quad, one example at a time

quad draws flow diagrams. You write the boxes, the arrows, and what should line up with what. quad works out where everything goes and sets it on a grid, so the boxes come out one size, the gaps come out even, and the same file always draws the same picture. There are no coordinates to place by hand and nothing to nudge afterwards.

It is for the diagrams you keep: an architecture, a pipeline, a state machine, anything that lives in a repository beside the thing it describes and has to be legible after the tenth edit. It is not a drawing program, and it will not lay out an arbitrary graph for you.

136 examples follow, from a single box to a whole pipeline. Read them in order: each one adds one thing to the one before it, and nothing is used before the page that shows it. Every picture was drawn from the source printed with it, so what you see on a page is what that source gives you.

CONTENTS

One finished diagram, so you know what you are working towards. Nothing on this page needs explaining yet — every statement in it gets its own page later, in order, starting with a single box.

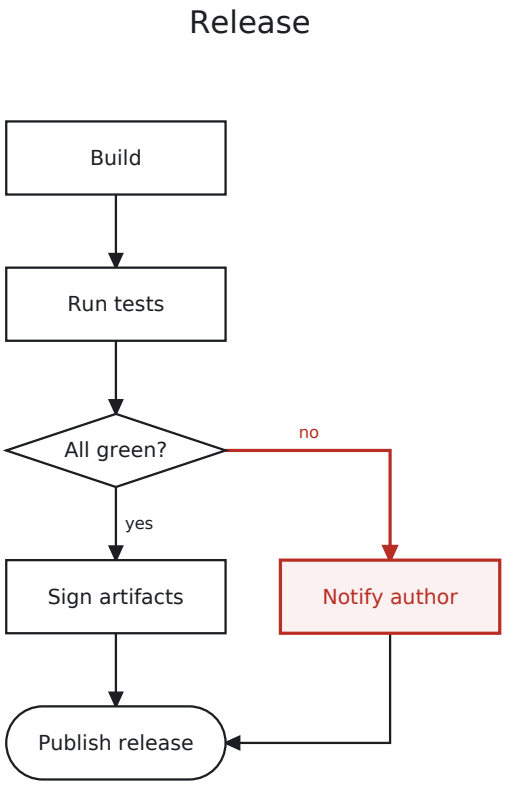
001 A finished diagram

Every line of this is explained later, one page at a time. Read it now for the shape of a quad file: what the boxes are, what points at what, and what should line up.

SOURCE

```
diagram "Release" { flow down }

kind stage box spans [1, 2]

stage  build  "Build"
stage  test   "Run tests"
choice green  "All green?"
stage  sign   "Sign artifacts"
stage  notify "Notify author"
end    ship   "Publish release"

build -> test -> green
green -> sign : yes
green -> notify : no
sign -> ship
notify -> ship

sameline build test green sign ship
order green: sign notify
role error { notify, green -> notify }
```

DIAGRAM



A diagram is made of boxes. This part draws one, gives it words, and shows every shape a box can take and what each one is for.

002 One box

Write a name on a line of its own and you get a box with that name in it.

SOURCE

```
alpha
```

DIAGRAM

003 A box that says something else

Put the words you want drawn in quotes. The bare name in front of them is how you will refer to this box later on.

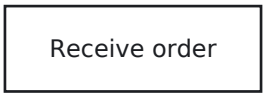
box says what kind of shape to draw. It is the only kind so far.

SEE ALSO All six shapes, page 7

SOURCE

```
box alpha "Receive order"
```

DIAGRAM



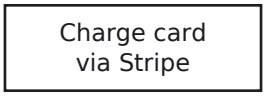
004 Two lines of text

Two quoted strings make two lines. You choose where the break falls; quad never moves it.

SOURCE

```
box charge "Charge card" "via Stripe"
```

DIAGRAM



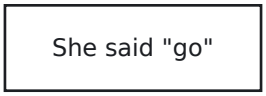
005 A quote mark in the text

To put a quote mark in the words, write it as \" so it is not read as the end of them.

SOURCE

```
box quoted "She said \"go\""
```

DIAGRAM



006 box

box draws a rectangle. A name on its own gives you the same box; this writes it out.

SOURCE

```
box step "Step"
```

DIAGRAM

007 choice

choice draws a diamond, for a question the diagram answers two ways.

A diamond has room for about half the words a rectangle does, so keep the question short. Too long and quad refuses to draw it rather than letting the words spill out.

SOURCE

```
choice valid "Valid?"
```

DIAGRAM

008 end

end draws a rounded box, for where something starts or finishes.

SOURCE

```
end done "Done"
```

DIAGRAM

009 io

io draws a slanted box, for something going in or coming out: a form filled in, a file written, a message sent.

SOURCE

```
io seed "Seed URL"
```

DIAGRAM

010 mark

mark draws a triangle, for pointing at something rather than doing something.

SOURCE

```
mark here "Here"
```

DIAGRAM

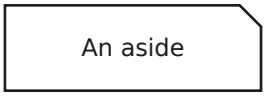
011 note

note draws a box with a folded corner, for a remark that is not a step.

SOURCE

```
note aside "An aside"
```

DIAGRAM



012 All six shapes

The six shapes side by side. Each takes up the same room, so a diamond and a rectangle in one diagram still line up with each other.

SOURCE

```
box    one   "box"
choice two   "choice"
end    three "end"
io     four  "io"
mark   five  "mark"
note   six   "note"
```

DIAGRAM

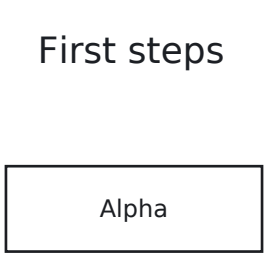
013 Giving the diagram a title

diagram names the whole drawing. The name is written larger than the words in a box, in a band of its own at the head of the page. Put this line at the top; you can leave it out.

SOURCE

```
diagram "First steps"

box alpha "Alpha"
```

DIAGRAM



Two boxes and an arrow between them is the whole of most diagrams. This part draws that, turns it in every direction, and shows the statements that say what should line up.

014 Two boxes

quad arranges boxes in steps, and boxes in the same step stand side by side. Nothing here says one comes after the other, so both are in the first step.

SOURCE

```
box alpha "Alpha"
box beta  "Beta"
```

DIAGRAM

015 An arrow between them

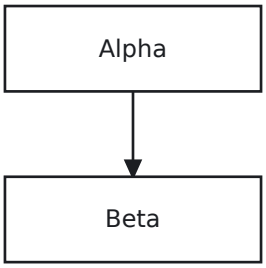
-> says beta comes after alpha, so beta is no longer in the same step but in the next one. Each step is drawn below the one before it, which is why the two boxes are now stacked instead of side by side.

Arrows are what sort the boxes into steps. Every other statement in quad arranges boxes the arrows have already sorted.

SOURCE

```
box alpha "Alpha"
box beta  "Beta"
alpha -> beta
```

DIAGRAM



016 Boxes you never declared

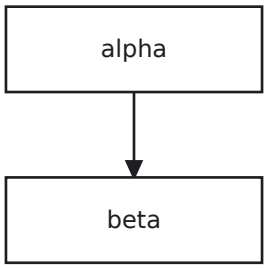
A name used in an arrow becomes a box on its own, so a whole diagram can be a list of arrows.

Write the **box** line only to give that box a different shape or different words.

SOURCE

```
alpha -> beta
```

DIAGRAM



017 The direction the diagram runs

Every diagram so far has run down the page, and **flow down** says so. Change that one word and the whole diagram turns.

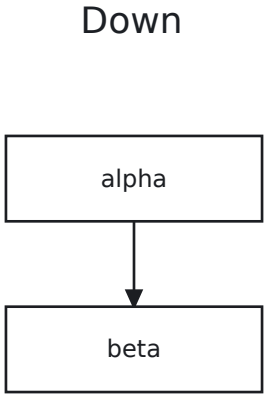
flow goes inside the **diagram** block and takes **down**, **right**, **up** or **left**. It is the only place a direction is written, so nothing else in the file has to change.

SEE ALSO flow right, page 11

SOURCE

```
diagram "Down" { flow down }
alpha -> beta
```

DIAGRAM



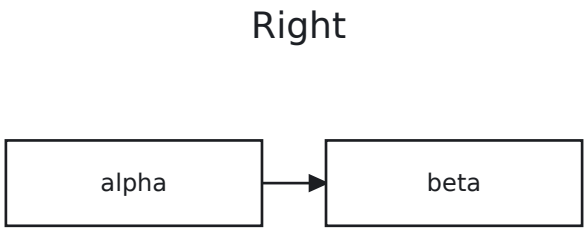
018 flow right

Each box is drawn to the right of the one before it, so the diagram runs across the page.

SOURCE

```
diagram "Right" { flow right }
alpha -> beta
```

DIAGRAM



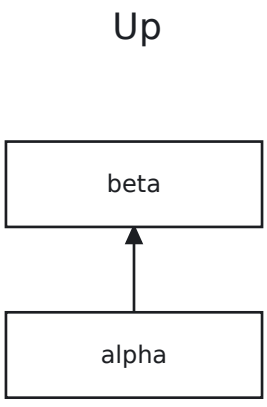
019 flow up

Each box is drawn above the one before it, so the diagram reads from the bottom of the page.

SOURCE

```
diagram "Up" { flow up }
alpha -> beta
```

DIAGRAM



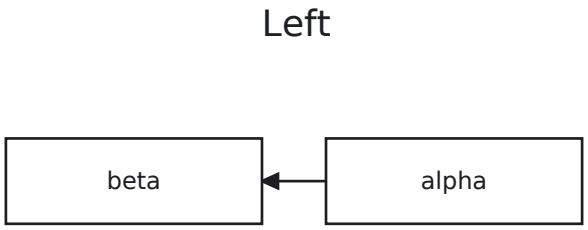
020 flow left

Each box is drawn to the left of the one before it.

SOURCE

```
diagram "Left" { flow left }
alpha -> beta
```

DIAGRAM



021 samestep

samestep puts the boxes it names in the same step, even when the arrows put them in different ones.

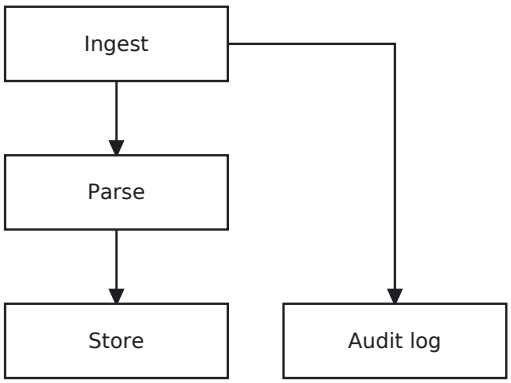
Use it when two things happen at the same stage and no arrow says so. Without it, a box sits one step after whatever points at it.

SOURCE

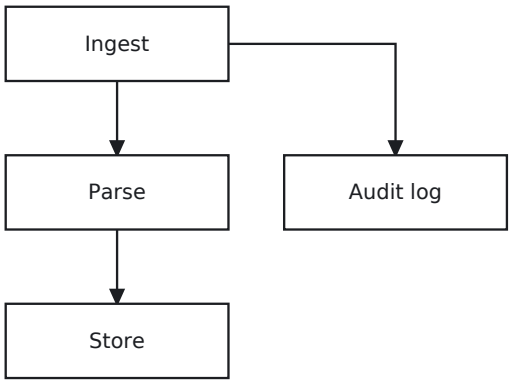
```
box ingest "Ingest"
box parse  "Parse"
box store  "Store"
box audit  "Audit log"
ingest -> parse -> store
ingest -> audit
samestep store audit
```

DIAGRAM

as written



without `samestep store audit`



022 samestep moves what comes after

When **samestep** moves a box to a later step, every box that comes after it moves a step later as well.

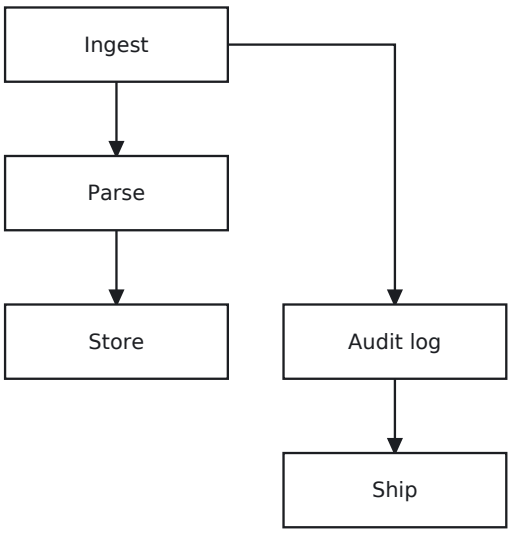
SEE ALSO samestep, page 13

SOURCE

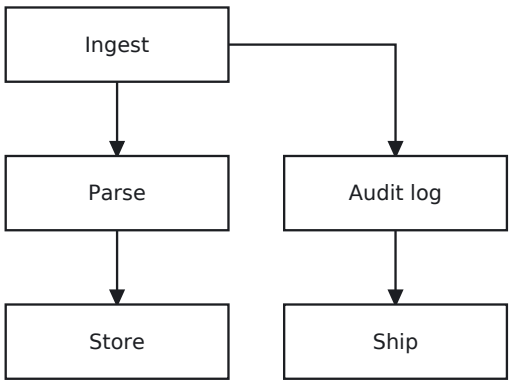
```
box ingest "Ingest"
box parse  "Parse"
box store  "Store"
box audit  "Audit log"
box ship   "Ship"
ingest -> parse -> store
ingest -> audit -> ship
samestep store audit
```

DIAGRAM

as written



without `samestep store audit`



023 sameline

sameline puts the boxes it names on one line in the direction the diagram runs: down the page under **flow down**, across it under **flow right**. The other boxes fill in around them.

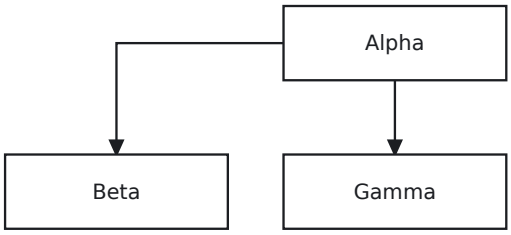
Use it when boxes should line up and no arrow makes that happen. It only lines them up: it says nothing about arrows, and you cannot pick which line they land on.

SOURCE

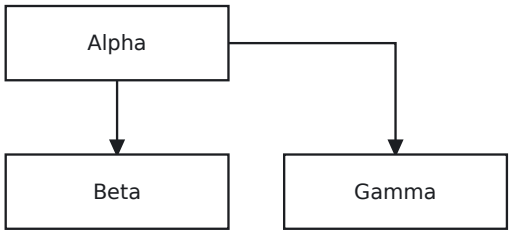
```
box alpha "Alpha"
box beta  "Beta"
box gamma "Gamma"
alpha -> beta
alpha -> gamma
sameline alpha gamma
```

DIAGRAM

as written



without `sameline alpha gamma`



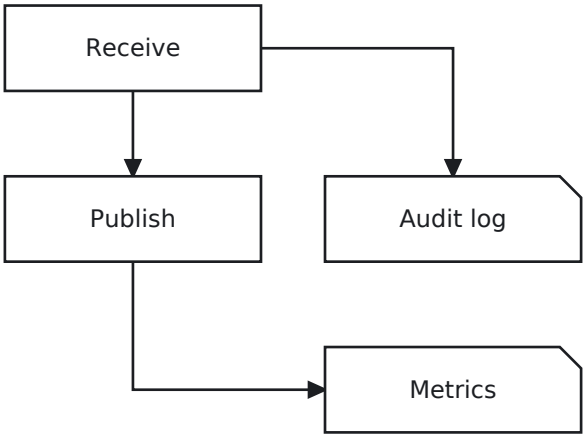
024 sameline, over boxes that never meet

The two boxes on the right come from different steps and have no arrow between them. **sameline** puts them on one line all the same.

SOURCE

```
box receive "Receive"
box publish "Publish"
note audit   "Audit log"
note metrics "Metrics"
receive -> publish
receive -> audit
publish -> metrics
sameline audit metrics
```

DIAGRAM



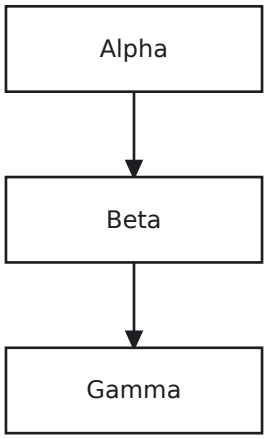
025 Keeping a path straight

Name a whole path with **sameline** and its boxes sit on one line, so the arrows between them come out straight.

SOURCE

```
box alpha "Alpha"
box beta  "Beta"
box gamma "Gamma"
alpha -> beta -> gamma
sameline alpha beta gamma
```

DIAGRAM



026 A straight path with a branch

sameline keeps the boxes it names on one line. Without it the line goes to whichever branch was written first.

