# The Black Rain

by J. Simon van der Walt

for amplified string quintet
with live computer processing

# The Black Rain

## Composer's note

> 'When the last trace of the rocket's presence, a whitish haze, had been absorbed by the atmosphere, when the wandering sandy waves gradually began to cover up the naked rock of the ground, at the same time filling in the deserted digging spaces – only then, much later, did a dark cloud gather in the west. Hovering low above the ground it pushed closer, grew, encircled the landing area with a threatening arm. There it remained, motionless.
>
> As the sun was about to set, a black rain fell on the desert.'

'The Black Rain' takes its title from the first chapter of Stanisław Lem's 1967 science fiction novel 'The Invincible', in which a mighty spaceship and her crew are overcome by a race of microscopic mechanical flies, individually insignificant, but capable of joining together into a vast quasi-intelligent 'cloud': surely one of the first fictional works to speculate on the possibilities of nanotechnology, calling to mind such devices as the nanostats which inhabit Neal Stephenson's 1995 novel 'The Diamond Age', and the EDust, or Everything Dust, in Iain M. Banks 2000 'Look to Windward'.

Aesthetically, 'The Black Rain' carries forward the composer's ongoing reconstruction of the career of his fictional alter ego Edward 'Teddy' Edwards. Something like:

> 'In 1959, Edwards created a work for string quartet (or quintet?) and five (or four?) taperecorders, incorporating radio equipment borrowed from Aldermaston, where he was at the time employed as an engineer on the ill-fated Blue Streak missile system. Working from his original sketches, I have replicated the piece using the music programming language SuperCollider, with the addition of a reconstructed lost (?) part for double bass.'

In terms of musical devices, 'The Black Rain' represents, through self-quotation, a critique of a group earlier works of mine ('smir', '4thought', '5lipside' etc), all of which float angular melodies across polymetric rhythmic frameworks, usually according to some quartal scheme, and usually, it would seem, in roughly the same key.

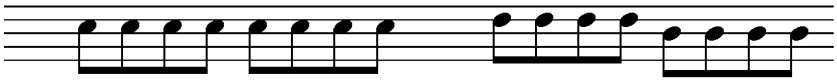
## Performance instructions

‘The Black Rain’ comprises two sections for the strings, both of which are accompanied by material from the computer, and a brief coda in the form of a recording of the composer’s spoken voice.

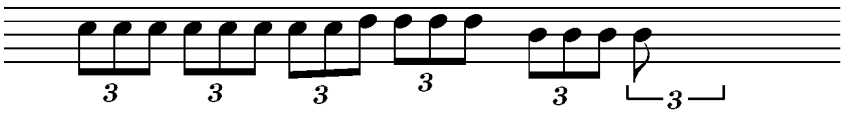
The first half of the piece is aleatoric in conception and minimalist in style. It is composed of two kinds of gesture – repeated notes and soaring phrases – from which the players make choices during the course of the piece. Each player should play *mostly* repeated notes, with occasional diversions to play any one or two of the soaring phrases. Ideally, no more than two soaring phrases should be heard at any one time, but always complete a phrase once you have started it. Soaring phrases need not start together, and will thus usually overlap in a quasi canonic fashion. However, if two players happen to catch each other’s eye and start playing (the same or different) soaring phrases together, that would be fine also.

Rhythmic variation must be introduced. Repeated notes are *usually* played in quavers at crotchet = 140 as notated. However, *occasionally*, an individual player may choose for a period to play in a different relationship to the underlying pulse, as shown:

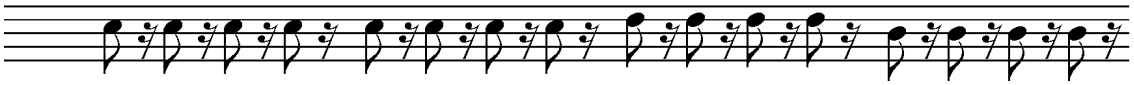
Usually:



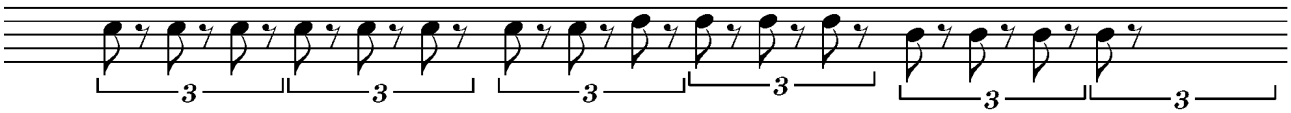
Occasionally:



