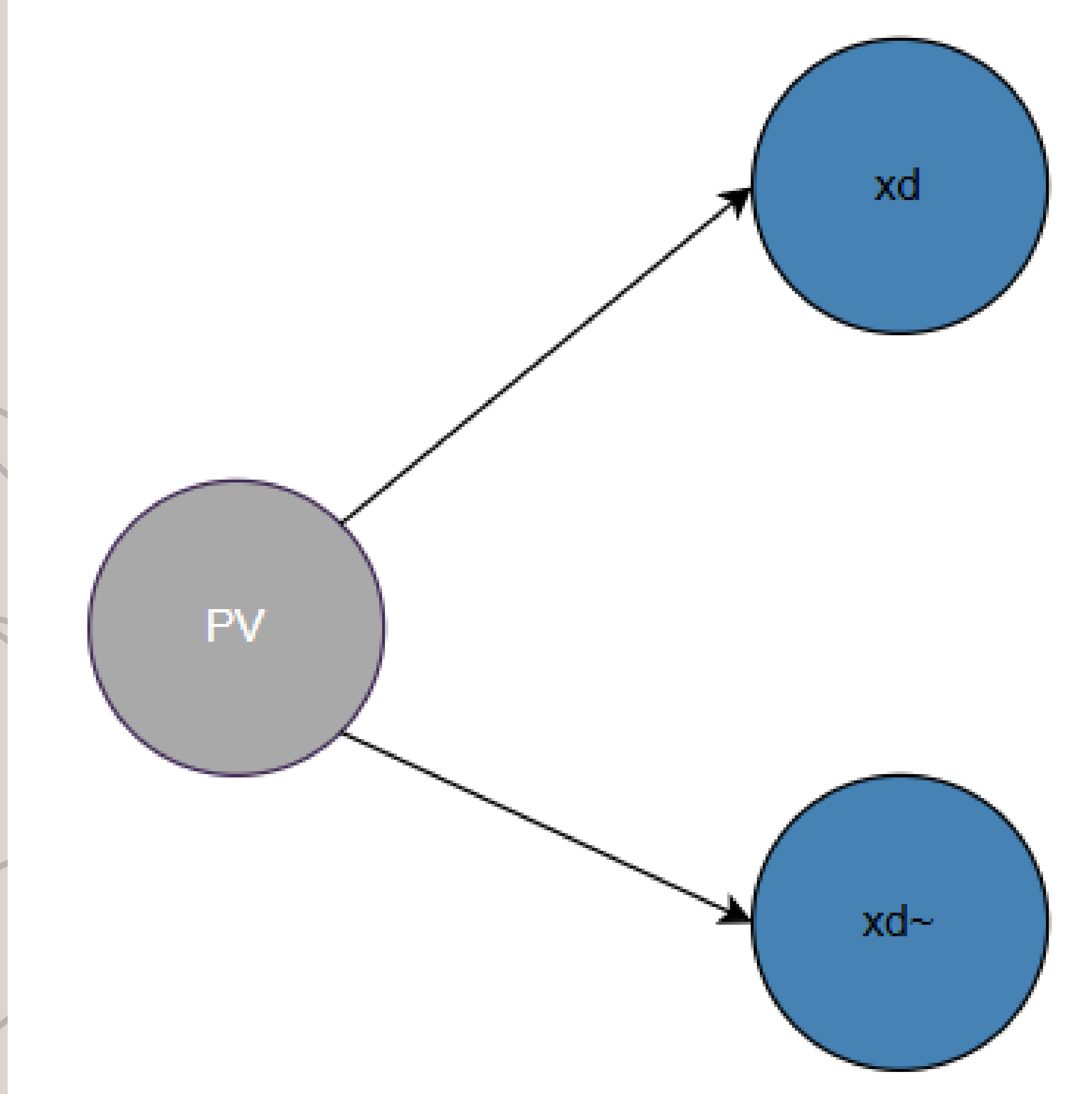
Reconstructing PV information in SModelS: a minimal proof of concept

by Lucas Magno D. Ramos
(IFUSP - University of São Paulo)

SModelS Fest 24 Workshop - Dec 18th, 2024

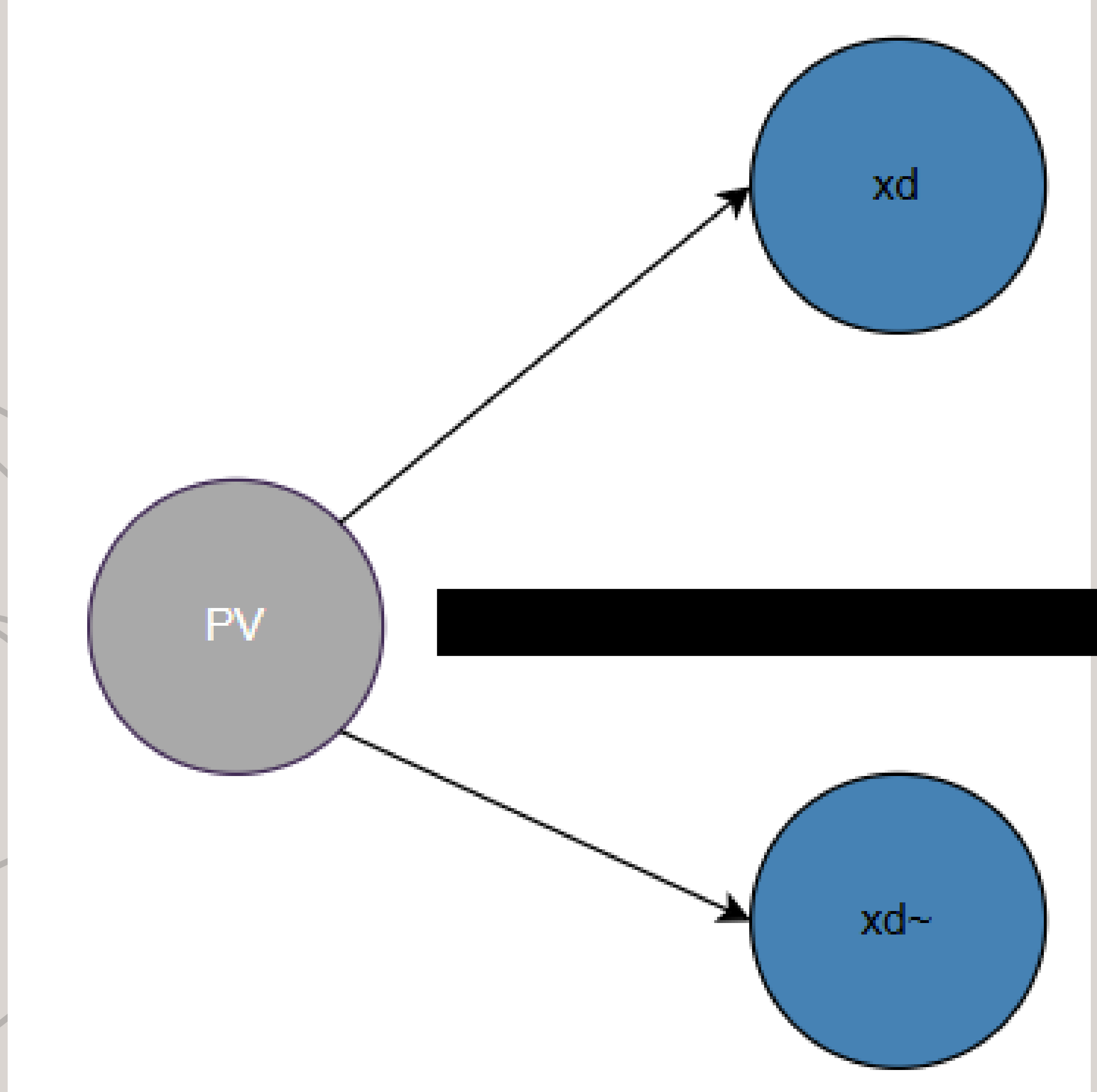
Some motivation...