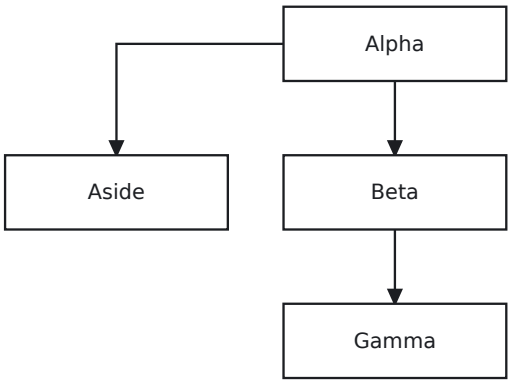
SEE ALSO Keeping a path straight, page 17

SOURCE

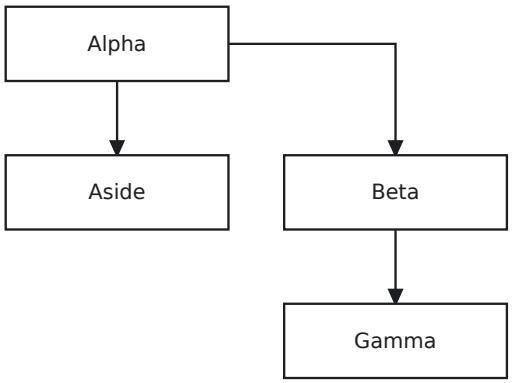
```
box aside "Aside"
box alpha "Alpha"
box beta  "Beta"
box gamma "Gamma"
alpha -> aside
alpha -> beta -> gamma
sameline alpha beta gamma
```

DIAGRAM

as written



without `sameline alpha beta gamma`



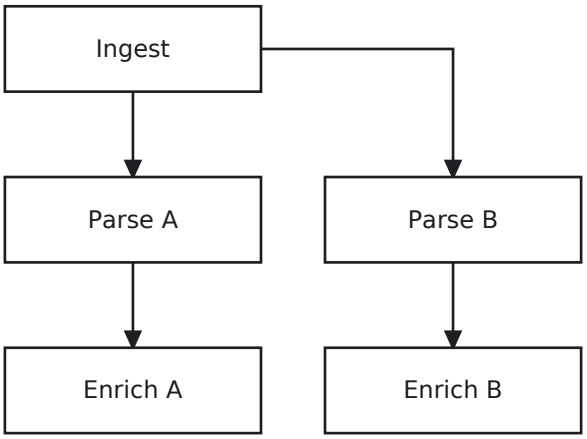
027 Two straight paths

Two paths, each kept on a line of its own.

SOURCE

```
box ingest "Ingest"
box pa "Parse A"
box ea "Enrich A"
box pb "Parse B"
box eb "Enrich B"
ingest -> pa -> ea
ingest -> pb -> eb
sameline pa ea
sameline pb eb
```

DIAGRAM



Once a diagram has more than a handful of boxes, where they sit starts to matter. This part is the statements that say how far apart things are, which side a branch takes, and what a box sits in the middle of.

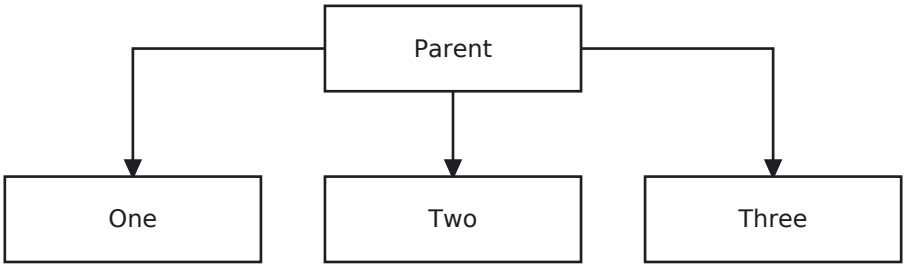
028 Several arrows out of one box

When several arrows leave one box, the boxes they point at are placed side by side, left to right, in the order you wrote them.

SOURCE

```
box parent "Parent"
box one "One"
box two "Two"
box three "Three"
parent -> one
parent -> two
parent -> three
```

DIAGRAM



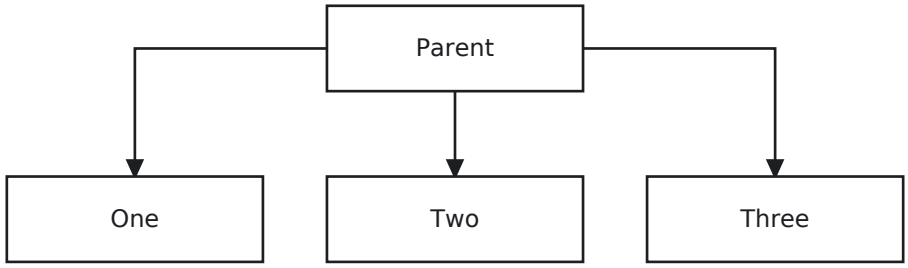
029 Centering over several boxes

A box that points at several is drawn at the middle of them.

SOURCE

```
box parent "Parent"
box one "One"
box two "Two"
box three "Three"
parent -> one
parent -> two
parent -> three
```

DIAGRAM



030 # order

Sometimes you want a branch on the other side. **order** sets the left-to-right order of the boxes under one box.

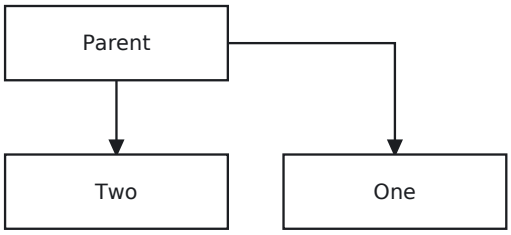
Without it they go in the order you wrote them. **order** changes that for one box and leaves the rest of the diagram alone.

SOURCE

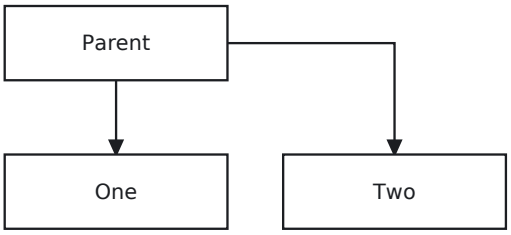
```
box parent "Parent"
box one "One"
box two "Two"
parent -> one
parent -> two
order parent: two one
```

DIAGRAM

as written



without `order parent: two one`



031 apart

Sometimes two branches need room between them. **apart** sets how far apart two boxes sit.

The number counts lines, not millimetres. That way it still means the same thing if you change the type size later, and quad keeps every other gap even.

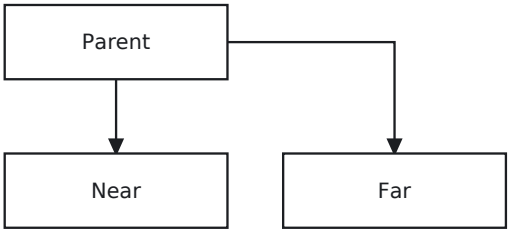
SOURCE

```
box parent "Parent"
box near "Near"
box far "Far"
parent -> near
parent -> far
apart near far: 2 lines
```

DIAGRAM

as written

without `apart near far: 2 lines`



032 apart, further

The same statement with a bigger number: one line wider.

SEE ALSO apart, page 23

SOURCE

```
box parent "Parent"
box near "Near"
box far "Far"
parent -> near
parent -> far
apart near far: 3 lines
```

DIAGRAM

033 between

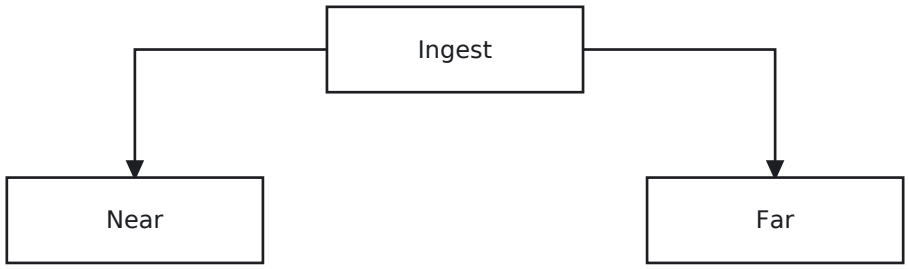
between puts a box exactly in the middle of the boxes it names.

A box is centered over the ones under it already. Write **between** when the centering has to be exact: if the middle falls in the gap between two lines, quad stops and tells you rather than picking a side.

SOURCE

```
box ingest "Ingest"
box near "Near"
box far "Far"
ingest -> near
ingest -> far
apart near far: 2 lines
between near far: ingest
```

DIAGRAM



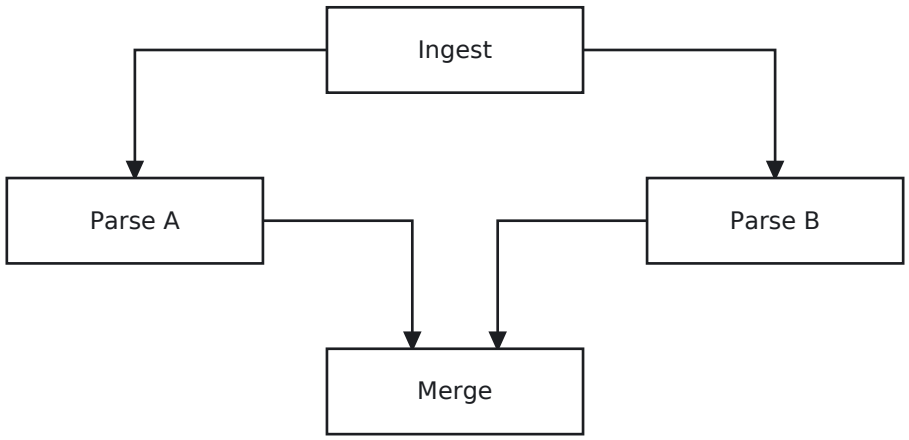
034 between at both ends

The box the branches leave and the box they meet at are both placed this way, so the diagram comes out symmetrical.

SOURCE

```
box ingest "Ingest"
box pa "Parse A"
box pb "Parse B"
box merge "Merge"
ingest -> pa -> merge
ingest -> pb -> merge
apart pa pb: 2 lines
between pa pb: ingest
between pa pb: merge
```

DIAGRAM



035 # side

Sometimes a small branch should always hang off the same side. **side** says which. Left is the side this one takes on its own, so the picture is unchanged.

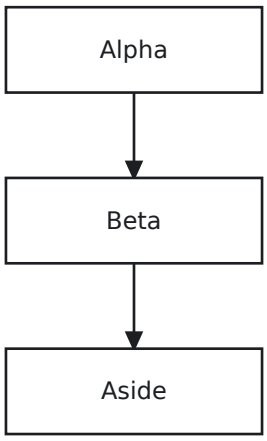
side is measured against the first **sameline** in the file, so a diagram needs one for **side** to mean anything.

SEE ALSO side right, page 27

SOURCE

```
box alpha "Alpha"
box beta "Beta"
box aside "Aside"
alpha -> beta
beta -> aside
sameline alpha beta
side aside left
```

DIAGRAM



036 side right

The same statement with **right**, which moves the branch to the other side of the line.

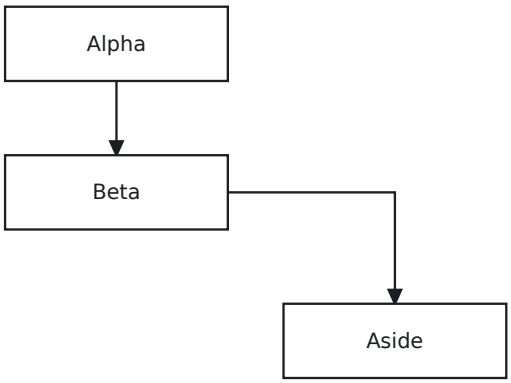
SEE ALSO side, page 26

SOURCE

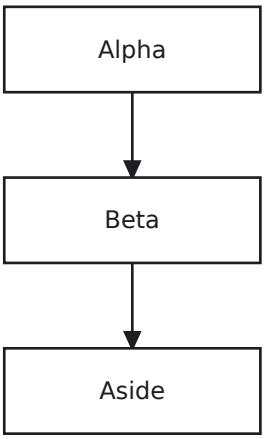
```
box alpha "Alpha"
box beta "Beta"
box aside "Aside"
alpha -> beta
beta -> aside
sameline alpha beta
side aside right
```

DIAGRAM

as written



without `side aside right`



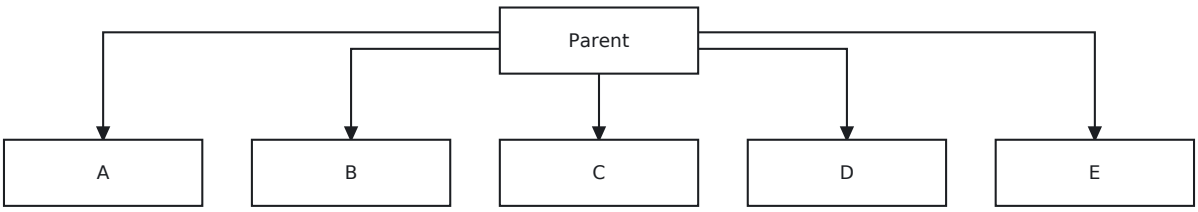
037 A wide step

Five arrows out of one box. The five boxes spread across one step, evenly spaced.

SOURCE

```
box parent "Parent"
box a "A"
box b "B"
box c "C"
box d "D"
box e "E"
parent -> a
parent -> b
parent -> c
parent -> d
parent -> e
```

DIAGRAM



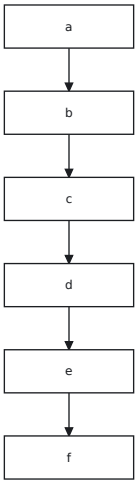
038 A long chain

However long the chain, the gap between one step and the next is the same.

SOURCE

```
a -> b -> c -> d -> e -> f
```

DIAGRAM



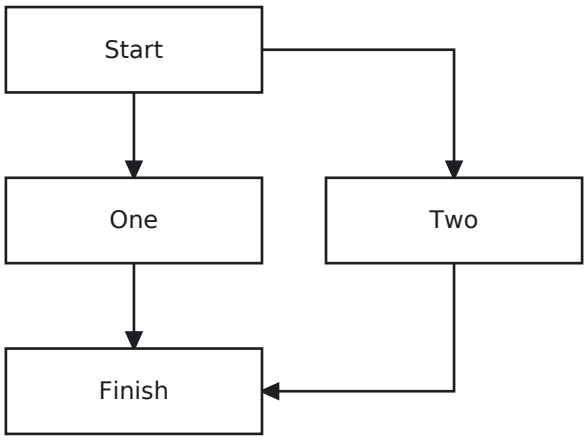
039 A fork that meets again

Two branches that come back together. The box they meet at is placed after the longer of the two.

SOURCE

```
box start "Start"
box one "One"
box two "Two"
box finish "Finish"
start -> one -> finish
start -> two -> finish
```

DIAGRAM



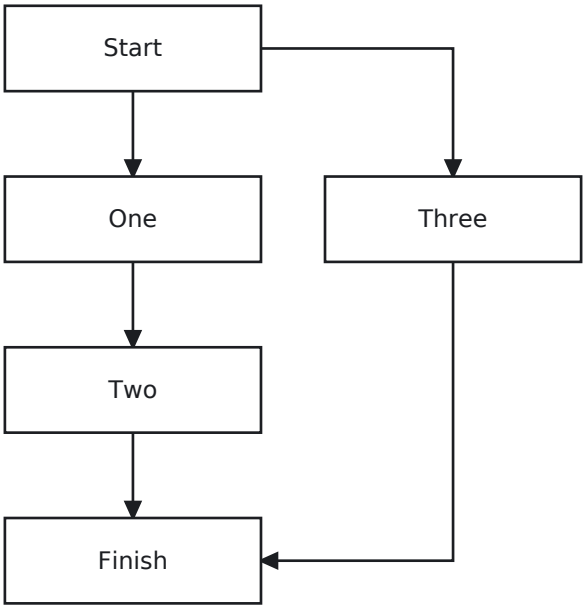
040 Branches of different lengths

When one branch is longer, the box they meet at is placed after the end of that branch.

SOURCE

```
box start "Start"
box one "One"
box two "Two"
box three "Three"
box finish "Finish"
start -> one -> two -> finish
start -> three -> finish
```

DIAGRAM



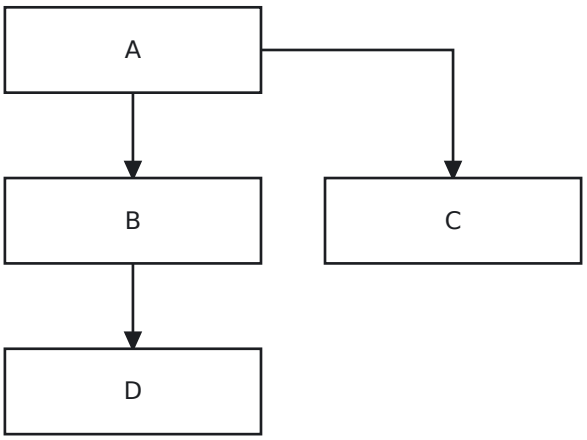
041 samestep and sameline together

sameline puts boxes on one line in the direction the diagram runs; **samestep** puts them side by side in one step.

