant instruction, we can't easily see what could run in parallel.

We also can't easily see where something failed.



Inputs and outputs matter

Every node should have a reasonably defined contract.

What goes in? What should come out?

For example:

Job: Research competitor pricing.

Input: Company name and website.

Output:

- ☐ Product
- ☐ Plan
- ☐ Price
- ☐ Currency
- ☐ Billing frequency
- ☐ Source
- ☐ Date checked

Now the next worker knows exactly what it is receiving.

Compare that with a node that returns:

"Here's what I found about the competitor..."

followed by twelve paragraphs of prose.

A person can read that.

Another AI can also read it, of course.

But defined outputs make automation much easier to control, validate and audit.

This becomes increasingly important as Graphs grow.



Then comes the checker

This is the part we would be reluctant to remove.

Imagine three research workers return their findings.

Should the same workers also decide whether their findings are correct?

We wouldn't love that design.

If an agent misunderstood something while creating an answer, asking it to check its own reasoning using exactly the same context can reproduce the same blind spot.

Let's rather introduce another node.

A verifier.

Its job isn't:

"Make this better."

Its job is:

"Try to find what is wrong."

That's a different mindset.

For research, the verifier might ask:

- ☐ Is the claim correct?
- ☐ Is it current?
- ☐ Does the source actually exist?
- ☐ Does the source really support the claim?
- ☐ Can it be contradicted by a stronger source?

Ideally, the verifier receives the finding and the evidence it needs, not a long explanation of how the original worker convinced itself.

Fresh eyes matter. They matter with people. They matter with AI too.

Parallel verification is possible as well

We can take this further.

Suppose we need to validate one important finding.

Instead of one checker asking everything sequentially, we could run three independent checks:



Checker 1

Is the claim factually supported?



Checker 2

Is the information current?



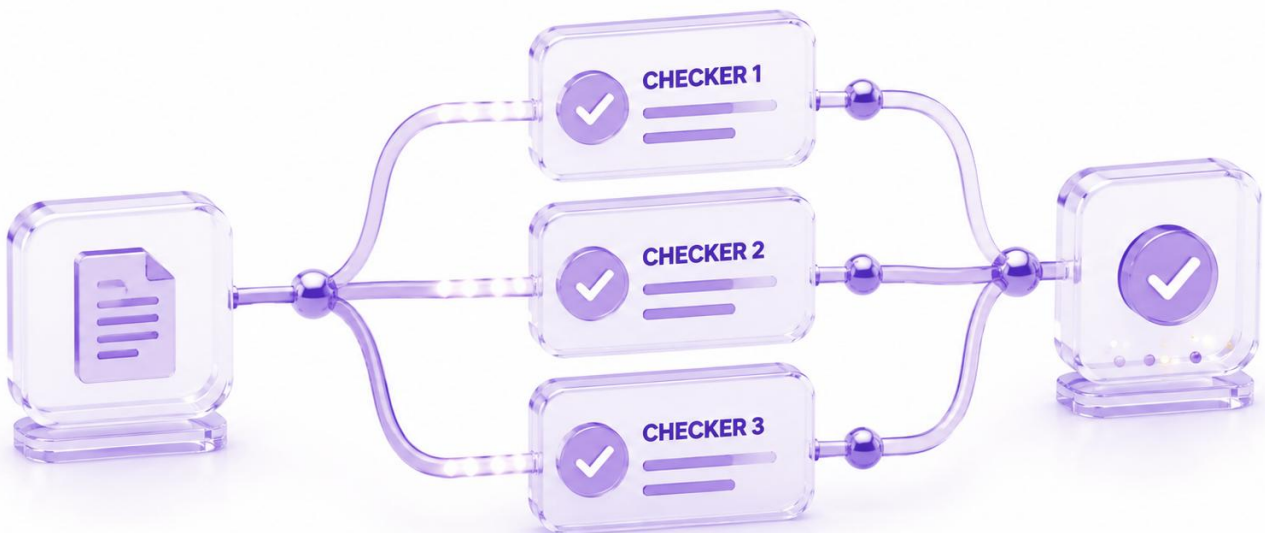
Checker 3

Is the source genuine and relevant?

Three checks. Run simultaneously. Then combine the results.

We're now building a Graph inside the verification process itself.

That's the interesting thing about these patterns. They can be nested.



But parallel isn't automatically better

Here's where we need a little restraint.

If three tasks genuinely depend on each other, trying to run them simultaneously doesn't make us clever.

It makes the workflow wrong.

Suppose we have:



- ☐ **The risk analysis genuinely needs the extracted clauses.**
- ☐ **Negotiation recommendations genuinely need the risk analysis.**

Those are real dependencies.

The correct workflow may be sequential.

There's nothing wrong with that.

The goal isn't maximum parallelism.