Under the current paradigm, the PV in SModelS is a placeholder:



Some motivation...

Under the current paradigm, the PV in SModelS is a placeholder:

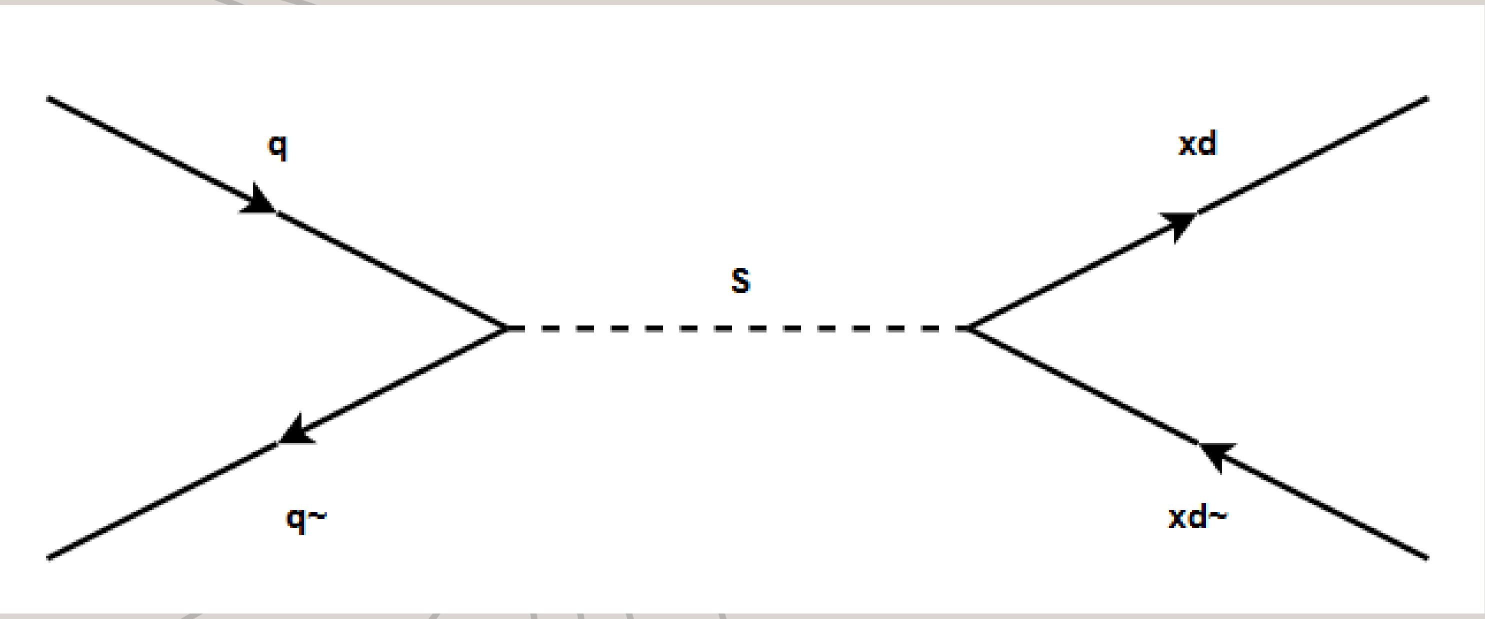


Could be an s-channel or a t-channel

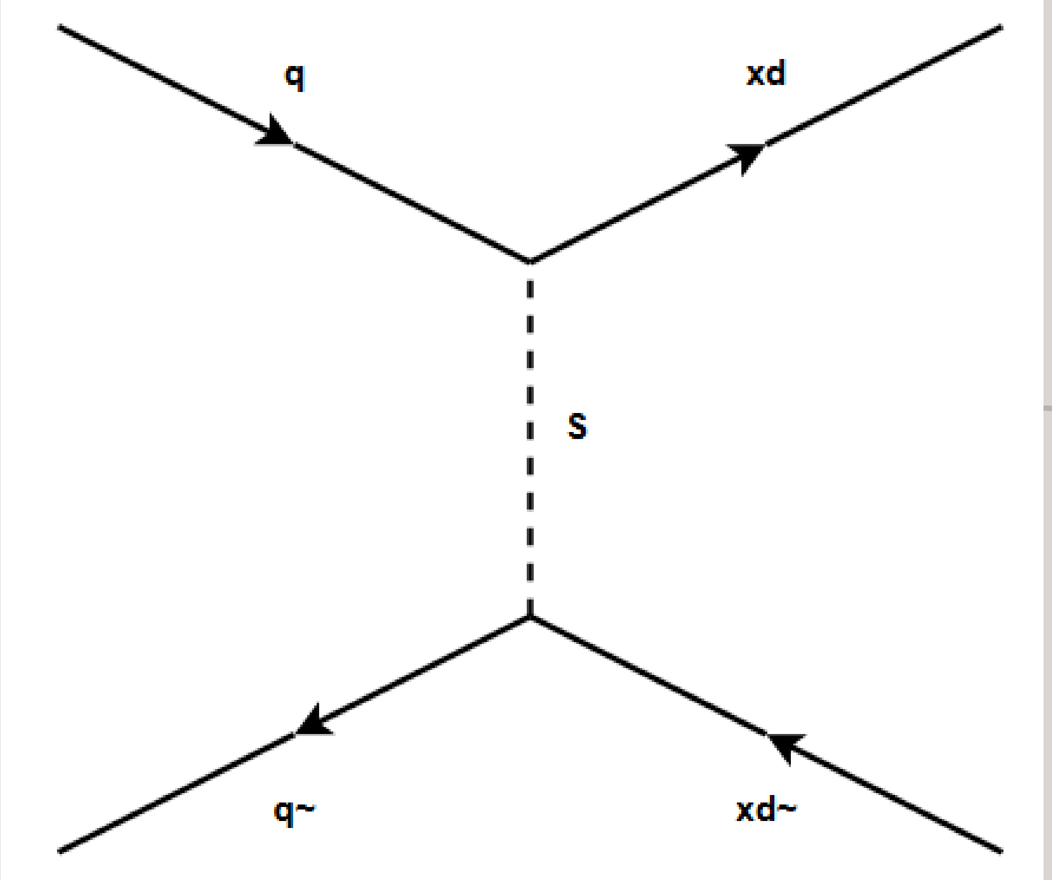
Some motivation...

This could give very useful information concerning dependence of final state distributions on model parameters such as the masses of mediators for dominant channels!

ie., some models might generate this:



But not this!



What do we already have?