SOURCE

```
box a "A"
box b "B"
box c "C"
box d "D"
a -> b
a -> c
b -> d
samestep b c
sameline a d
```

DIAGRAM



An arrow can carry a word, leave from a side you pick, come back to the box it left, or run twice between the same pair. This part is everything an arrow can do.

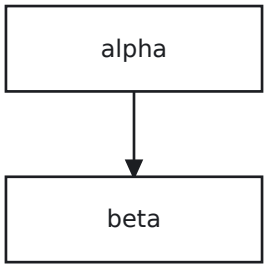
042 The plain arrow

-> is the arrow you have been using. The others change what the line means rather than where it goes.

SOURCE

```
alpha -> beta
```

DIAGRAM



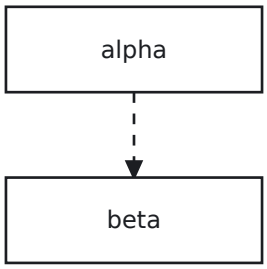
043 A dashed arrow

Sometimes a connection is weaker than the others around it. **-->** draws the same arrow dashed.

SOURCE

```
alpha --> beta
```

DIAGRAM



044 An arrow both ways

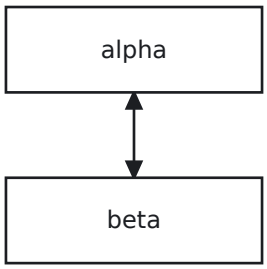
<-> draws one arrow with a head at each end, for something that goes both ways.

It is one arrow, not two. Writing the pair out as two separate arrows gives you two lines, each with its own label.

SOURCE

```
alpha <-> beta
```

DIAGRAM



045 An arrow that runs straight

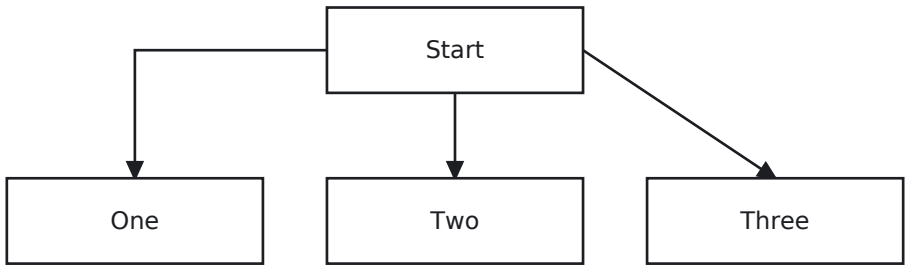
All three arrows leave the same box for the same step. The two written **->** turn a right angle to get there; the one written **~>** is a straight line, at whatever angle that takes.

~> draws one arrow as a straight line rather than as turns at right angles. Reach for it where the turns send an arrow a long way round to say something short.

SOURCE

```
box start "Start"
box one "One"
box two "Two"
box three "Three"
start -> one
start -> two
start ~> three
```

DIAGRAM



046 Words on an arrow

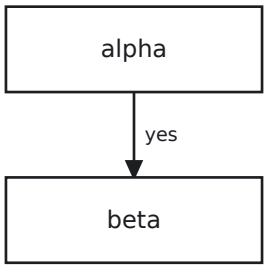
Sometimes an arrow needs a word on it. Put a colon after it and write the word.

The word is set beside the line, never across it, and never on top of a box or another label.

SOURCE

```
alpha -> beta : yes
```

DIAGRAM



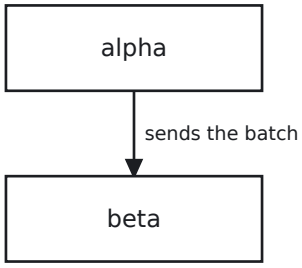
047 More than one word

Quote the words when there is more than one of them.

SOURCE

```
alpha -> beta : "sends the batch"
```

DIAGRAM



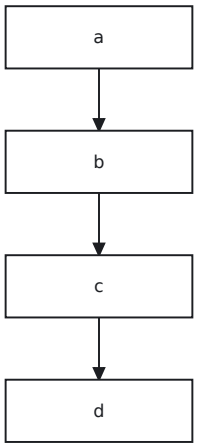
048 A chain of arrows

Arrows can be written one after another on a line, which is the same as writing each of them out.

SOURCE

```
a -> b -> c -> d
```

DIAGRAM



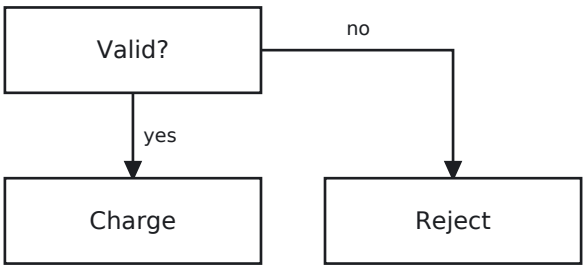
049 A choice with two answers

Two arrows out of one box, each with its own label.

SOURCE

```
box valid "Valid?"
box charge "Charge"
box reject "Reject"
valid -> charge : yes
valid -> reject : no
```

DIAGRAM



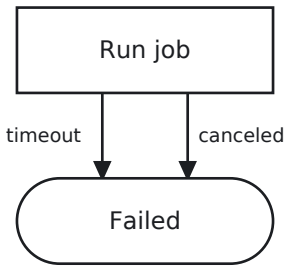
050 Two arrows between the same boxes

Two arrows between the same pair of boxes stay two arrows. Each one leaves from its own place on the side of the box, so both labels can be read.

SOURCE

```
box job "Run job"
end failed "Failed"
job -> failed : timeout
job -> failed : canceled
```

DIAGRAM



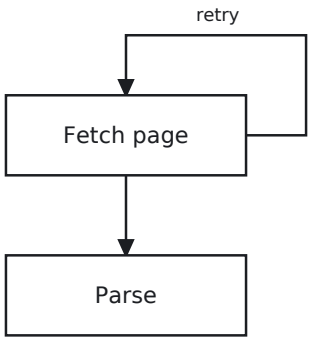
051 An arrow back to the same box

Write a box's own name at both ends and the arrow leaves one side and comes back into another.

SOURCE

```
box fetch "Fetch page"
box parse "Parse"
fetch -> fetch : retry
fetch -> parse
```

DIAGRAM



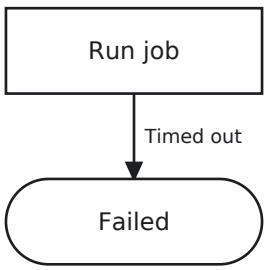
052 Giving an arrow a name

edge names an arrow and gives it its own words, so a later statement can pick that arrow out.

SOURCE

```
box job "Run job"
end failed "Failed"
edge timeout job -> failed "Timed out"
```

DIAGRAM



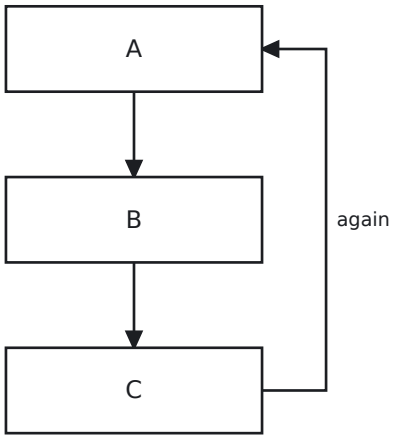
053 An arrow that goes back

An arrow pointing at an earlier box is drawn round the outside, rather than back up between the boxes.

SOURCE

```
box a "A"
box b "B"
box c "C"
a -> b -> c
c -> a : again
```

DIAGRAM



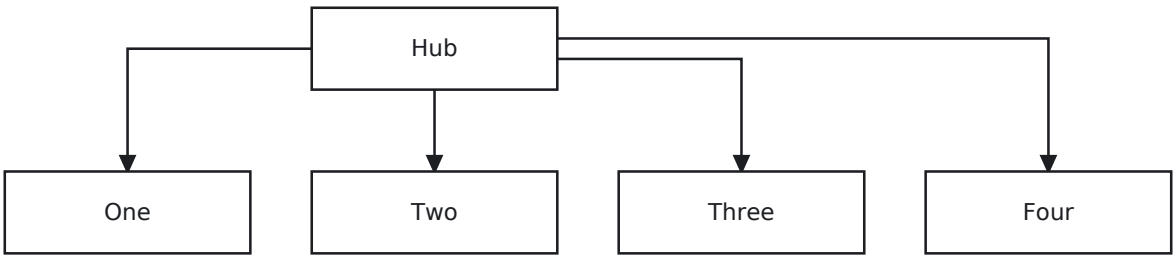
054 Many arrows out of one box

Each arrow leaves by the side nearest the box it points at.

SOURCE

```
box hub "Hub"
box one "One"
box two "Two"
box three "Three"
box four "Four"
hub -> one
hub -> two
hub -> three
hub -> four
```

DIAGRAM



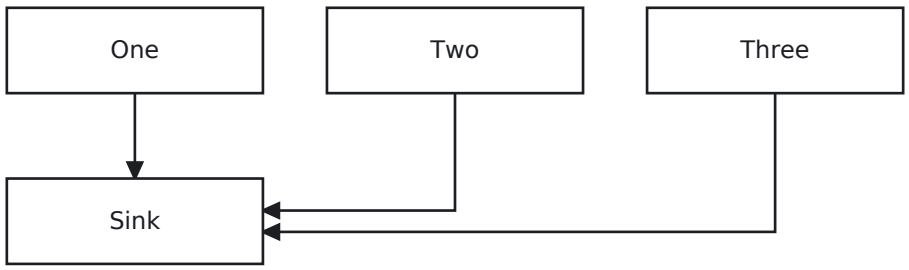
055 Many arrows into one box

Each arrow arrives on the side nearest where it came from, so none of them cross.

SOURCE

```
box one "One"
box two "Two"
box three "Three"
box sink "Sink"
one -> sink
two -> sink
three -> sink
```

DIAGRAM



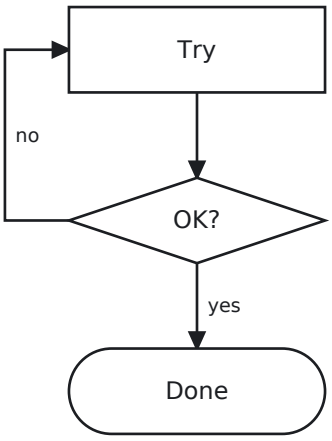
056 Choosing a side yourself

Write a side after a name and that end of the arrow uses it. Here **.left** sends both ends out of the left of their box.

SOURCE

```
box try "Try"
choice ok "OK?"
end done "Done"
try -> ok
ok -> done : yes
ok.left -> try.left : no
```

DIAGRAM



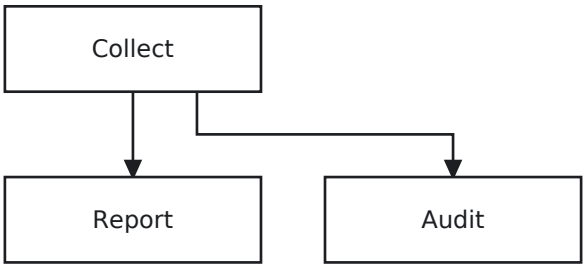
057 The side the diagram runs out of

.out names the side the diagram runs out of: the bottom under **flow down**. Naming it sends this arrow out of there rather than out of the left or the right.

SOURCE

```
box collect "Collect"
box report "Report"
box audit "Audit"
collect -> report
collect.out -> audit
```

DIAGRAM



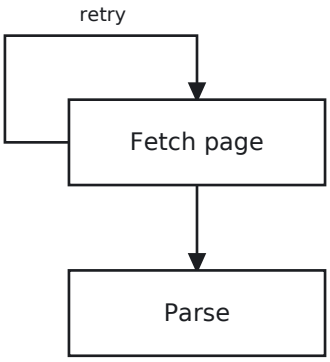
058 Turning a loop over

side turns a box's loop over, so it leaves and returns on the other side.

SOURCE

```
box fetch "Fetch page"
box parse "Parse"
fetch -> fetch : retry
fetch -> parse
side fetch left
```

DIAGRAM



A question with two answers is where a diagram stops being a list. This part is branches: how they open, which way they go, and how they come back together.

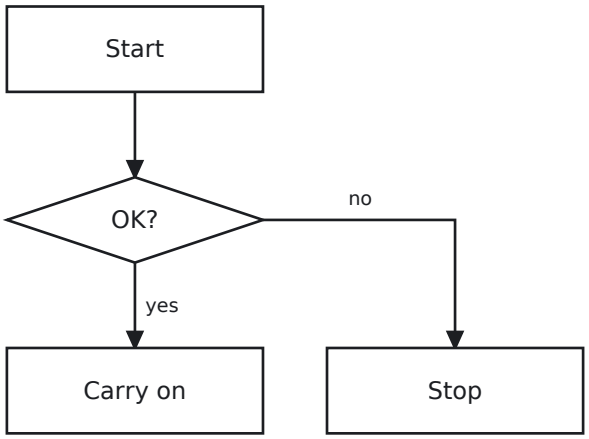
059 A question with two answers

A diamond with an arrow out for each answer.

SOURCE

```
box    start    "Start"
choice ok       "OK?"
box    carry_on "Carry on"
box    stop     "Stop"
start -> ok
ok -> carry_on : yes
ok -> stop : no
```

DIAGRAM



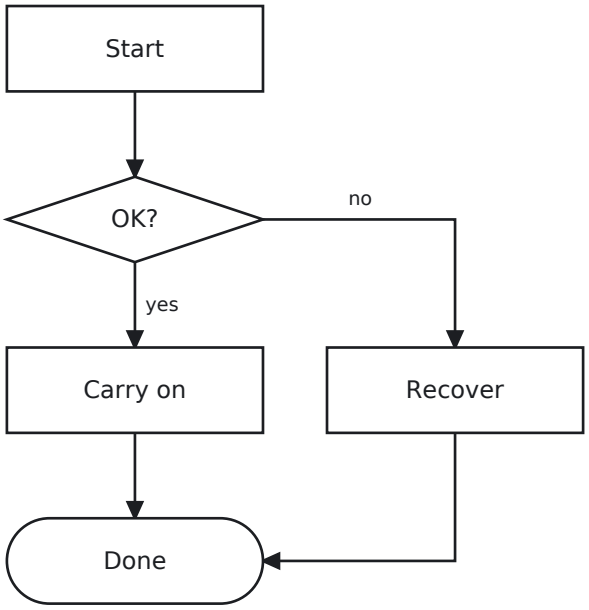
060 Both answers carry on

The two branches meet again at a later step.

SOURCE

```
box    start    "Start"
choice ok       "OK?"
box    carry_on "Carry on"
box    recover  "Recover"
end    done     "Done"
start -> ok
ok -> carry_on : yes
ok -> recover : no
carry_on -> done
recover -> done
```

DIAGRAM



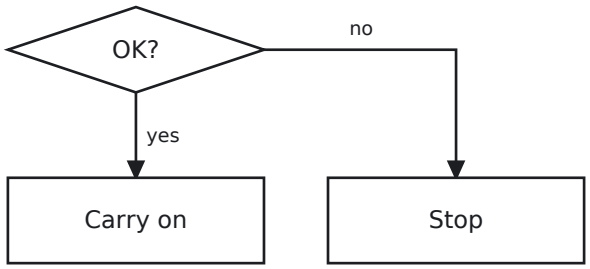
061 The answer on the left

order puts the answer you want read first on the left.

SOURCE

```
choice ok       "OK?"
box    carry_on "Carry on"
box    stop     "Stop"
ok -> carry_on : yes
ok -> stop : no
order ok: carry_on stop
```

DIAGRAM



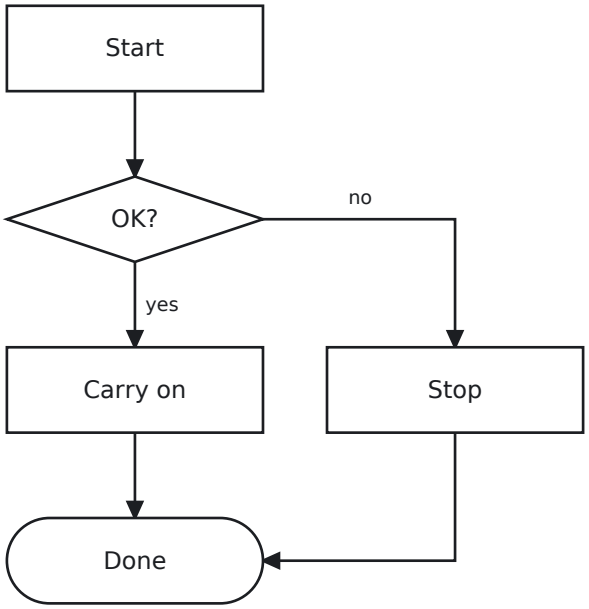
062 Keeping the main answer straight

Naming the main way through with **sameline** keeps it on one line, and the other answer is moved out of its way.