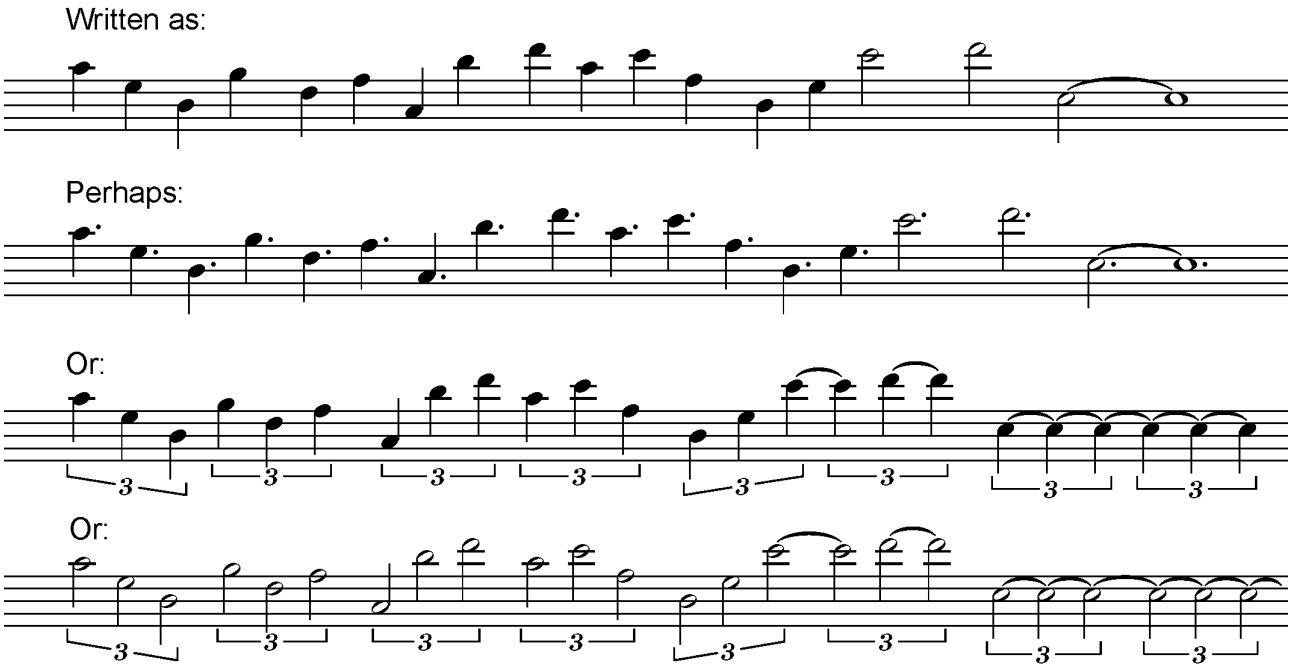

Or:



Or:




Soaring phrases should *seldom* be played in crotchets as written, but *usually* in some other metrical relationship:




The performance of this first section will be greatly complicated by the contribution of the computer, which is intentionally rather louder than the players themselves. Each player's material is subject to further rhythmic variation, by means of a slicing algorithm implemented in SuperCollider. The speed at which this slicing occurs goes through a sequence of tempo glissandi, departing from and returning to the basic crotchet = 140. The players must strive to keep a sense of this tempo among themselves, even when playing in a different metrical relationship. Furthermore, particularly when playing repeated notes, the players must not allow themselves to be seduced by the computer into making their own gradual changes of tempo. Always switch cleanly to a new metrical relationship with the underlying pulse.

After approximately four minutes, the computer will abruptly initiate the second section of the piece, which is built around the composer's waltz 'Slipping Away'. At this point, all players immediately stop what they are doing, in the middle of a phrase if need be, and turn to the waltz music. Play the given sequence of chromatically descending variations exactly as written, ignoring as best you can the occasional interjections (in the wrong key!) of the same material by the computer.

After a further three minutes (approximately), a recording of the composer's voice will be heard. Complete the variation you are on, plus one more, ending inconclusively at the fermata. Wait for the recording of the composer's voice to finish: the piece ends with the sound of a computer spacebar being hit.

## The Patch

*The Black Rain* utilises the file *theblackrain.rtf* authored in the music programming language SuperCollider. This patch is fully self-documented, and comes complete with the necessary dependencies: two audio files, and redFrik's extTempoClock.sc extension to TempoClock. It has been tested on SuperCollider 3.5 under Mac OS X 10.6.8.

Briefly: two audio files are loaded into buffers, these being a synthesised version of *Slipping Away* and a recording of the composer's voice. Seven Synths are defined:

*patbuf* is used to slice the recorded version of *Slipping Away*, *playthrough* does what it says, and five synths *yap0* - *yap4* are used to chop the live audio from the five instruments.