All info parsed from the SLHA input file:

- SM and BSM particles in the model;
- Decays and Branching ratios;
- SM quantum numbers;
- (Some) cross sections (usually p p > BSM or BSM BSM);
- Other quantum numbers/properties via blocks.

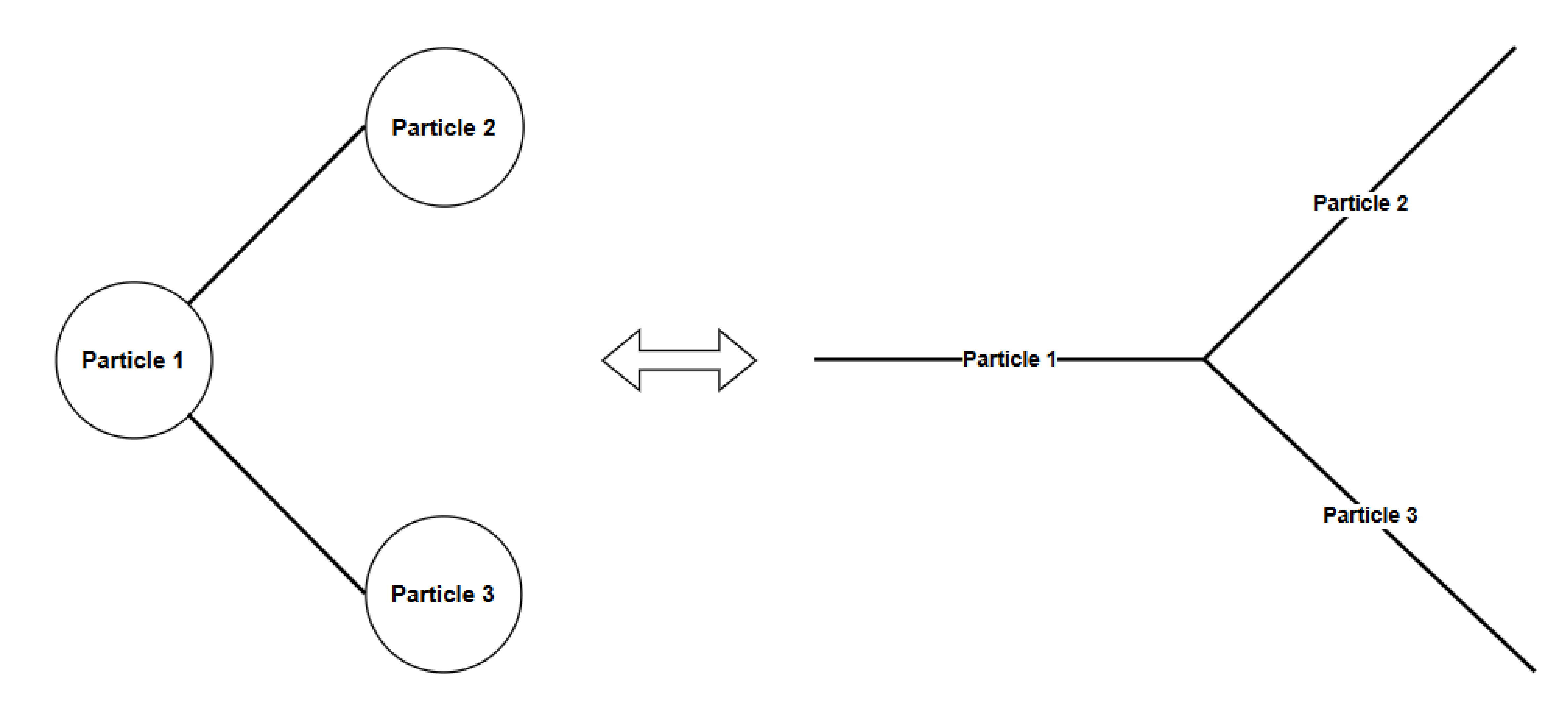
What can we do with it?

Quite a lot, actually!

- The decay lists can be used to reconstruct basic vertices from the model
- Information on the charges can be used to construct other vertices that won't show up in decays (eg. gqq~)
- The cross sections determine the relevant final states to be built in 2-to-N processes
- Reconstructed vertices can be combined into full diagrams

Basic vertex reconstruction

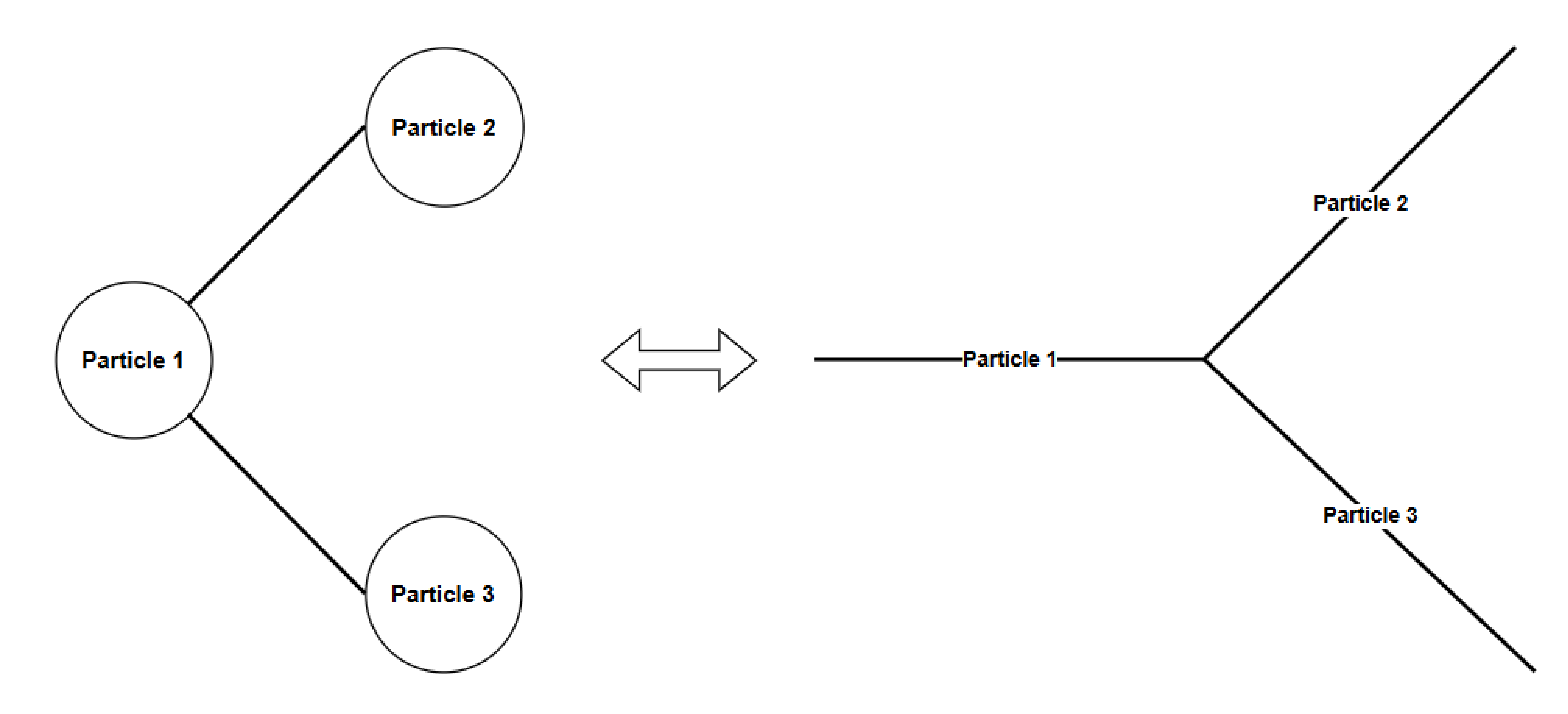
Basic association between Feynman diagram and SModelS graph representations:



This is exactly the decay P1 > (P2,P3) due to the vertex P1 P2 P3

Basic vertex reconstruction

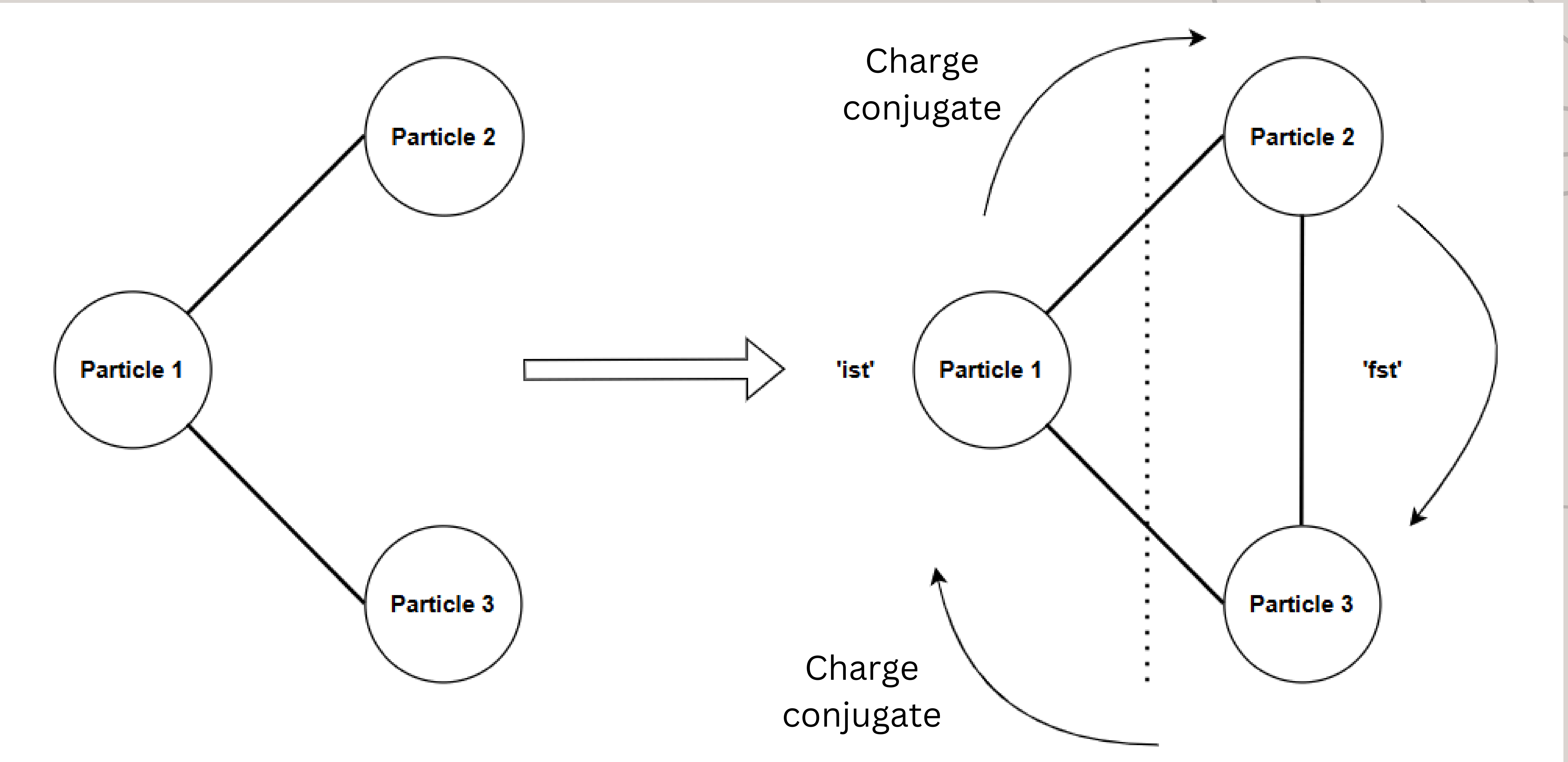
Basic association between Feynman diagram and SModelS graph representations:



While the decay has a well-defined direction (parent->daughters), the vertex can appear with any orientation!

Basic vertex reconstruction

Decay-vertex representation: build all vertices with external states from a single decay