SOURCE

```
box    start    "Start"
choice ok       "OK?"
box    carry_on "Carry on"
box    stop     "Stop"
end    done     "Done"
start -> ok
ok -> carry_on : yes
ok -> stop : no
carry_on -> done
stop -> done
sameline start ok carry_on done
```

DIAGRAM



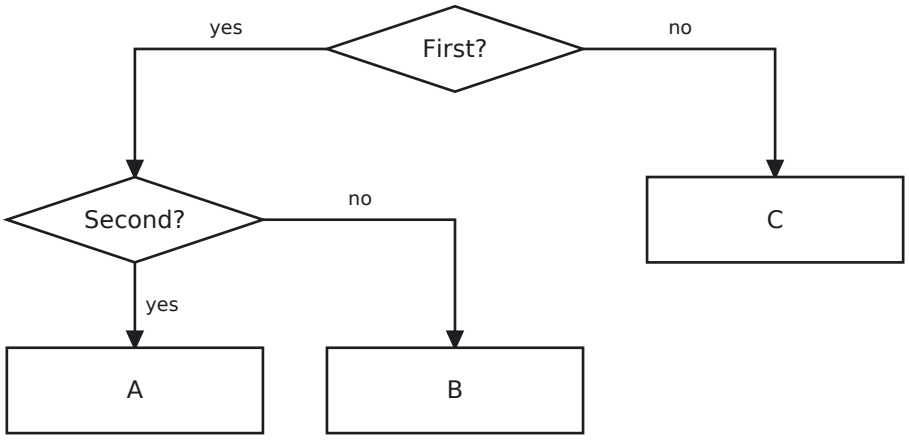
063 A question after a question

One diamond leads to another.

SOURCE

```
choice first "First?"
choice second "Second?"
box a "A"
box b "B"
box c "C"
first -> second : yes
first -> c : no
second -> a : yes
second -> b : no
```

DIAGRAM



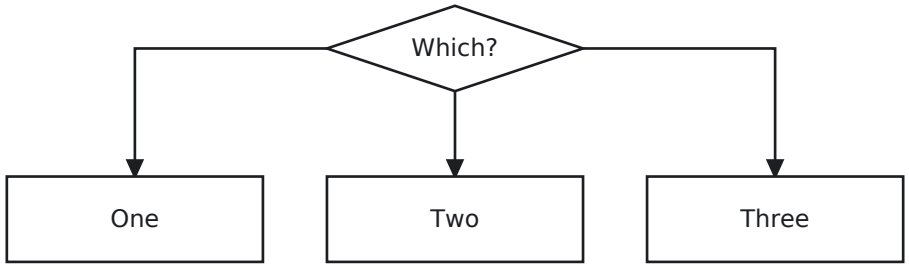
064 A question with three answers

Three arrows out of one box put all three at the next step, side by side.

SOURCE

```
choice pick "Which?"
box one "One"
box two "Two"
box three "Three"
pick -> one
pick -> two
pick -> three
```

DIAGRAM



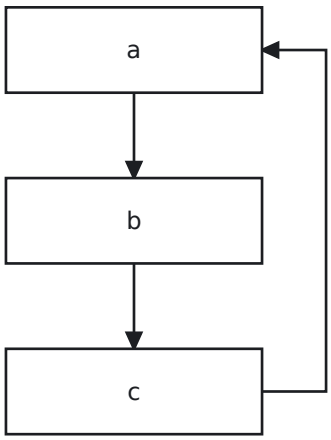
065 A loop in the diagram

Arrows that come back round to where they started. quad draws one of them backwards, and the same one every time.

SOURCE

```
a -> b -> c -> a
```

DIAGRAM



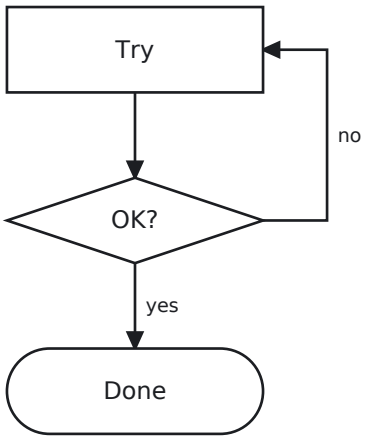
066 Trying again

The failing answer points back at an earlier box, which draws the retry as a loop.

SOURCE

```
box try "Try"
choice ok "OK?"
end done "Done"
try -> ok
ok -> done : yes
ok -> try : no
```

DIAGRAM



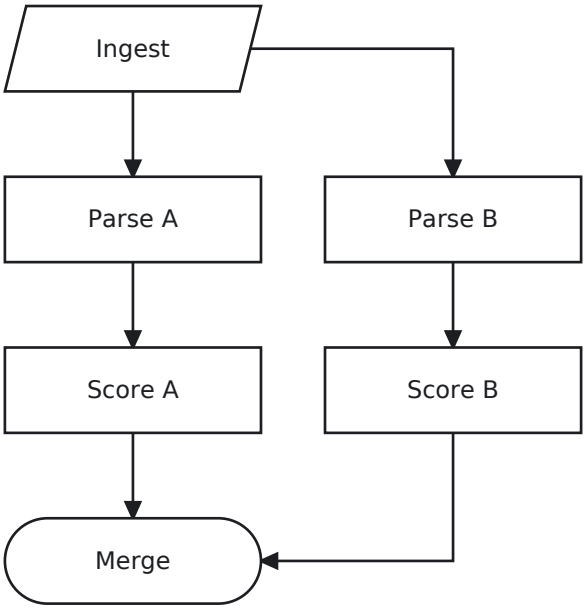
067 Two paths that fork and meet

Two runs of boxes that split at the start and come back together at the end.

SOURCE

```
io  ingest "Ingest"
box pa "Parse A"
box sa "Score A"
box pb "Parse B"
box sb "Score B"
end merge "Merge"
ingest -> pa -> sa -> merge
ingest -> pb -> sb -> merge
sameline pa sa
sameline pb sb
```

DIAGRAM



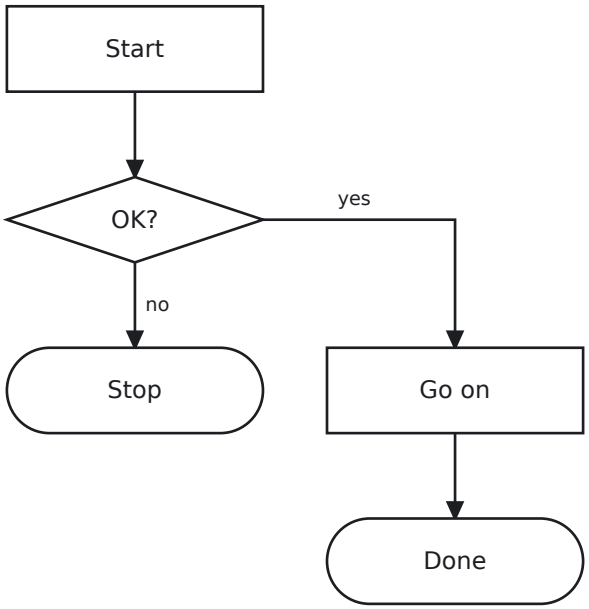
068 A branch that stops

One answer stops there instead of rejoining the other.

SOURCE

```
box   start "Start"
choice ok "OK?"
end   stop "Stop"
box   go "Go on"
end   done "Done"
start -> ok
ok -> stop : no
ok -> go : yes
go -> done
```

DIAGRAM



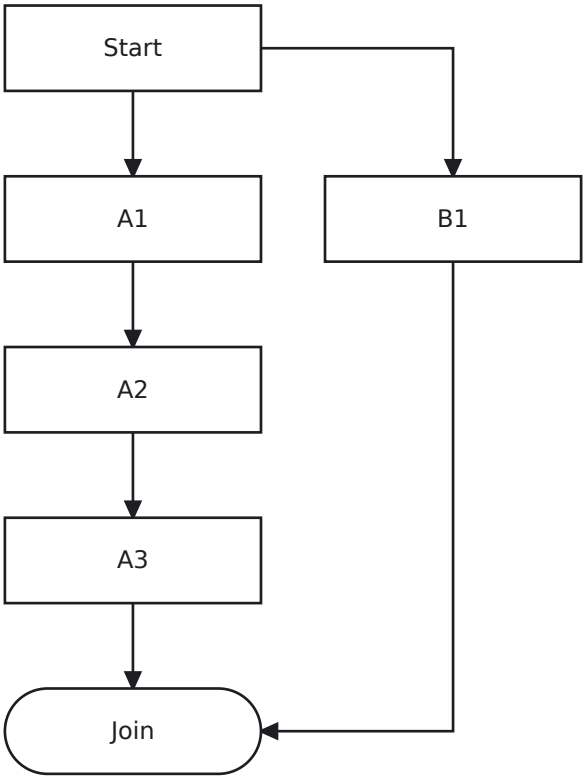
069 One branch longer than the other

The box they meet at is placed after the longer branch, so no arrow has to point backwards.

SOURCE

```
box start "Start"
box a1 "A1"
box a2 "A2"
box a3 "A3"
box b1 "B1"
end join "Join"
start -> a1 -> a2 -> a3 -> join
start -> b1 -> join
```

DIAGRAM



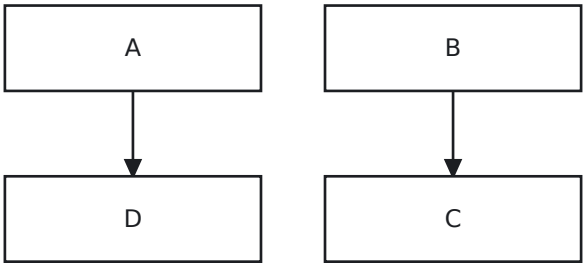
070 Arrows that cross

Sometimes two arrows have to cross. **quadc lint** counts them, and **quadc suggest** proposes a reordering that removes them.

SOURCE

```
box a "A"
box b "B"
box c "C"
box d "D"
a -> d
b -> c
samestep a b
samestep c d
```

DIAGRAM



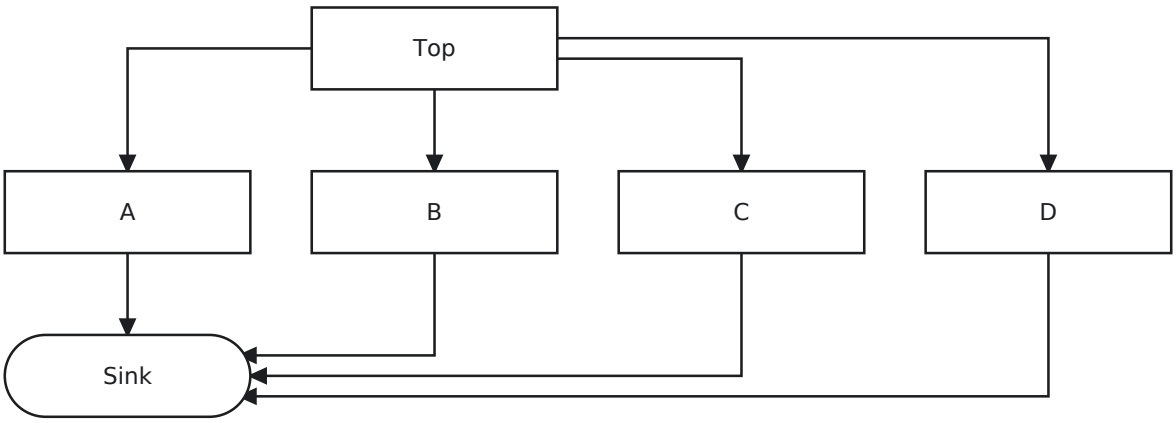
071 Wide, then narrow

Four boxes at one step, all pointing at the same box in the next one.

SOURCE

```
box top "Top"
box a "A"
box b "B"
box c "C"
box d "D"
end sink "Sink"
top -> a
top -> b
top -> c
top -> d
a -> sink
b -> sink
c -> sink
d -> sink
```

DIAGRAM



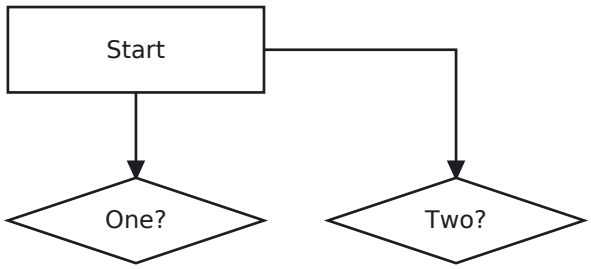
072 Two questions at once

samestep holds both diamonds in one step, so they read as two questions asked at the same point.

SOURCE

```
box start "Start"
choice one "One?"
choice two "Two?"
start -> one
start -> two
samestep one two
```

DIAGRAM



Every box of the same kind is the same size, and that makes a diagram look laid out rather than assembled. This part is how a kind is declared, how a size is settled, and how to make one box as wide as several.

073 kind

Sometimes several boxes mean the same sort of thing. **kind** gives that sort a name, so you write the name instead of the shape.

spans says how wide a box of this kind may be, counted in lines. One number means one width, always.

SOURCE

```
kind stage box spans [1]
stage build "Build"
```

DIAGRAM

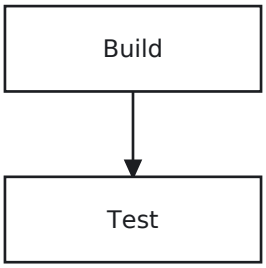
074 Two boxes of one kind

Every box of a kind is drawn the same size, so two of them can be compared.

SOURCE

```
kind stage box spans [1]
stage build "Build"
stage test "Test"
build -> test
```

DIAGRAM



075 Letting a kind grow

Two numbers in **spans** mean the kind may take one line or two. The narrower one is used where the words fit it.

SOURCE

```
kind stage box spans [1, 2]
stage build "Build"
```

DIAGRAM

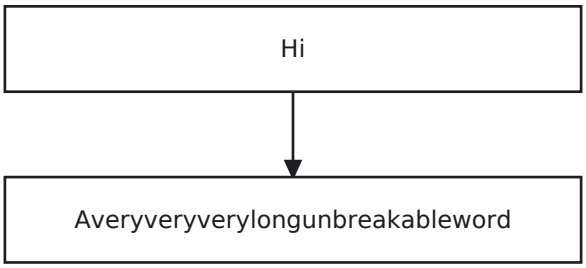
076 One long label widens them all

If one box of a kind needs the wider size, every box of that kind takes it, so they stay the same size as each other.

SOURCE

```
kind stage box spans [1, 2]
stage short "Hi"
stage long "Averyveryverylongunbreakableword"
short -> long
```

DIAGRAM



077 A kind that may not grow

With one width allowed, words that will not fit are an error rather than a box that overflows.

SOURCE

```
kind gate choice spans [1]
gate ok "Fits"
```

DIAGRAM

078 A diamond runs out of room sooner

A diamond has about half the room a rectangle does, so it takes the wider size on shorter words.

SOURCE

```
kind gate choice spans [1, 2]
gate ok "Payload valid?"
```

DIAGRAM

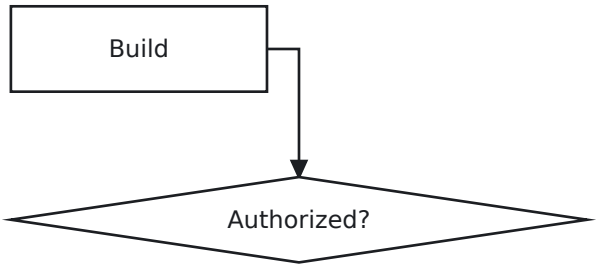
079 Two kinds side by side

Each kind settles on its own width. Widening one leaves the other where it was.

SOURCE

```
kind stage box spans [1, 2]
kind gate choice spans [1, 2]
stage build "Build"
gate ok "Authorized?"
build -> ok
```

DIAGRAM



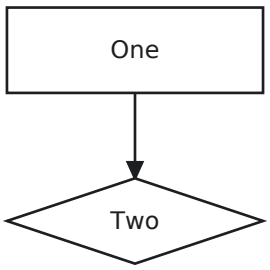
080 The shape does not change the size

A rectangle and a diamond take up exactly the same room, whatever their outlines do inside it.

SOURCE

```
box    one "One"
choice two "Two"
one -> two
```

DIAGRAM



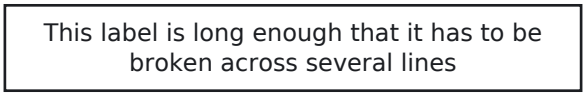
081 Long words wrap

Words too long for the box are broken across lines, and only where they do not fit.

SOURCE

```
box notice "This label is long enough that it has to be broken across several lines"
```

DIAGRAM



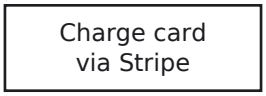
082 Your own line breaks are kept

Each quoted string is one line. quad adds no breaks it does not need.

SOURCE

```
box charge "Charge card" "via Stripe"
```

DIAGRAM



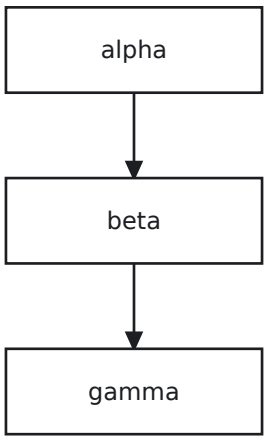
083 Without any kinds

With no **kind** anywhere, every name is a box of the same size. Declare one when two sorts of box should be told apart.