The goal is: Don't make something wait unless it genuinely needs to wait.

That's a much healthier design principle.

Parallel workers also have a cost

There's another reality.

Three AI workers running simultaneously may reduce elapsed time.

They don't necessarily reduce computing cost.

In some cases, they increase it.

If we split a task into ten workers when three would have been enough, we may simply create more processing, more duplicated research and more outputs to reconcile.

Faster doesn't automatically mean cheaper.

And more agents don't automatically mean better work.

The business case depends on the value of time saved, quality gained and human effort removed.

If parallel execution saves fifteen minutes in an annual task, who cares?

If it cuts a critical customer-response process from 40 minutes to five minutes across 20,000 cases, now we have something worth investigating.

Context matters.



A healthcare example

Think about caregiver matching in home healthcare.

We need to find an appropriate caregiver for a patient.

Potential considerations include:



Qualifications

Clinical skills. Care level.



Availability & Location

Availability. Distance and travel. Shift duration.



Patient Preferences

Language. Gender preference. Previous patient relationship.



Commercial Rules

Commercial constraints.

Some of those checks can happen independently.

One worker checks qualifications. Another checks availability. Another calculates distance and travel. Another reviews continuity and preference. Another checks commercial rules.

Their findings converge into a matching decision.

But suppose availability fails completely. Do we need to perform every downstream calculation? Maybe not.

The Graph can contain conditional branches too. That's when things become particularly useful.

A Graph isn't just parallelism. It's a way of explicitly describing how work moves.

Think about customer service

A customer writes:

"Why hasn't my order arrived, and can you refund the delivery charge?"

Several things may need checking:



Order status.



Courier tracking.



Promised delivery date.



Customer service policy.



Customer history.



Refund authority.

Why retrieve all those one after another?

If they're independent, start them together.

Then a decision node receives:

- ☐ Courier says delayed.
- ☐ Delivery SLA was breached by two days.
- ☐ Policy permits refund up to ₹500.
- ☐ Customer has no abuse flags.
- ☐ Refund amount is ₹150.

Now the decision is straightforward.

The slow part wasn't necessarily reasoning. It was gathering context. Graphs can dramatically improve that part of the workflow.

This is why **Agent**, **Loop** and **Graph** fit together

These are three related ideas.

Agent = Autonomy

Give AI a goal and let it work toward the outcome using defined tools.

Loop = Reliability

Don't accept the first output automatically. Check it, learn from failure and try again until it passes or reaches a hard stop.

Graph = Speed + Coordination

Break the work into nodes, identify real dependencies, run independent tasks simultaneously and bring everything back together.

Put the three together and the operating pattern starts to look very different from today's chat experience.

The user doesn't manage every prompt.

The Agent manages the goal.

The Loop manages quality.

The Graph manages the work.



This is closer to how real teams operate

Think about a good human team.

The CEO doesn't give one person every task in sequence.

- ☐ Research waits for research.
- ☐ But finance doesn't necessarily wait for marketing.
- ☐ Legal can review one issue while technology investigates another.
- ☐ People work in parallel where possible.
- ☐ They synchronize where necessary.
- ☐ Specialists review specialist work.
- ☐ Important decisions get independent checks.
- ☐ Outputs converge at defined points.

Why shouldn't digital workers operate the same way?

In fact, AI gives us an opportunity to rethink workflows that grew sequentially only because that's how our systems or organizations happened to evolve.



Start with a workflow you already know

You don't need complicated software to begin thinking this way.

Take one workflow from your business.

Write down every step.

Look at everything with no dependencies.

Those are your parallel candidates.

Draw arrows between them.

Then ask about every arrow: Does the next task genuinely need the previous task's output? If yes, keep it. If no, remove it.

Then ask:

Where should the outputs converge? What needs checking? Which decisions require humans? Which steps could be agents?

You'll probably learn something about the business process before you write a single line of AI code.

And honestly, that's often where the biggest value is.



One formula worth remembering

If Agents gave us:

Agent = Goal + Tools + Memory + Action + Check

And Loops gave us:

Loop = Work + Check + Memory + Stop Condition

Then the simplest way to remember Graphs is:

Graph = Nodes + Real Dependencies + Parallel Work + Convergence

Or even more simply:



**Nodes define
the work.**



**Edges define what
must wait.**



**Everything else is
execution.**

So the next time someone shows you a ten-step AI workflow, don't immediately ask which model they're using.

Ask something more useful:

Why are these ten steps running one after another?

You may discover that half the arrows aren't real.

Remove those fake edges.

Run the independent work together.

Wait only where you have to.

That's the real idea behind AI Graphs.

Not making AI work harder.

Making the work flow better.





Let's build what's next.

Partner with us to accelerate
transformation and deliver
measurable impact.



www.10xDS.com



www.10xDS.ai



Connect@10xDS.com