Scattering diagrams generation

List all decays

↓

Produce all decay-vertices

↘

Generate all vertices

Scattering diagrams generation

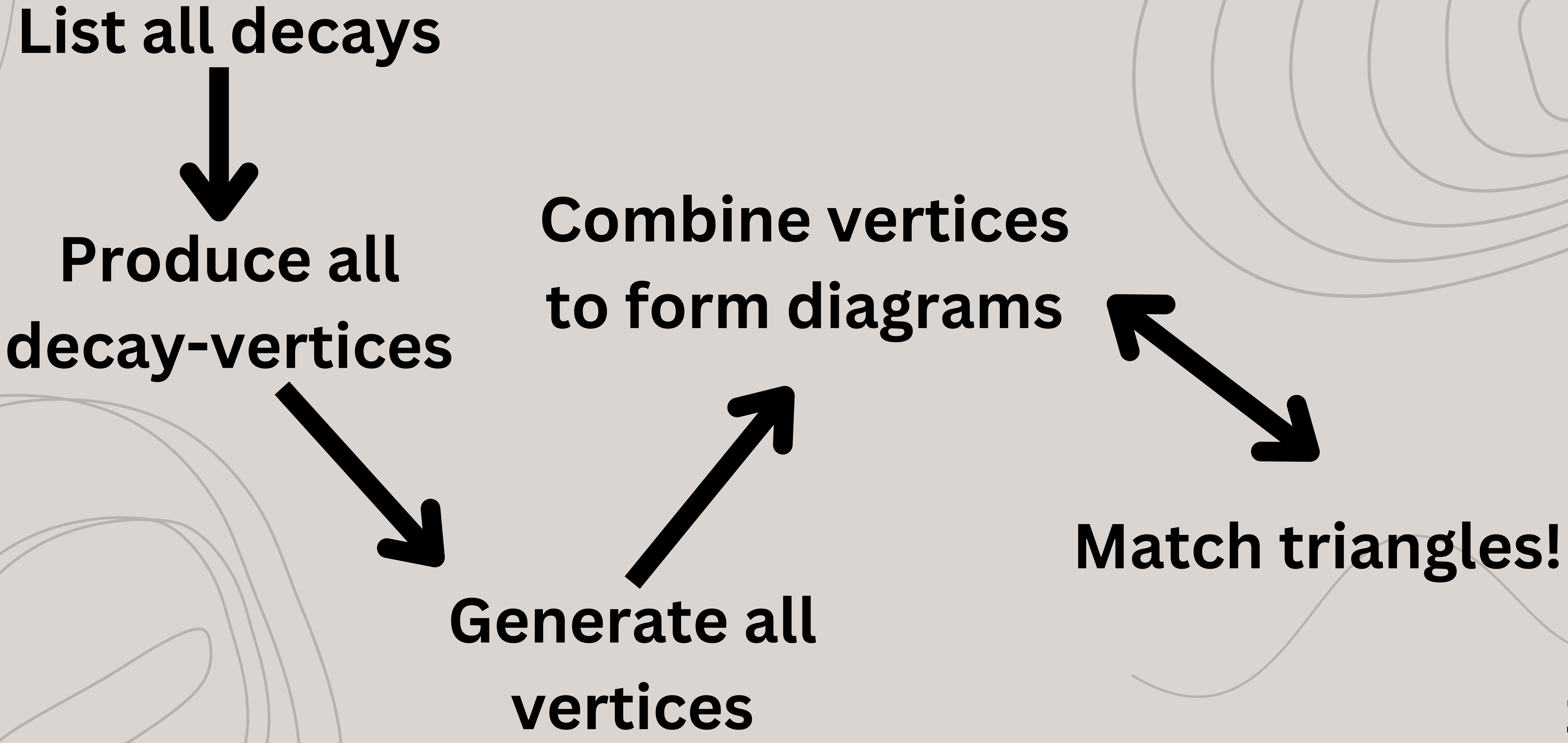
List all decays

Produce all decay-vertices

Generate all vertices

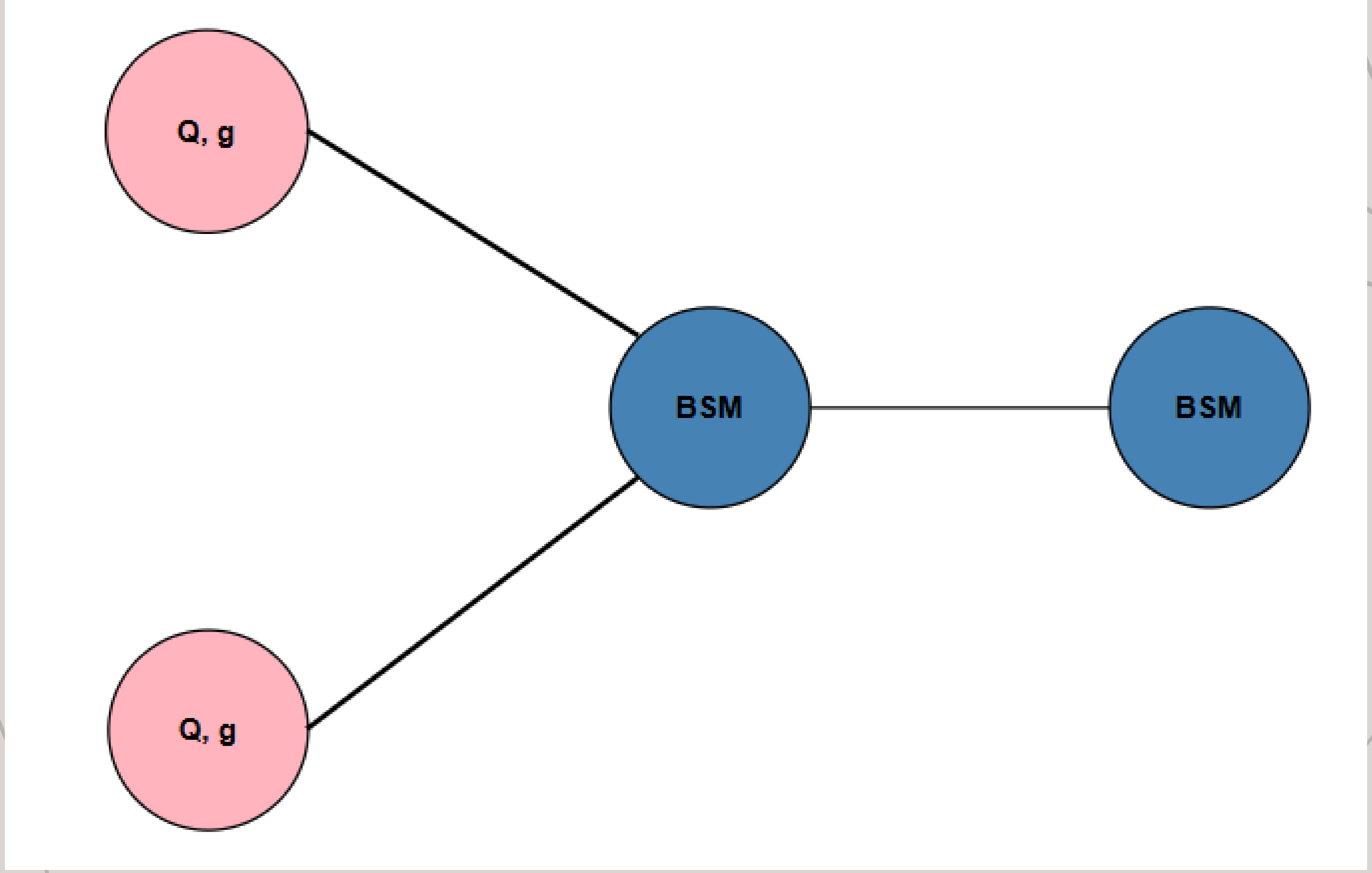