SOURCE

```
alpha -> beta -> gamma
```

DIAGRAM



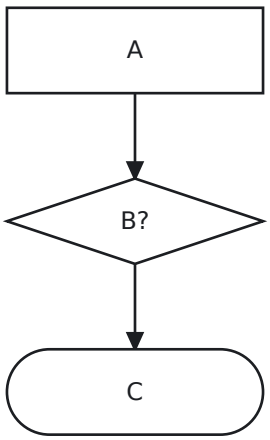
084 A kind of any shape

Any of the six shapes can be given a name of its own.

SOURCE

```
kind stage  box    spans [1]
kind gate   choice spans [1]
kind finish end    spans [1]
stage a "A"
gate  b "B?"
finish c "C"
a -> b -> c
```

DIAGRAM



085 covers

Sometimes one box serves several others and should be as wide as all of them together. **covers** says so without you counting lines.

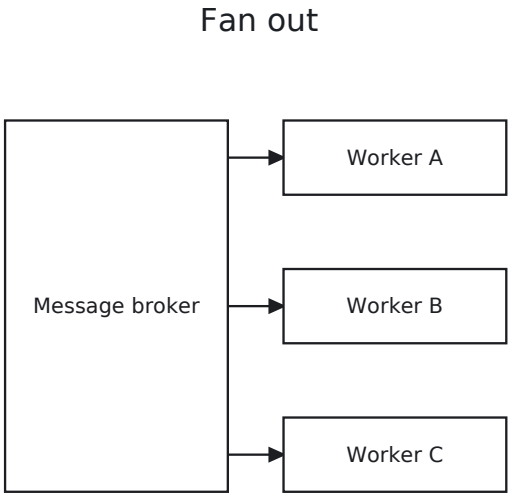
It stretches from the first box it names to the last. Add another to the list and the box is still the right size.

SOURCE

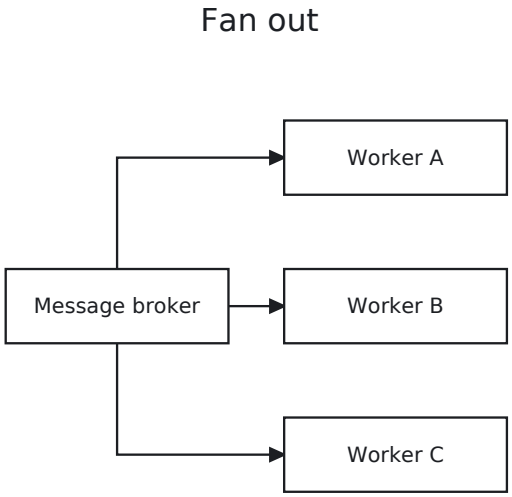
```
diagram "Fan out" { flow right }
box broker "Message broker"
box wa "Worker A"
box wb "Worker B"
box wc "Worker C"
broker -> wa
broker -> wb
broker -> wc
covers broker: wa wb wc
```

DIAGRAM

as written



without `covers broker: wa wb wc`



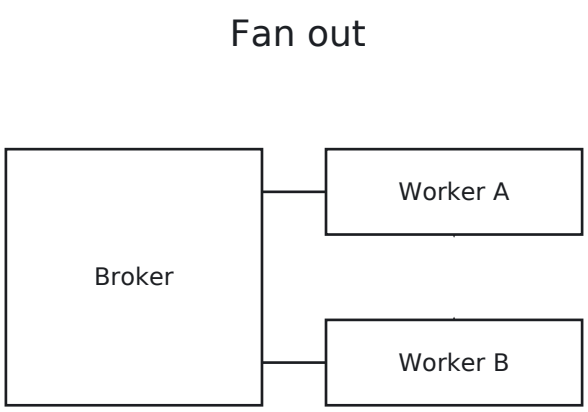
086 Covering two

The same statement over two boxes instead of three.

SOURCE

```
diagram "Fan out" { flow right }
box broker "Broker"
box wa "Worker A"
box wb "Worker B"
broker -> wa
broker -> wb
covers broker: wa wb
```

DIAGRAM



087 Covering the other way

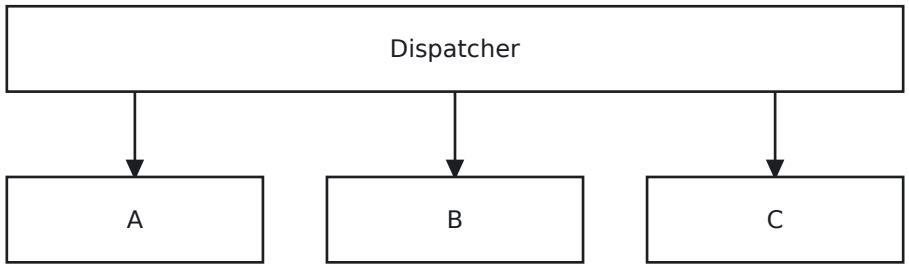
With the diagram running down the page the same statement makes a box wide rather than tall.

SEE ALSO covers, page 66

SOURCE

```
box dispatcher "Dispatcher"
box a "A"
box b "B"
box c "C"
dispatcher -> a
dispatcher -> b
dispatcher -> c
covers dispatcher: a b c
```

DIAGRAM



Some boxes belong together. This part draws a region around them, places the region as one thing, and lets what is inside run in its own direction.

088 group

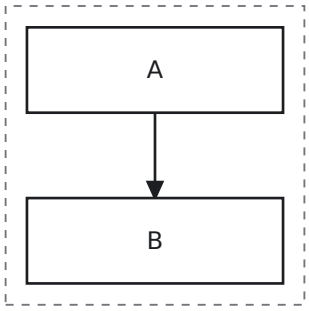
Sometimes a few boxes belong together. **group** draws a region round them and keeps them side by side.

A group is placed as one thing: nothing from outside it lands among its boxes, and moving it moves them all.

SOURCE

```
box a "A"
box b "B"
a -> b
group pair { a b }
```

DIAGRAM



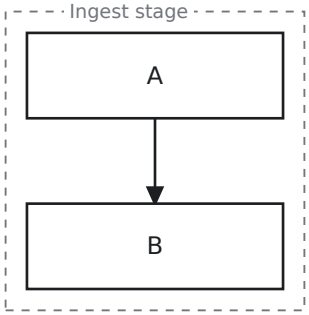
089 Naming the region

Put words in quotes after the group's name and they are drawn on the outline.

SOURCE

```
box a "A"
box b "B"
a -> b
group ingest "Ingest stage" { a b }
```

DIAGRAM



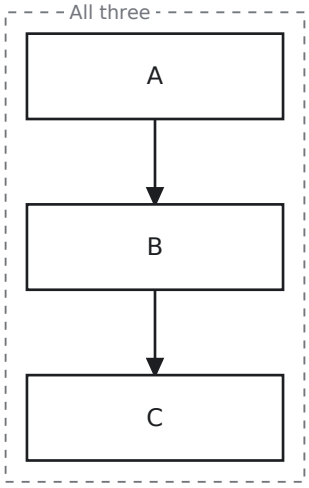
090 # A group of three

However many boxes a group holds, they stay next to each other.

SOURCE

```
box a "A"
box b "B"
box c "C"
a -> b -> c
group all "All three" { a b c }
```

DIAGRAM



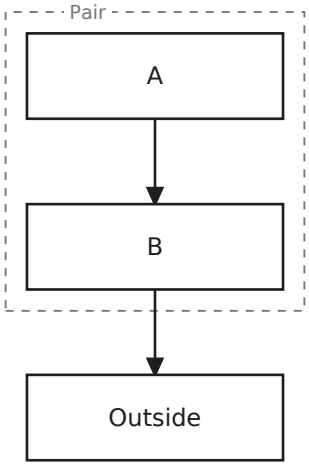
091 A box left outside

Only the boxes you name are enclosed. An arrow may still cross the outline.

SOURCE

```
box a "A"
box b "B"
box outside "Outside"
a -> b
b -> outside
group pair "Pair" { a b }
```

DIAGRAM



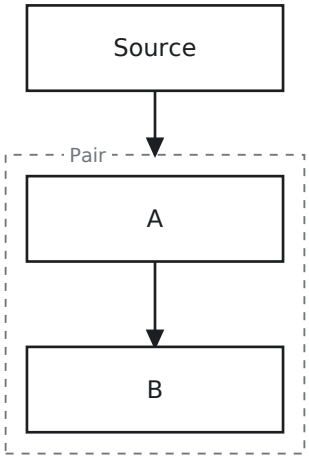
092 An arrow to the whole group

Use the group's name at the end of an arrow and the arrow is about the group rather than any box in it.

SOURCE

```
box source "Source"
box a "A"
box b "B"
a -> b
source -> pair
group pair "Pair" { a b }
```

DIAGRAM



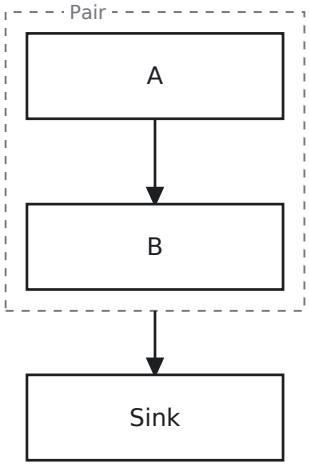
093 An arrow from the whole group

Either end of an arrow may be a group, so an arrow can leave one as well as arrive at one.

SOURCE

```
box a "A"
box b "B"
box sink "Sink"
a -> b
pair -> sink
group pair "Pair" { a b }
```

DIAGRAM



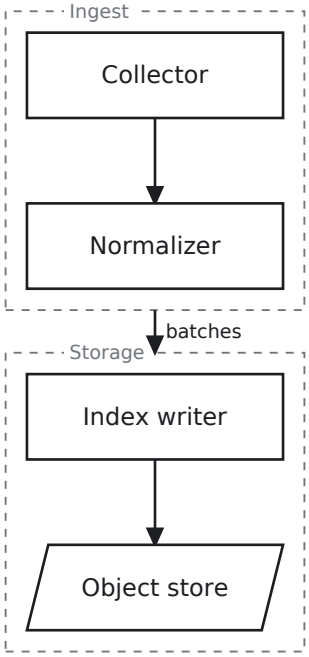
094 One group to another

An arrow between two groups connects the two regions, without naming a box on either side.

SOURCE

```
box collector "Collector"
box normalizer "Normalizer"
box writer "Index writer"
io  objects "Object store"
collector -> normalizer
writer -> objects
ingest -> storage : batches
group ingest "Ingest" { collector normalizer }
group storage "Storage" { writer objects }
```

DIAGRAM



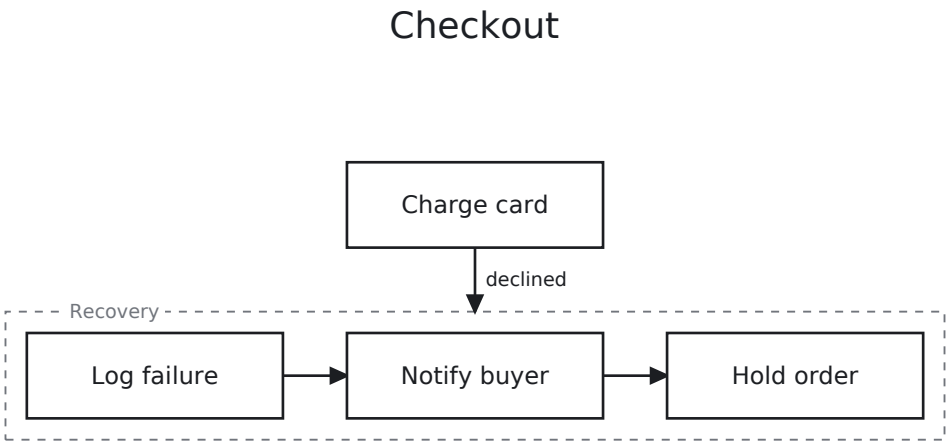
095 A group that runs its own way

Write **flow** inside the group and its boxes run that way. The rest of the diagram is unchanged.

SOURCE

```
diagram "Checkout" { flow down }
box charge "Charge card"
box log "Log failure"
box notify "Notify buyer"
box hold "Hold order"
log -> notify -> hold
charge -> recovery : declined
group recovery "Recovery" { flow right; log notify hold }
```

DIAGRAM



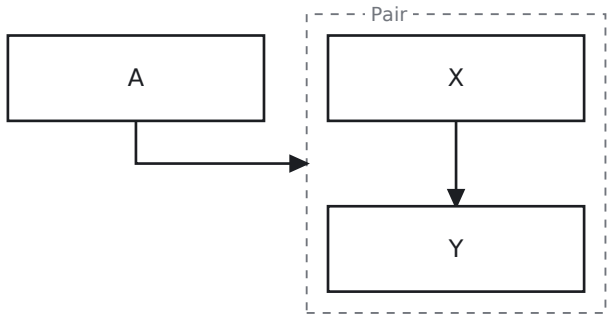
096 The other way round

A group running down inside a diagram running across the page. Both sit on the same grid.

SOURCE

```
diagram "Pipeline" { flow right }
box a "A"
box x "X"
box y "Y"
x -> y
a -> pair
group pair "Pair" { flow down; x y }
```

DIAGRAM



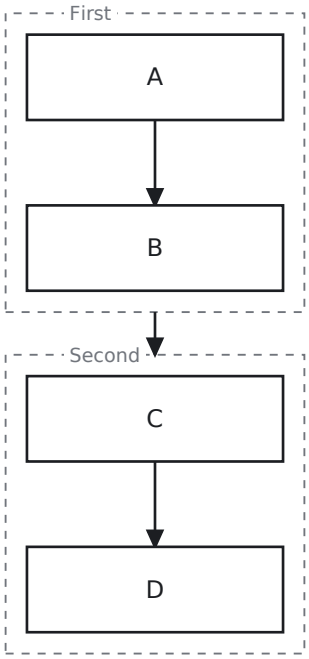
097 Two groups

Each is placed as one thing, so the two regions line up with each other rather than with the boxes inside them.

SOURCE

```
box a "A"
box b "B"
box c "C"
box d "D"
a -> b
c -> d
first -> second
group first "First" { a b }
group second "Second" { c d }
```

DIAGRAM



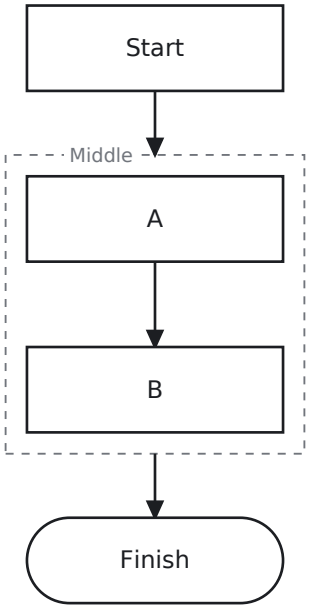
098 A group in the middle of a chain

A group takes its place in the order as a box would.

SOURCE

```
box start "Start"
box a "A"
box b "B"
end finish "Finish"
a -> b
start -> middle
middle -> finish
group middle "Middle" { a b }
```

DIAGRAM



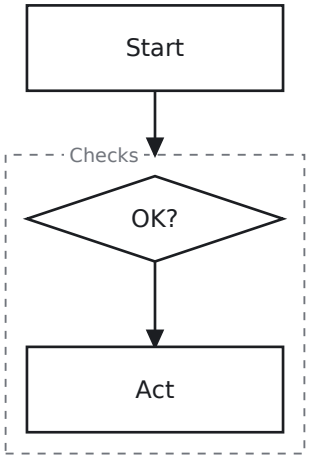
099 Any shape can be a member

A group holds whatever you name in it, diamonds included.

SOURCE

```
box start "Start"
choice ok "OK?"
box act "Act"
start -> checks
ok -> act
group checks "Checks" { ok act }
```

DIAGRAM



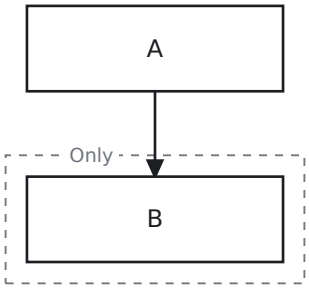
100 A group of one

A region round a single box.

SOURCE

```
box a "A"
box b "B"
a -> b
group only "Only" { b }
```

DIAGRAM



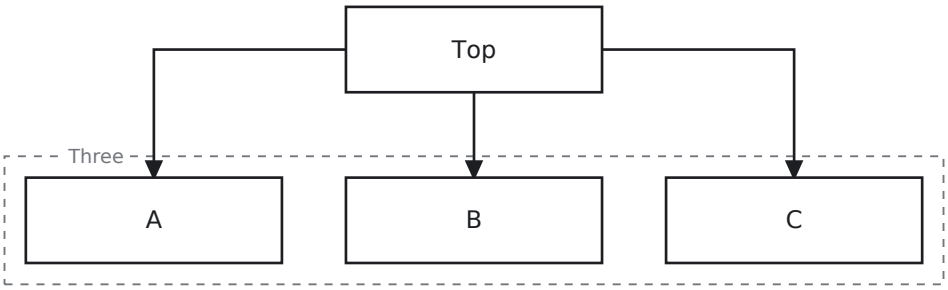
101 The width a group takes

The boxes inside set it: three side by side make a region three lines wide.