Combine vertices to form diagrams

Scattering diagrams generation



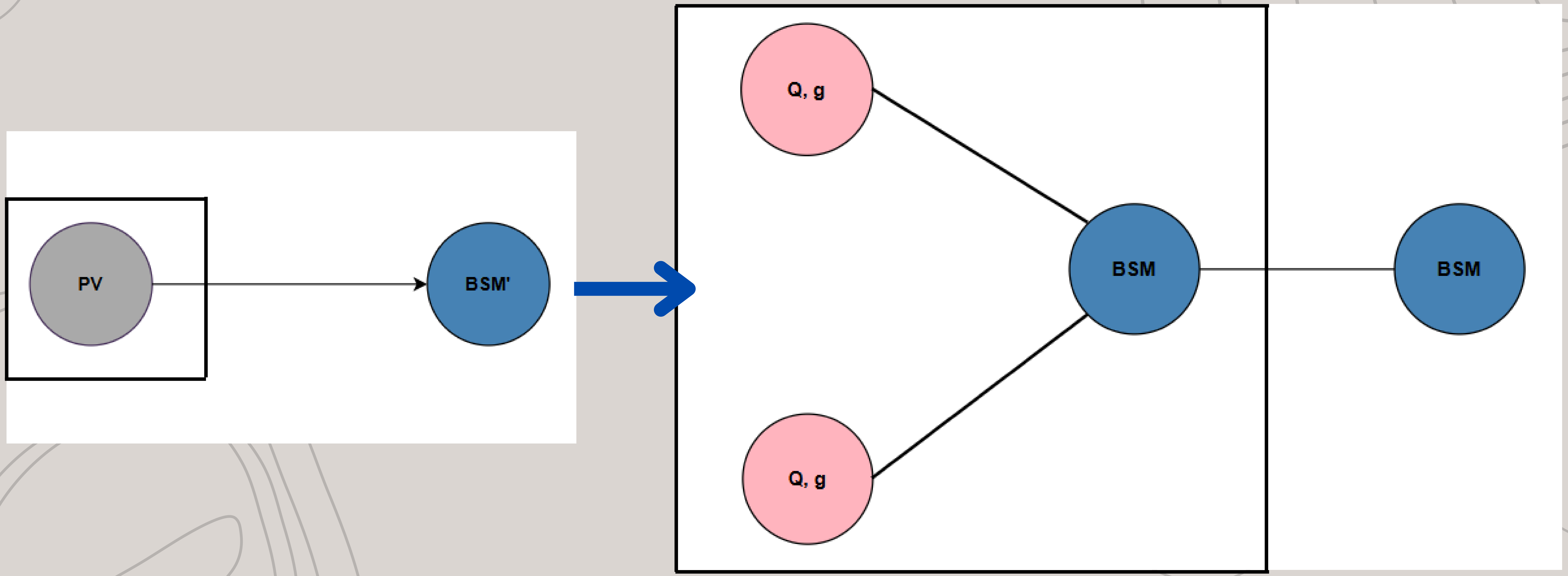
Scattering diagrams generation

2-to-1: Match initial state, and we’re done!



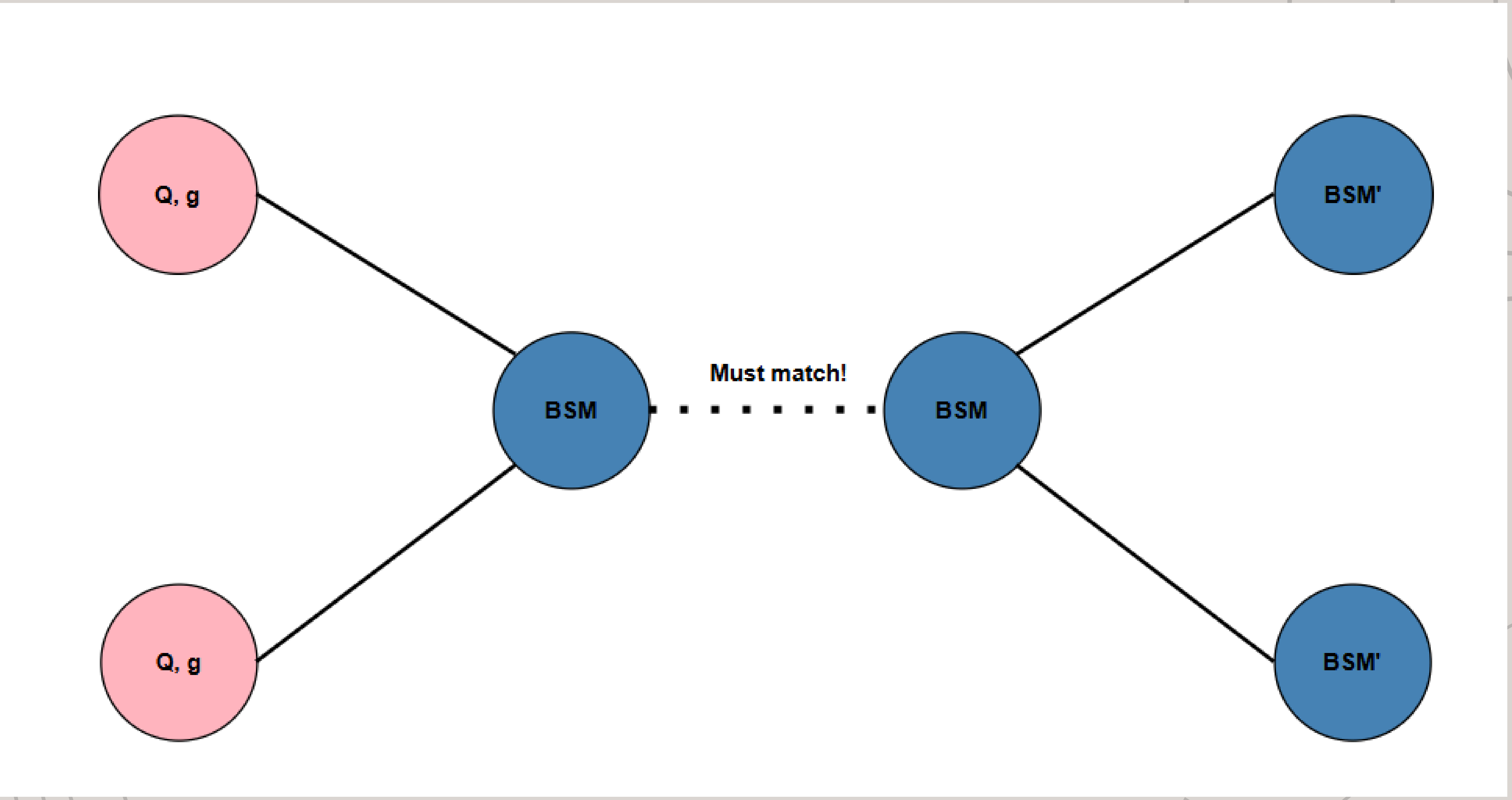
Scattering diagrams generation

2-to-1: Match initial state, and we're done!