SOURCE

```
box top "Top"
box a "A"
box b "B"
box c "C"
top -> a
top -> b
top -> c
group three "Three" { a b c }
```

DIAGRAM



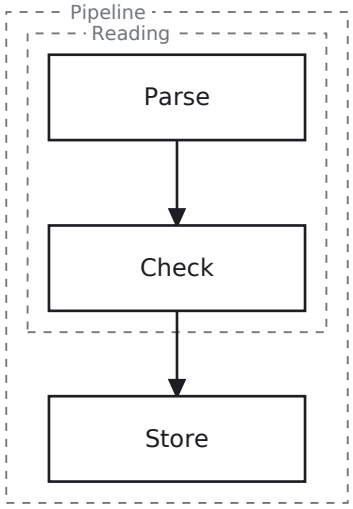
102 A group inside a group

A group can hold another group. The inner outline sits one gap inside the outer one.

SOURCE

```
box parse "Parse"
box check "Check"
box store "Store"
parse -> check -> store
group reading "Reading" { parse check }
group pipeline "Pipeline" { reading store }
```

DIAGRAM



When a diagram has owners — a person, a team, a service — a lane puts a band behind each one's boxes. This part is how the bands are declared and how work hands over between them.

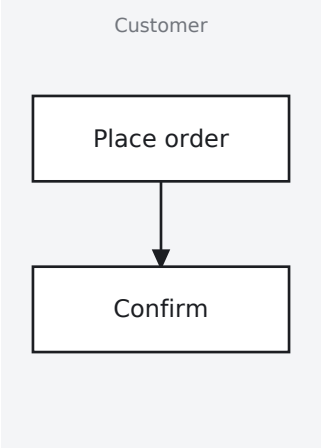
103 lane

Sometimes the question is who does what, not what happens next. **lane** puts the boxes you name in a band of their own, running the length of the diagram.

SOURCE

```
box place "Place order"
box confirm "Confirm"
place -> confirm
lane customer "Customer" { place confirm }
```

DIAGRAM



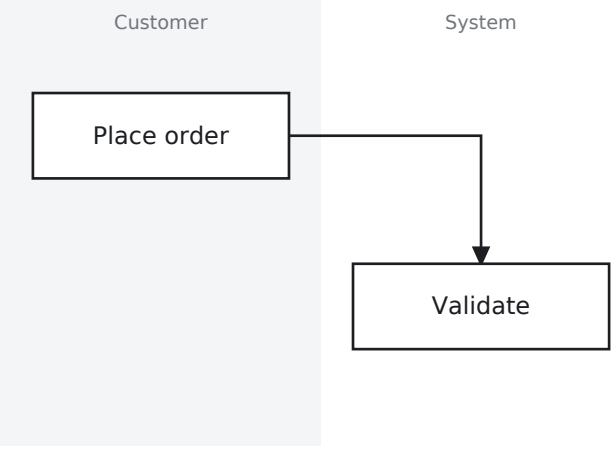
104 Two lanes

Lanes are laid out in the order you declare them, side by side across the diagram.

SOURCE

```
box place "Place order"
box validate "Validate"
place -> validate
lane customer "Customer" { place }
lane system "System" { validate }
```

DIAGRAM



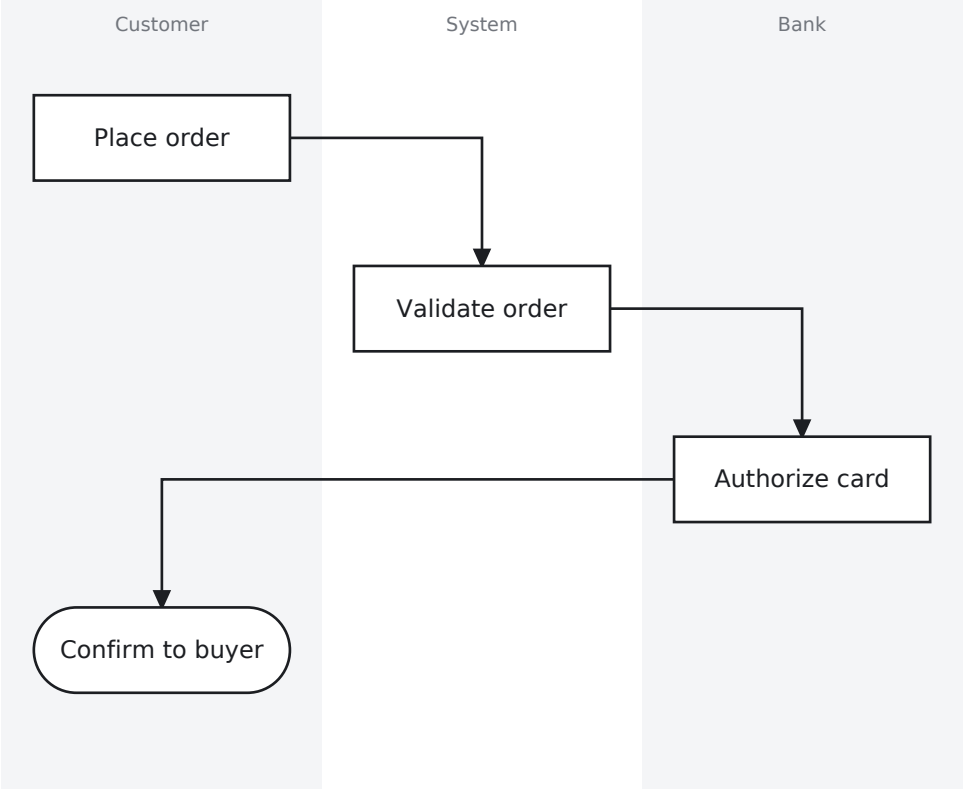
105 Three lanes

Work hands over from one lane to the next, which is the job lanes do.

SOURCE

```
box place "Place order"
box validate "Validate order"
box authorize "Authorize card"
end confirm "Confirm to buyer"
place -> validate -> authorize -> confirm
lane customer "Customer" { place confirm }
lane system "System" { validate }
lane bank "Bank" { authorize }
```

DIAGRAM



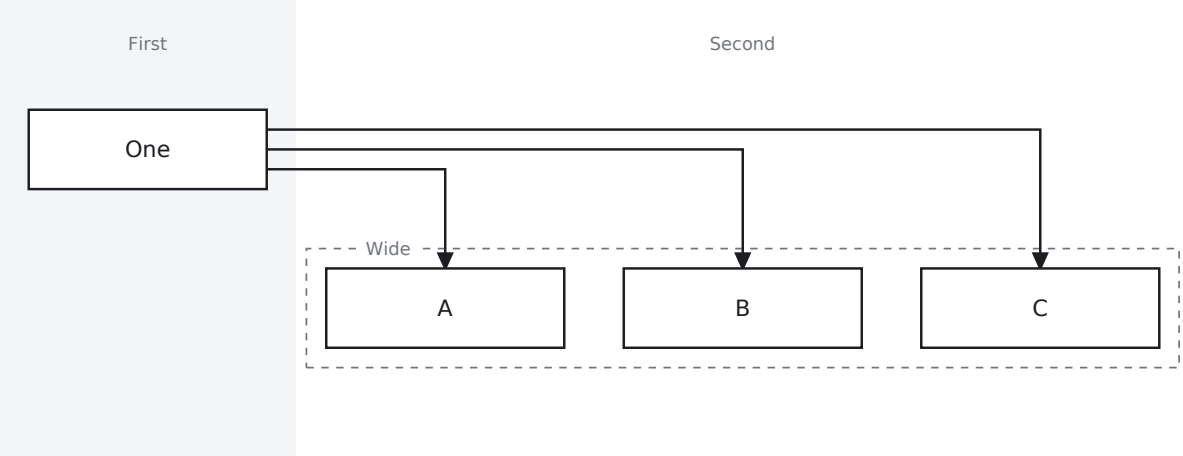
106 The width a lane takes

Each band is as wide as its own boxes need. A wide one does not widen its neighbors.

SOURCE

```
box one "One"
box a "A"
box b "B"
box c "C"
one -> a
one -> b
one -> c
lane first "First" { one }
lane second "Second" { wide }
group wide "Wide" { a b c }
```

DIAGRAM



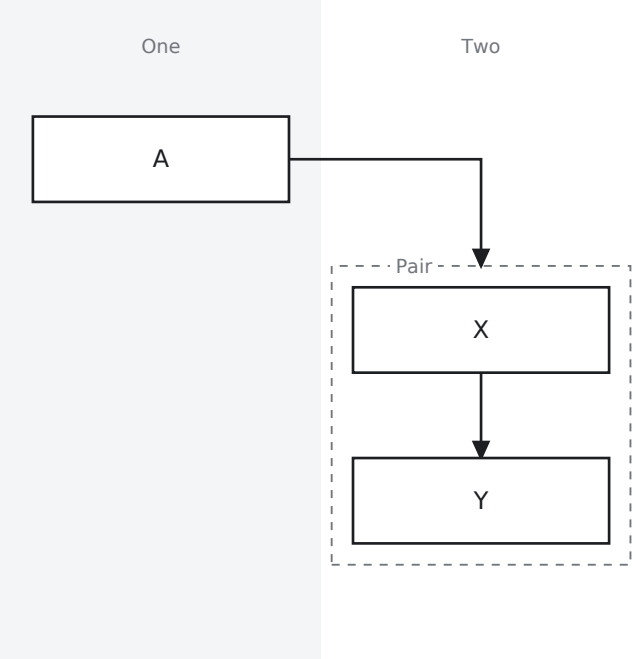
107 A lane holding a group

A lane can hold a group as well as boxes.

SOURCE

```
box a "A"
box x "X"
box y "Y"
x -> y
a -> pair
lane one "One" { a }
lane two "Two" { pair }
group pair "Pair" { x y }
```

DIAGRAM



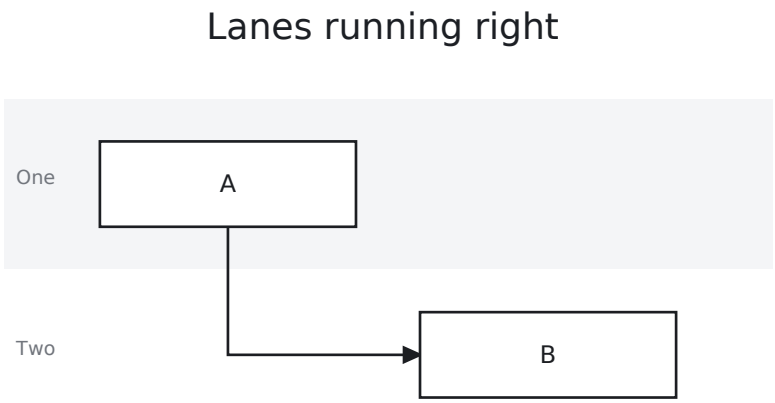
108 Lanes the other way

With the diagram running across the page, the bands are rows instead of columns.

SOURCE

```
diagram "Lanes running right" { flow right }
box a "A"
box b "B"
a -> b
lane one "One" { a }
lane two "Two" { b }
```

DIAGRAM



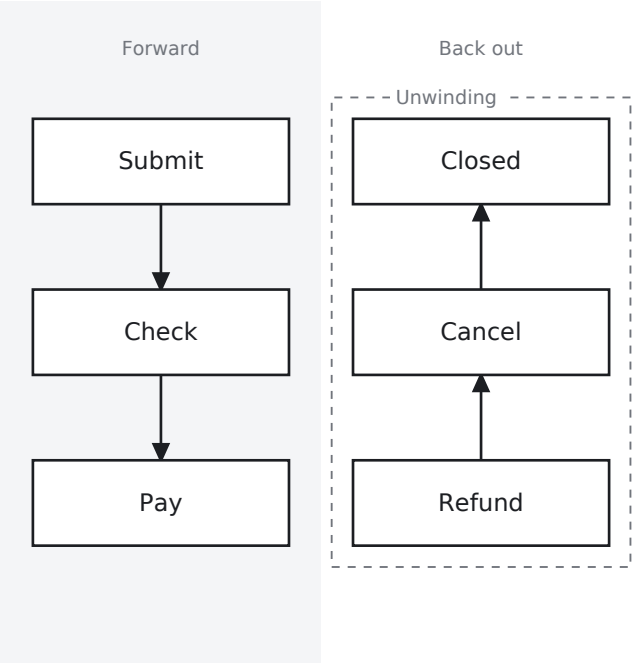
109 A lane that reads the other way

A lane has no direction of its own. To make one read the other way, put a group inside it with its own **flow**.

SOURCE

```
box submit "Submit"
box check "Check"
box pay "Pay"
box refund "Refund"
box cancel "Cancel"
box closed "Closed"
submit -> check -> pay
refund -> cancel -> closed
group unwind "Unwinding" { flow up; refund cancel closed }
lane forward "Forward" { submit check pay }
lane backward "Back out" { unwind }
```

DIAGRAM



A box that means something different should look different. This part marks boxes and arrows with a role, and puts the colors in a theme file so the same diagram can be drawn two ways.

110 role

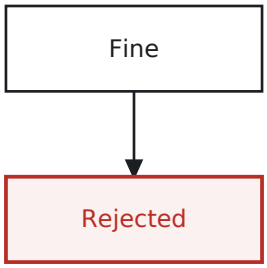
Sometimes a box means something different from the rest: an error, a slow step. **role** says which, and says nothing about how it looks.

A theme, in a separate file, sets what a role looks like. One diagram can then be drawn for a slide and for print without editing it.

SOURCE

```
box ok "Fine"
box bad "Rejected"
ok -> bad
role error { bad }
```

DIAGRAM



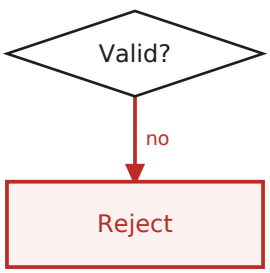
111 A role on an arrow

Arrows take roles as well, so a whole failure path can be marked in one statement.

SOURCE

```
choice valid "Valid?"
box reject "Reject"
valid -> reject : no
role error { reject, valid -> reject }
```

DIAGRAM



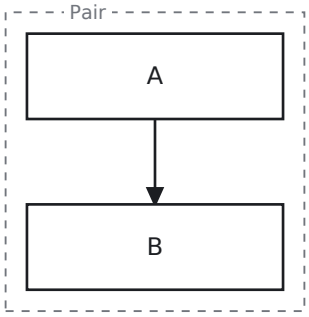
112 A role on a group

A whole region can take a role.

SOURCE

```
box a "A"
box b "B"
a -> b
group pair "Pair" { a b }
role error { pair }
```

DIAGRAM



113 Three roles at once

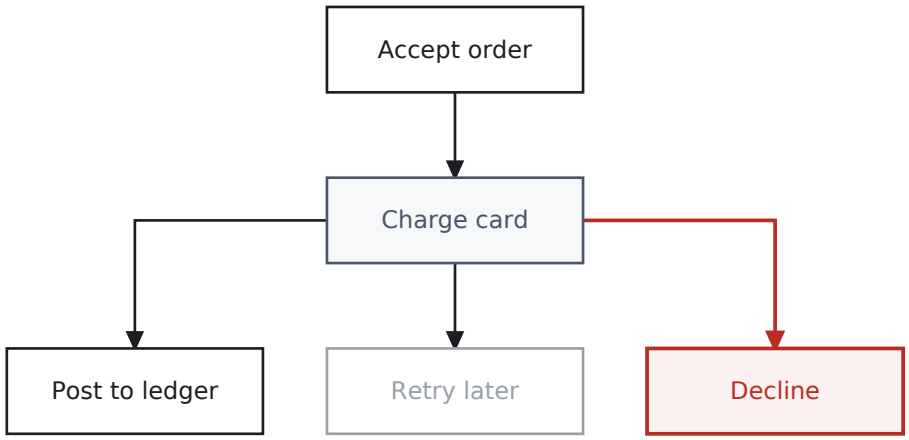
The theme this book is drawn against defines three names, and draws each one differently.

The name is yours to pick. A theme that does not define your name draws that box like any other, so an unknown role is never an error.

SOURCE

```
box   accept  "Accept order"
box   charge  "Charge card"
box   ledger  "Post to ledger"
box   retry   "Retry later"
box   decline "Decline"

accept -> charge -> ledger
charge -> retry
charge -> decline

role external { charge }
role muted { retry }
role error { decline, charge -> decline }
```

DIAGRAM



114 A role you invent

overnight means nothing to quad. The file beside this one is what gives it a look.

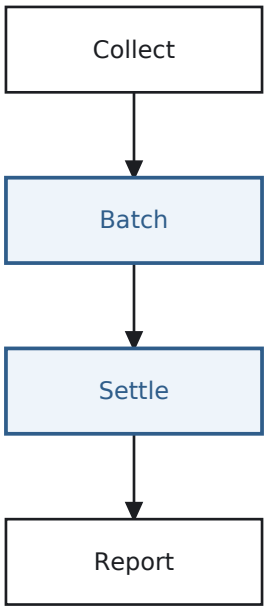
SOURCE

```
# theme: 05-a-role-you-invent.toml

box collect "Collect"
box batch   "Batch"
box settle  "Settle"
box report  "Report"

collect -> batch -> settle -> report

role overnight { batch, settle }
```

THEME 05-a-role-you-invent.toml

```
# A theme that knows one role, and it is not a name the house theme has ever seen.

[role.overnight]
stroke = "#2f5d8a"
fill = "#eef4fa"
text = "#2f5d8a"
width = "1.4pt"
```

DIAGRAM



115 A theme

A theme is a separate file holding the colors, the typeface and the grid. The source names none of them.

The source says what the diagram is; the theme says how it should look. That is what lets one diagram be drawn two ways.

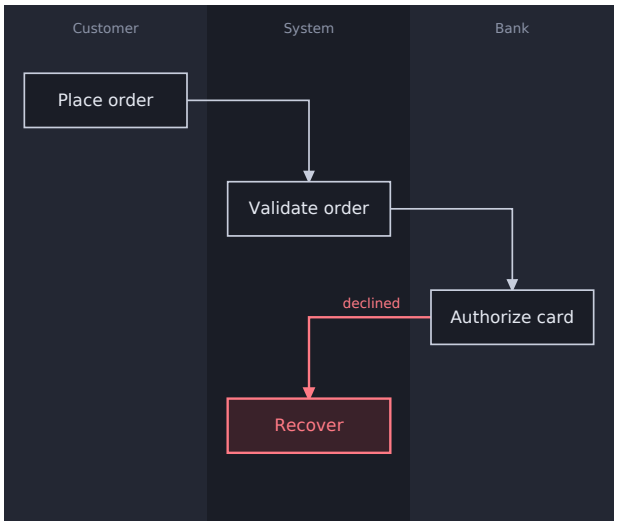
SOURCE

```
# theme: night.toml

box place "Place order"
box validate "Validate order"
box authorize "Authorize card"
box recover "Recover"
place -> validate -> authorize
authorize -> recover : declined
lane customer "Customer" { place }
lane system "System" { validate recover }
lane bank "Bank" { authorize }
role error { recover, authorize -> recover }
```

THEME night.toml

```
# A theme for the same diagram read on a dark page.

[grid]
size = "10pt"

[style]
lane = "#232733"
page = "#1a1d26"
group = "#8b93a7"

[role.default]
stroke = "#c8cedd"
fill = "#1a1d26"
text = "#e6eaf2"

[role.error]
stroke = "#ff7a85"
fill = "#3a222a"
text = "#ff7a85"
width = "1.4pt"
```

DIAGRAM



116 The same diagram, another theme

The same source, drawn against a second theme. Not one box has moved.

SEE ALSO A theme, page 94

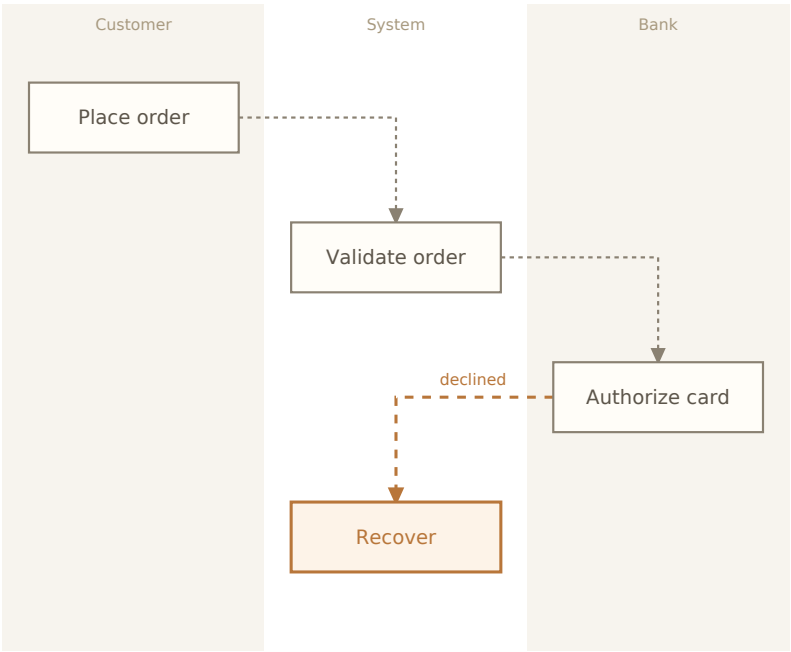
SOURCE

```
# theme: draft.toml

box place "Place order"
box validate "Validate order"
box authorize "Authorize card"
box recover "Recover"
place -> validate -> authorize
authorize -> recover : declined
lane customer "Customer" { place }
lane system "System" { validate recover }
lane bank "Bank" { authorize }
role error { recover, authorize -> recover }
```

THEME draft.toml

```
# A theme for a diagram still being argued about: everything lighter, everything dashed.

[style]
lane = "#f7f4ee"
group = "#a89a80"

[role.default]
stroke = "#8a8172"
fill = "#fffdf8"
text = "#5c554a"
dash = "2pt"
dashed = true

[role.error]
stroke = "#b8763a"
fill = "#fdf3e8"
text = "#b8763a"
dashed = true
```

DIAGRAM



117 A theme can change shapes too

Not only color: this theme draws a loop as a circle rather than as three corners.

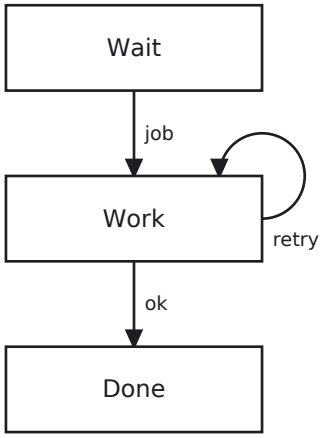
SOURCE

```
# theme: round.toml

box wait "Wait"
box work "Work"
box done "Done"
wait -> work : job
work -> work : retry
work -> done : ok
```

THEME round.toml

```
# A theme for a state machine, where a self-loop is drawn as a circle.

[style]
loops = "round"
```

DIAGRAM



118 A theme can round the corners off

A theme also sets how sharply an arrow turns. This one takes the whole corner away, so an arrow that turns has no straight part left.

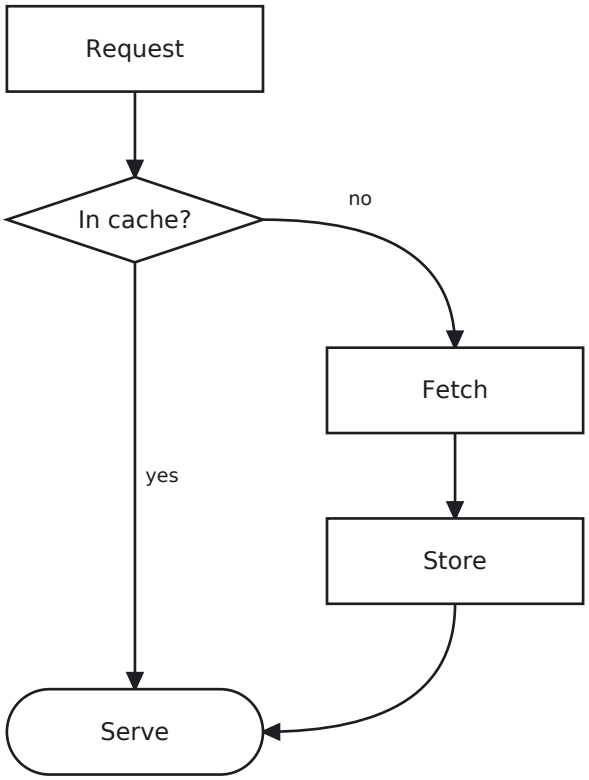
SOURCE

```
# theme: eased.toml

box    request "Request"
choice cached  "In cache?"
box    fetch   "Fetch"
box    store   "Store"
end    serve   "Serve"

request -> cached
cached -> serve : yes
cached -> fetch : no
fetch -> store -> serve
```

THEME eased.toml

```
# A theme where an arrow turns in a curve, keeping none of the corner.

[style]
corners = 0
```

DIAGRAM



119 Or only part of the way

The number a theme names is how much of each corner is kept, rather than a yes or a no. This one keeps half of it, so every arrow turns in a curve with a straight run either side of it.

SEE ALSO A theme can round the corners off, page 97

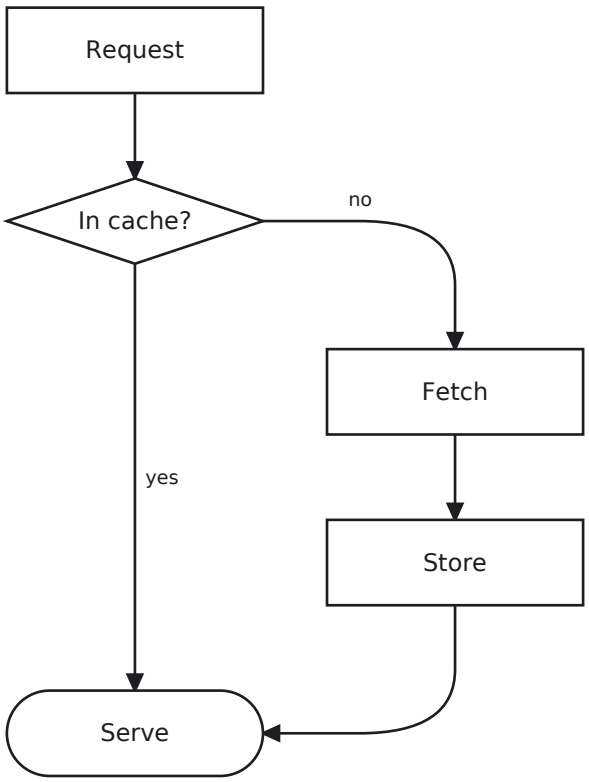
SOURCE

```
# theme: halfway.toml

box    request "Request"
choice cached  "In cache?"
box    fetch   "Fetch"
box    store   "Store"
end    serve   "Serve"

request -> cached
cached -> serve : yes
cached -> fetch : no
fetch -> store -> serve
```

THEME halfway.toml

```
# A theme keeping half of each corner, so a curve has a straight run
# either side of it.

[style]
corners = 50
```

DIAGRAM



120 An arrow that turns twice

An arrow going back turns at both ends of the long run down the side. With the whole corner taken away the two curves meet in the middle of that run, so none of it is left straight.

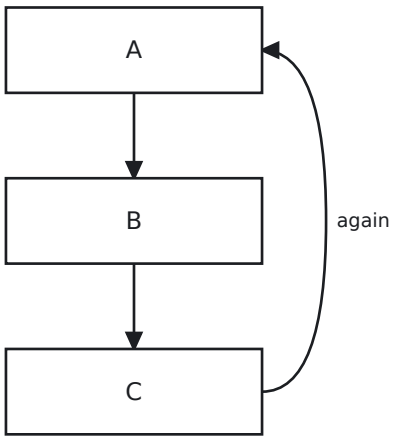
SOURCE

```
# theme: eased.toml

box a "A"
box b "B"
box c "C"
a -> b -> c
c -> a : again
```

THEME eased.toml

```
# A theme where an arrow turns in a curve, keeping none of the corner.

[style]
corners = 0
```

DIAGRAM



Everything so far was drawn on the same grid. This part changes it: the size of a box, the gaps between them, and the face the words are set in.

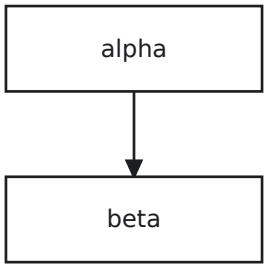
121 The measurements

Every box so far has been drawn to the same size, with the same gaps. This part shows where those numbers come from.

SOURCE

```
alpha -> beta
```

DIAGRAM



122 A smaller unit

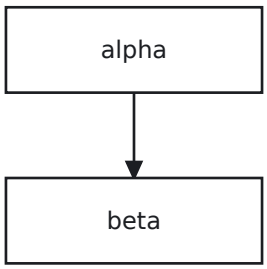
grid sets the measurements, and **unit** is the size everything else is a multiple of. A smaller one draws the same diagram smaller.

This is the only place in a diagram a length appears. Everything else is counted in boxes, steps and lines, so changing these numbers moves nothing you wrote.

SOURCE

```
grid { unit 4pt }
alpha -> beta
```

DIAGRAM



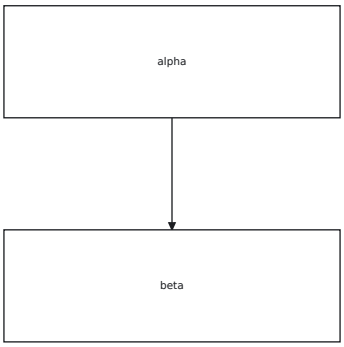
123 A larger unit

The same diagram again, with a bigger unit.

SOURCE

```
grid { unit 12pt }
alpha -> beta
```

DIAGRAM



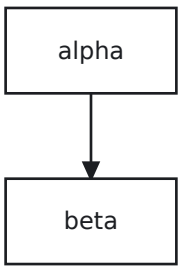
124 A narrower box

cell is how many units wide and tall a box is. The first number is the width.

SOURCE

```
grid { cell 16 x 8 }
alpha -> beta
```

DIAGRAM



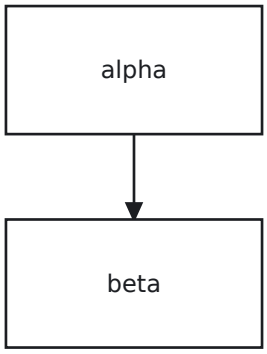
125 A taller box

The second number is the height.

SOURCE

```
grid { cell 24 x 12 }
alpha -> beta
```

DIAGRAM



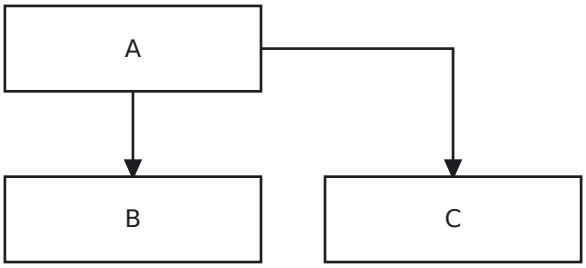
126 More room across

gutter is the gap between boxes side by side. Every gap across the diagram is the same, so changing it moves everything.

SOURCE

```
grid { gutter 6 }
box a "A"
box b "B"
box c "C"
a -> b
a -> c
```

DIAGRAM



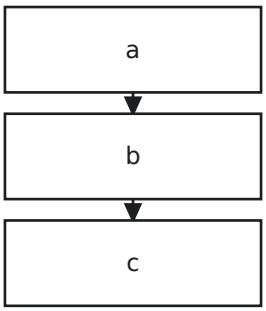
127 Less room between steps

rhythm is the gap between one step and the next.

SOURCE

```
grid { rhythm 2 }
a -> b -> c
```

DIAGRAM



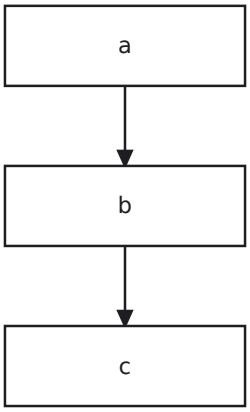
128 More room between steps

The same diagram, with the steps further apart.

SOURCE

```
grid { rhythm 8 }
a -> b -> c
```

DIAGRAM



129 A larger typeface

face names the typeface and its size. Boxes are measured against it, so larger words can mean wider boxes.

SOURCE

```
grid { face "DejaVu Sans" 12pt }
box alpha "Receive order"
```

DIAGRAM

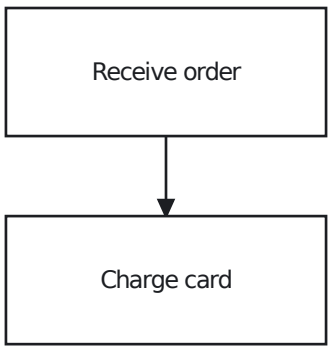
130 All of it at once

Every value in one block. Set as few of them as you like; the rest keep their usual measurements.

SOURCE

```
grid { unit 6pt; cell 20 x 8; gutter 4; rhythm 5; face "DejaVu Sans" 8pt }
box alpha "Receive order"
box beta "Charge card"
alpha -> beta
```

DIAGRAM



Whole diagrams, each using most of what the book has shown. Read them for how the statements sit together in a real file.

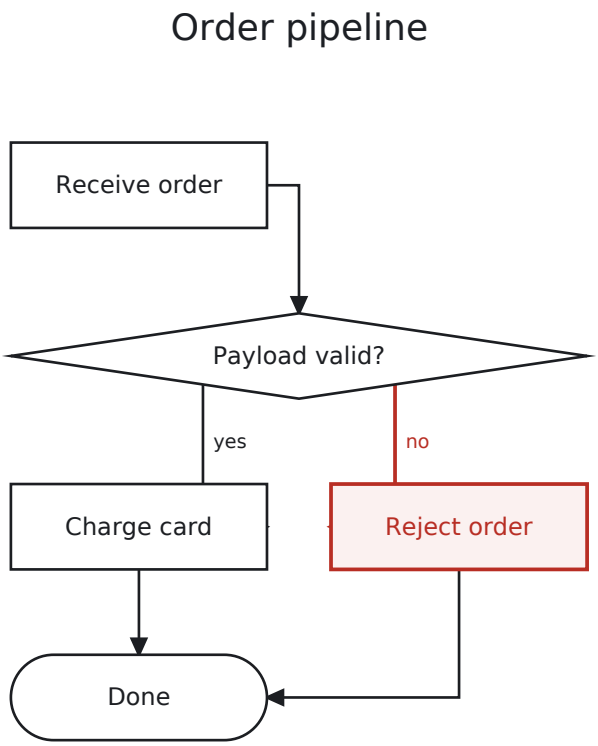
131 An order pipeline

A question, two answers, and a straight path through the middle.