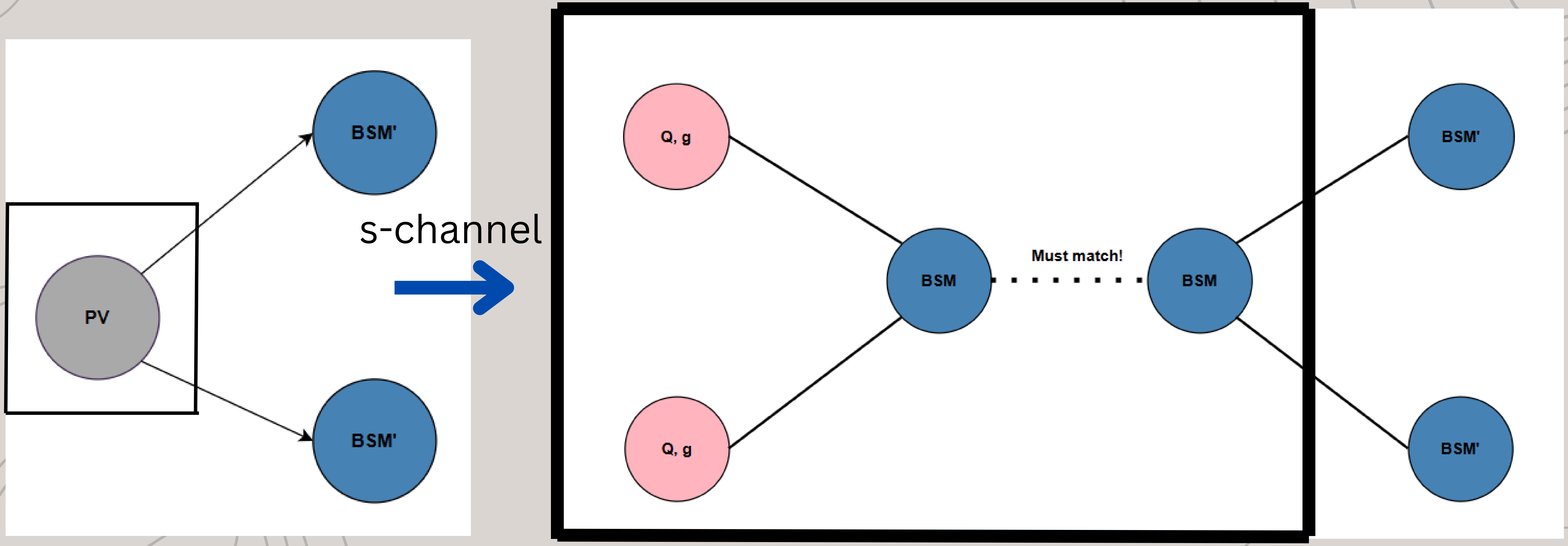
Scattering diagrams generation

2-to-2: Match initial state, and then find matching mediators



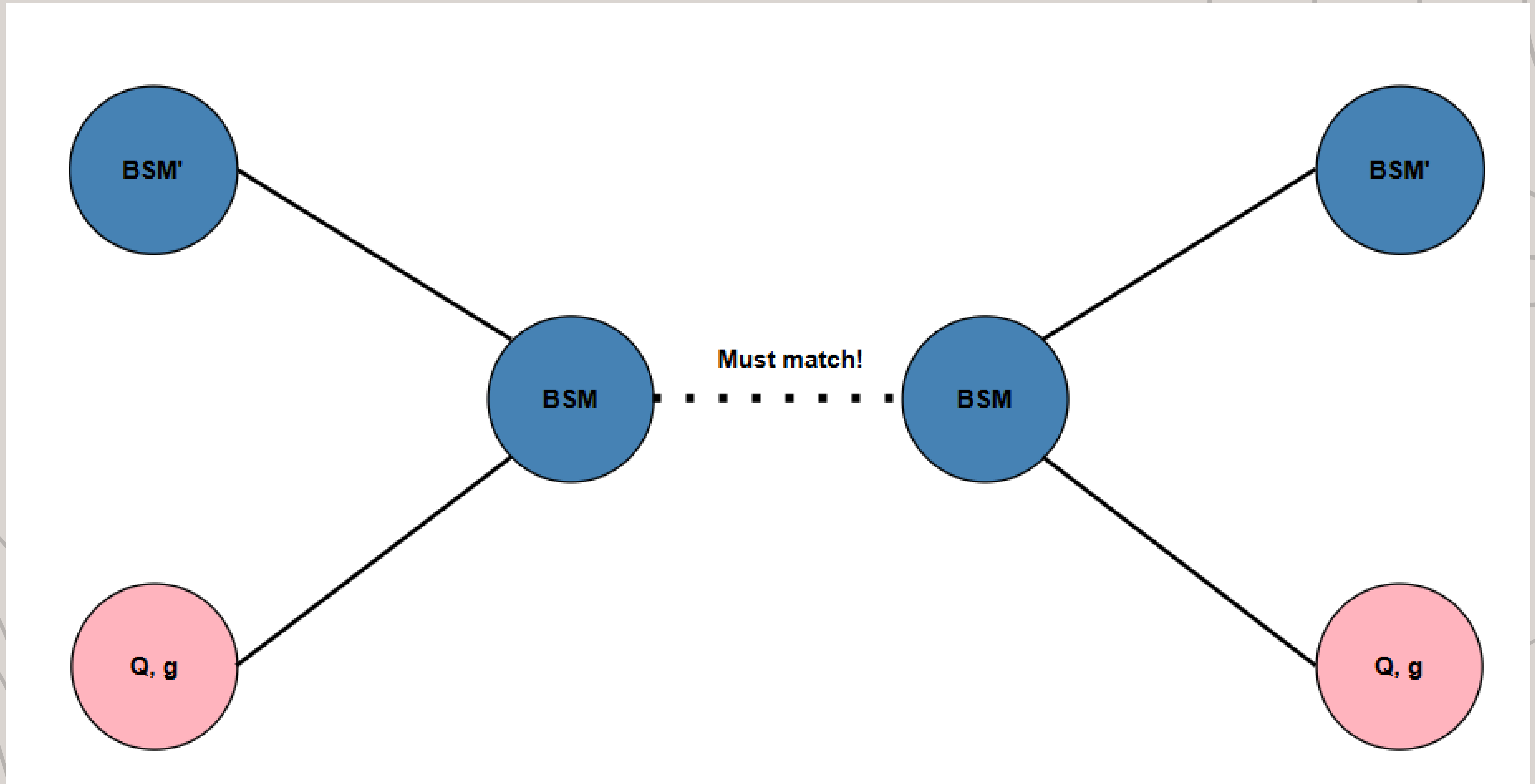
Scattering diagrams generation

2-to-2: Match initial state, and then find matching mediators



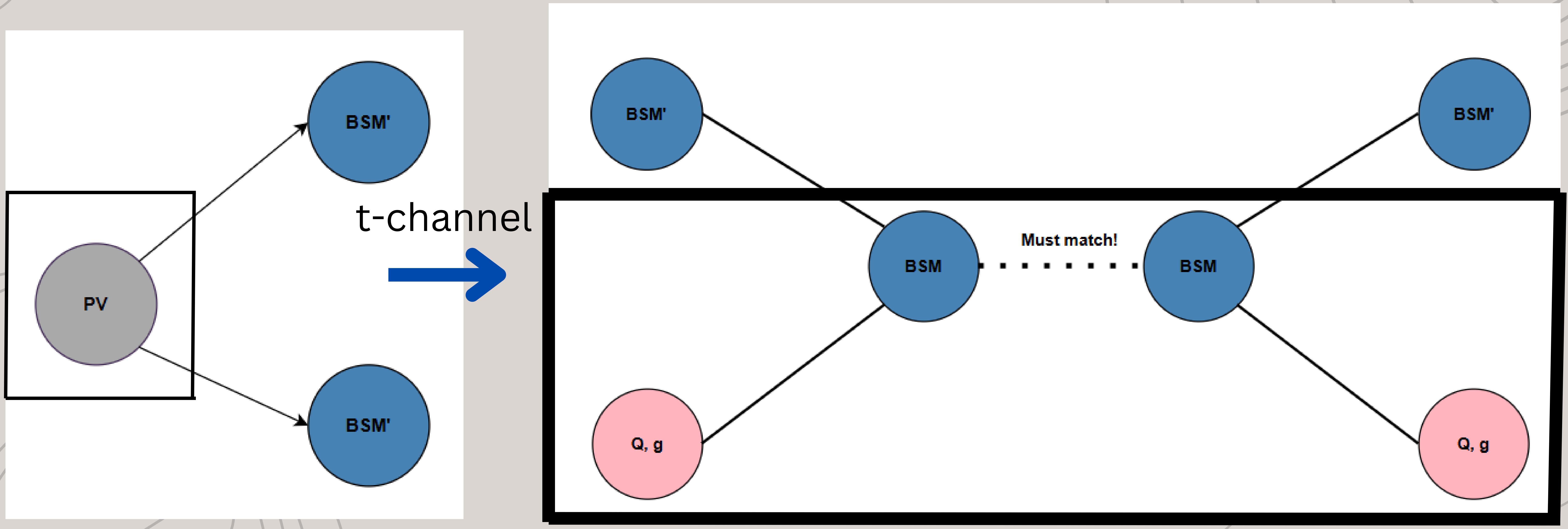
Scattering diagrams generation

2-to-2: Match initial state, and then find matching mediators



Scattering diagrams generation

2-to-2: Match initial state, and then find matching mediators



To do List

- Generalize to N-body decay lists
- Add SM vertices explicitly
- Add BSM vertices implied by charges
- Include 2-to-3 and 2-to-4 production

Thank you for the attention!