SOURCE

```
diagram "Order pipeline" {
  flow down
}

box    receive "Receive order"
choice valid   "Payload valid?"
box    charge  "Charge card"
box    reject  "Reject order"
end    done    "Done"

receive -> valid
valid   -> charge : yes
valid   -> reject : no
charge  -> done
reject  -> done

sameline receive valid charge done
samestep charge reject
order valid: charge reject
role  error { reject, valid -> reject }
```

DIAGRAM



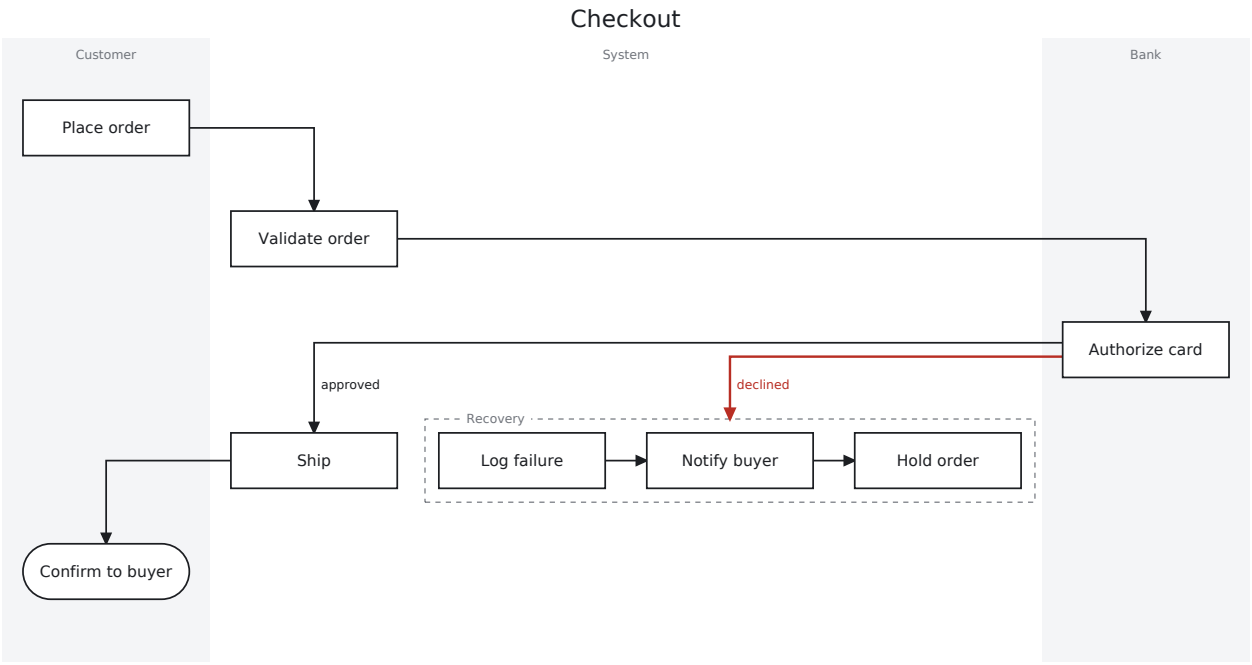
132 Checkout

Lanes for who does what, and a group that runs its own way inside one of them.

SOURCE

```
diagram "Checkout" {
  flow down
}

box place     "Place order"
box validate  "Validate order"
box authorize "Authorize card"
box ship      "Ship"
end confirm   "Confirm to buyer"

box log       "Log failure"
box notify    "Notify buyer"
box hold      "Hold order"

place -> validate -> authorize
authorize -> ship : approved
authorize -> recovery : declined
ship -> confirm
log -> notify -> hold

group recovery "Recovery" { flow right; log notify hold }

lane customer "Customer" { place confirm }
lane system   "System"   { validate ship recovery }
lane bank     "Bank"     { authorize }

role error { recovery, authorize -> recovery }
```

DIAGRAM



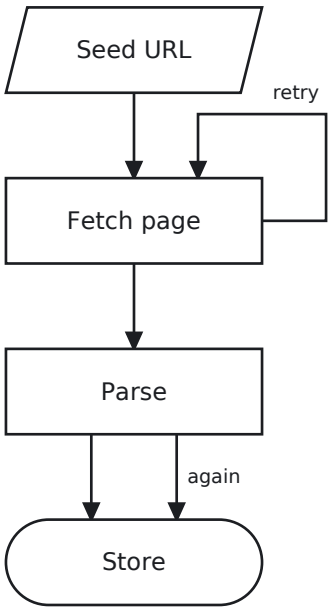
133 A crawler

A box that retries itself, and two arrows between the same pair of boxes.

SOURCE

```
diagram "Crawl" { flow down }
io  seed  "Seed URL"
box fetch "Fetch page"
box parse "Parse"
end store "Store"
seed -> fetch
fetch -> fetch : retry
fetch -> parse
parse -> store
parse -> store : again
```

DIAGRAM



134 Two straight paths

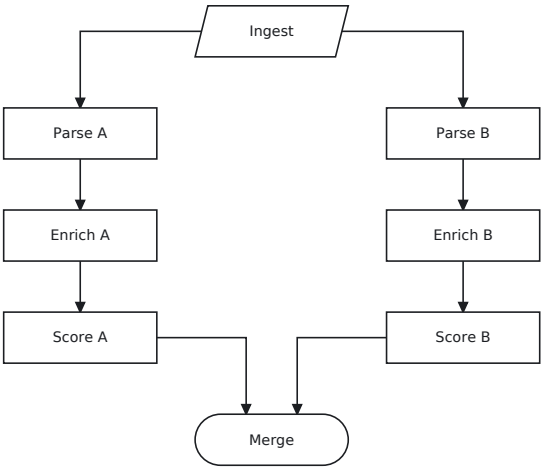
Two runs of boxes with none in common, held two lines apart, with the box they leave and the box they meet at both placed by name.

SOURCE

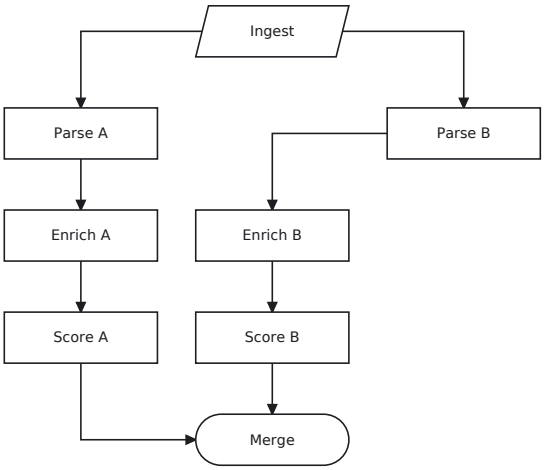
```
io  ingest "Ingest"
box pa "Parse A"
box ea "Enrich A"
box sa "Score A"
box pb "Parse B"
box eb "Enrich B"
box sb "Score B"
end merge "Merge"
ingest -> pa -> ea -> sa -> merge
ingest -> pb -> eb -> sb -> merge
sameline pa ea sa
sameline pb eb sb
apart pa pb: 2 lines
between pa pb: ingest
between pa pb: merge
```

DIAGRAM

as written



without sameline pb eb sb



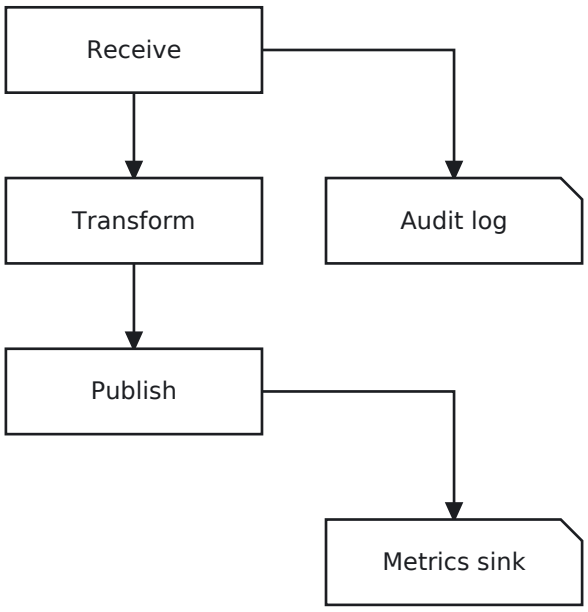
135 Lining up boxes that never meet

The two boxes on the right have no arrow between them and line up because **sameline** names them.

SOURCE

```
box receive "Receive"
box transform "Transform"
box publish "Publish"
note audit "Audit log"
note metrics "Metrics sink"
receive -> transform -> publish
receive -> audit
publish -> metrics
sameline audit metrics
```

DIAGRAM



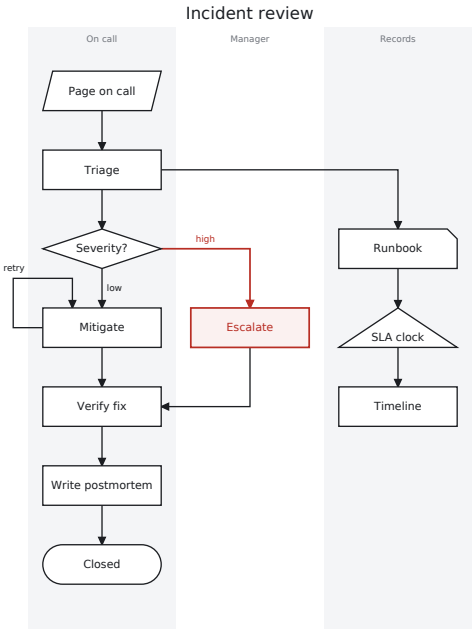
136 An incident review

Every shape, lanes for who does what, a straight path, a loop and a role.

SOURCE

```
diagram "Incident review" { flow down }

io     page    "Page on call"
box    triage  "Triage"
choice sev     "Severity?"
box    mitigate "Mitigate"
box    escalate "Escalate"
box    verify  "Verify fix"
box    postmortem "Write postmortem"
end    closed  "Closed"

note   runbook "Runbook"
mark   clock   "SLA clock"
box    timeline "Timeline"

page -> triage -> sev
sev -> mitigate : low
sev -> escalate : high
mitigate -> verify
escalate -> verify
verify -> postmortem -> closed
triage -> runbook
runbook -> clock -> timeline
mitigate -> mitigate : retry

sameline page triage sev mitigate verify postmortem closed
samestep mitigate escalate
sameline runbook clock timeline
order sev: mitigate escalate

lane oncall  "On call"  { page triage sev mitigate verify postmortem closed }
lane manager "Manager"  { escalate }
lane records "Records"  { runbook clock timeline }

role error { escalate, sev -> escalate }
```

DIAGRAM



Every statement

Every statement, one line each. The book teaches these in order; this page is for looking one up afterwards. Each line shows what you write, in full, and then what it does.

Drawing something

`box a "Words"`	A box. The words are optional, and without them the box is empty.
`box a "One" "Two"`	The same box over two lines. Each set of words is one line, and quad breaks a line further only if it still does not fit.
`a`	A name on its own is a box labeled with its own name. So is a name that appears only in an arrow.
`choice a "Words?"`	A diamond, for a question with more than one answer coming out of it.
`end a "Words"`	A rounded box, for where a diagram stops.
`io a "Words"`	A slanted box, for something going in or coming out.
`note a "Words"`	A box with a folded corner, for a remark rather than a step.
`mark a "Words"`	A triangle, for drawing attention to something.
`kind k box spans [1, 2]`	A new kind of box: the shape it takes, and how wide it may grow. Any of the six shapes above.

Arrows

`a -> b`	An arrow from **a** to **b**. Either name may be one you have not declared, and the box appears.
`a -> b : yes`	The same arrow with a word on it.
`a -> b -> c`	Two arrows, written as a chain.
`a --> b`	Dashed.
`a <-> b`	A head at both ends.
`a -> a`	From a box back to itself, drawn as a loop on one corner.
`a ~> b`	Allowed to take any route, for the one that would otherwise go the long way round.
`edge e a -> b`	Names an arrow, so **role** can pick it out.
`a.left -> b`	Made to leave by the left side of **a**.
`a.right -> b`	By the right side.
`a.out -> b`	By the side arrows leave from.
`a -> b.in`	Made to arrive at the side arrows come in on. Either end may name its side.

Saying what lines up

`sameline a b c`	These boxes sit on one line in the direction the diagram runs.
`samestep a b c`	These boxes sit in the same step, whatever the arrows would have done.
`order p: a b c`	The order of what comes out of **p**, left to right.
`side x left`	**x** and everything after it falls on the left of the first **sameline**.
`side x right`	On the right.
`apart a b: 2 lines`	Exactly how many lines sit between these two.
`between a b: x`	**x** sits at the middle of what the boxes before the colon cover.
`covers x: a b c`	**x** is as wide as the boxes named, without you writing how many.

Grouping and coloring

`group g "Words" { a b }`	A region drawn around these, placed as one thing. Groups may hold groups.
`group g "Words" { flow up; a b }`	The same, running its own way inside while the diagram places it as one thing.
`lane l "Words" { a b }`	A band behind these, running the length of the diagram.
`role r { a, b -> c }`	These boxes and arrows play role **r**. A theme says what that looks like.

The diagram itself

`diagram "Words"`	The name, set larger in a band at the head of the page.
`diagram "Words" { flow down }`	The name, and which way the diagram runs.
`flow down`	It runs down the page. Inside a **group** block it sets that group instead.
`flow right`	It runs across the page.
`flow up`	It runs up the page.
`flow left`	It runs across the page, the other way.
`grid { unit 4pt }`	The size everything else is a whole number of. The only place a length is written.
`grid { cell 24 x 8 }`	How wide and how deep one box is, counted in units.
`grid { gutter 6 }`	The gap between boxes side by side.
`grid { rhythm 8 }`	The gap between one step and the next.
`grid { face "DejaVu Sans" 9pt }`	The face the words are set in, and the size. Boxes are measured with it, and quad carries this one face.

The theme, a separate file

`--theme f.toml`	Hands the file below to any command. It holds what is true of every diagram in a project, so a source names none of it.
`[grid]`	Opens the measurements. A value left out keeps the built-in one; a value written wrongly is refused by name.
`unit = "4pt"`	The same five as the **grid** block above, written as TOML.
`cell = [24, 8]`	
`gutter = 6`	
`rhythm = 8`	
`face = "DejaVu Sans"`	
`size = "9pt"`	The type size. Naming it scales the three below unless you name those too.
`leading = "11.25pt"`	Line to line.
`label = "7.2pt"`	The size a word on an arrow is set at.
`title = "13.5pt"`	The size the diagram's name is set at.
`[style]`	Opens how the drawing is made, apart from the ink a role decides.
`lane = "#f4f5f7"`	The shade behind a lane band. Bands alternate between it and the page.
`page = "#ffffff"`	The page itself, which is also what a word on an arrow is set on.
`group = "#71757c"`	The rule drawn round a group.
`rule = "0.7pt"`	How wide that rule is drawn.
`dash = "3pt"`	How long each of its dashes.
`loops = "square"`	A loop on one box turns three corners.
`loops = "round"`	It is three quarters of a circle instead.
`corners = 100`	Every arrow turns a square corner, which is the house value.
`corners = 0`	Every arrow that turns is a curve with no straight part left. Anything between the two keeps that share of each corner.

`[role.default]`	Opens the ink everything falls back to.
`stroke = "#1c1e22"`	The line round a box, and the arrows.
`fill = "#ffffff"`	Inside a box.
`text = "#1c1e22"`	The words.
`width = "1pt"`	How wide a line is drawn.
`dash = "4pt"`	How long each dash, where something is dashed.
`head = "6pt"`	How long an arrowhead is.
`dashed = false`	Dashes the outlines and the arrows alike.
`[role.error]`	Opens the ink for one role. Any name you invent in a **role** statement can have a table here, and it keeps what it does not name.

Running it

`quadc build f.quad -o f.svg`	Draws it.
`quadc build f.quad -f pdf -o f.pdf`	Draws it as PDF.
`quadc build f.quad -f tikz -o f.tex`	As TikZ, to set in a LaTeX document.
`quadc build f.quad -f text -o f.txt`	As box-drawing characters, to read in a terminal.
`quadc build f.quad -f json -o f.json`	As the finished geometry, for a tool that draws it itself.
`quadc fmt f.quad`	Writes the file back in its canonical form.
`quadc lint f.quad`	Counts what is worth reducing.
`quadc suggest f.quad`	Offers an order that crosses fewer arrows.
`quadc why f.quad a`	Says which statement decided where box **a** sits.