Backups

Some test models

TN1N1_tchannel.slha

- Contains ~d_L (squark), n1 (neutralino);
- Decay: ~d_L -> n1, d
- Cross section: p p > n1 n1

TRV1_1800_300_300 .slha

- Contains x_c, x_d (s/df DM), y1 (Z’)
- Decays: y1 -> (u,u~),(d,d~),(x_d,x_d~)
- Cross section: p p > y1

TRVS1_1800_300_300 .slha

- Contains same as previous
- Decays: Adds xc->(xd,xd),(g,g)
- Cross sections: p p > y1, p p > xc

Some code snippets

Step 1: Load Model

```
slhafile = 'inputFiles/slha/TRVS1_1800_300_300.slha'
#slhafile = 'inputFiles/slha/TRV1_1800_300_300.slha'
#slhafile = 'inputFiles/slha/TN1N1_tchannel.slha'
```

```
# Load the BSM model
runtime.modelFile = slhafile
BSMList = load()
model = Model(BSMparticles=BSMList, SMparticles=SMList)
model.updateParticles(inputFile=slhafile)
```

Some code snippets

Steps 2&3: Generate decay vertices + all vertices

```
def generate_vertices(model):
    decays = []
    for ptc in model.BSMparticles:
        decaylist = ptc.decays
        for decay in decaylist:
            decays.append([ptc.pdg,decay.ids])

    decay_vertices=[]
    for decay in decays:
        decay_vertices.append({'ist':normSelfConj_pdg(model,[decay[0]]),'fst':normSelfConj_pdg(model,decay[1])})

    vertices = {}
    for vert in decay_vertices:
        while vert['ist']:
            ptc = vert['ist'][0]
            vert = istTofst(vert,ptc,model)
        for part in vert['fst']:
                #vertices.append(fstToist(vert,part,model))
                #vertices[-1]['fst'].sort(key=lambda p: order_states(p,model),reverse=True)
                aux=fstToist(vert,part,model)
                aux['fst'].sort(key=lambda p: order_states(p,model),reverse=True)
                vertices[part,tuple(aux['fst'])]=aux.copy()

    return vertices
```

Some code snippets

Steps 4: Generate “initial” states

```
def build_proton_ists(model,vert_list):
    ists = []
    #temp definition of the proton particle content, change for something in a wider scope in future implementation
    proton = [1,-1,2,-2,3,-3,21]
    for vert in vert_list.values():
        if vert['fst'] in tuple([q1,q2] for q1 in proton for q2 in proton):
            ist_aux = vert['ist']
            fst_aux = vert['fst']
            vert_aux = {'ist':ist_aux.copy(),'fst':fst_aux.copy()}
            #^Very crude implementation for now to guarantee functionality and no cross-references, will refine later
            for ptc in ist_aux:
                vert_aux = istTofst(vert_aux,ptc,model)
            for ptc in fst_aux:
                vert_aux = fstToist(vert_aux,ptc,model)
            ists.append({'ist':vert_aux['ist'].copy(),'fst':vert_aux['fst'].copy()})
    return ists

def build_mixed_ists(model,vert_list):
    ists = []
    #temp definition of the proton particle content, change for something in a wider scope in future implementation
    proton = [1,-1,2,-2,3,-3,21]
    for vert in vert_list.values():
        if any(vert['ist'][0] == p.pdg for p in model.BSMparticles):
            for ptc in vert['fst']:
                if any(ptc == p for p in proton):
                    conj_vert = fstToist(vert,ptc,model)
                    ists.append({'ist':sorted(conj_vert['ist'],key=lambda p: order_states(p,model),reverse=True),'fst':sorte
                    #ists.append({'ist':conj_vert['fst'],'fst':conj_vert['ist']})
    return ists
```

```
testeists = build_proton_ists(model,allvertices)
for ist in testeists:
    print(ist)
```

✓ 0.1s

```
{'ist': [21, 21], 'fst': [-51]}
{'ist': [21, 21], 'fst': [51]}
{'ist': [-1, 1], 'fst': [55]}
{'ist': [-2, 2], 'fst': [55]}
```

```
testemists = build_mixed_ists(model,allvertices)
for ist in testemists:
    print(ist)
```

✓ 0.0s

```
{'ist': [51, 21], 'fst': [21]}
{'ist': [51, 21], 'fst': [21]}
{'ist': [-51, 21], 'fst': [21]}
{'ist': [-51, 21], 'fst': [21]}
{'ist': [55, -1], 'fst': [-1]}
{'ist': [55, 1], 'fst': [1]}
{'ist': [55, -2], 'fst': [-2]}
{'ist': [55, 2], 'fst': [2]}
```

Some code snippets

Step 5: Match “initial” states and vertices to form diagrams

```
halfchannel = build_proton_ists(model,vert_list)
for instates in halfchannel:
    for vert in vert_list.values():
        if vert['ist']==instates['fst']:
            #schannel.append({'ist':instates, 'fst':vert})
            schannel[tuple(instates['ist']),tuple(instates['fst']),tuple(vert['ist']),tuple(vert['fst'])]=
            {'ist':instates, 'fst':vert.copy()}
```

(eg. s-channel builder)

```
tto = build_2to1(model,allvertices)
ttt_s,ttt_t = build_2to2(model,allvertices)
print('2-to-1 processes: \n',tto,'\n')
print('s-channel 2-to-2 processes: \n',ttt_s,'\n')
print('t-channel 2-to-2 processes: \n', ttt_t)
```

Outputs many diagrams in a complicated format, but we can print them in a familiar form with the print_diagram function

Some code snippets

Step 6: Match final states with input cross sections

```
def generate_xsecs(model):
    fsts = []
    for xsec in model.xsections:
        fsts.append(xsec.pid)
    return fsts
```

```
xsecs = generate_xsecs(model)
dgms = {}
for xsec in xsecs:
    dgms[xsec] = []
    for process in tto.keys():
        if sorted(list(xsec)) == sorted(list(process[-1])):
            dgms[xsec].append(tto[process])
    for process in ttt_s.keys():
        if sorted(list(xsec)) == sorted(list(process[-1])):
            dgms[xsec].append(ttt_s[process])
    for process in ttt_t.keys():
        sort_xsec = sorted(list(xsec))
        sort_fst = sorted([process[0][0],process[-1][0]])
        if sort_xsec == sort_fst:
            dgms[xsec].append(ttt_t[process])

#print(dgms)
for xsec in model.xsections:
    print("Cross Section:",xsec.full_pid[0:2],"->",xsec.pid)
    for process in dgms[xsec.pid]:
        print_diagram(process)
```

Outputs