In the first section of the piece, five TempoClocks are established. The incoming audio is chopped at five different tempi, which undergo a series of metric modulations sequenced in a Routine. In the second section of the piece, two Pbinds are used to introduce random slices of a synthesised version of *Slipping Away*.

Each string player must be individually close miked, ideally with miniature clip-ons. Five channels of audio are routed into SuperCollider, and five channels of audio are returned, one for each instrument. These should ideally be routed to five individual speakers located adjacent to or behind each player: alternatively, they may be panned across a stereo or multi-channel image to correspond roughly to the position of the players.

## Duration

Approximately 8 mins.

**JSVDW**
Glasgow
31/03/2012

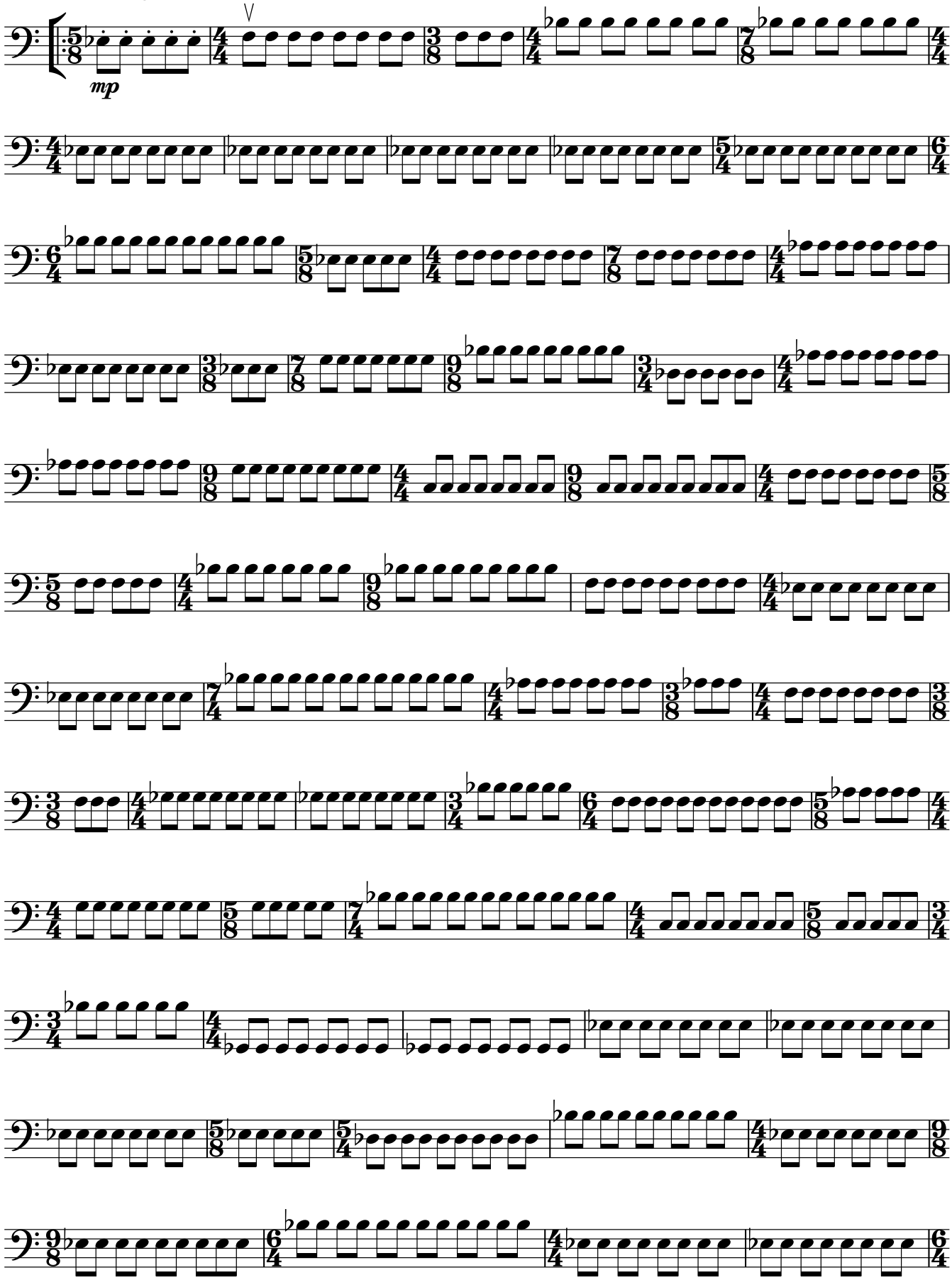
Bass

# The Black Rain

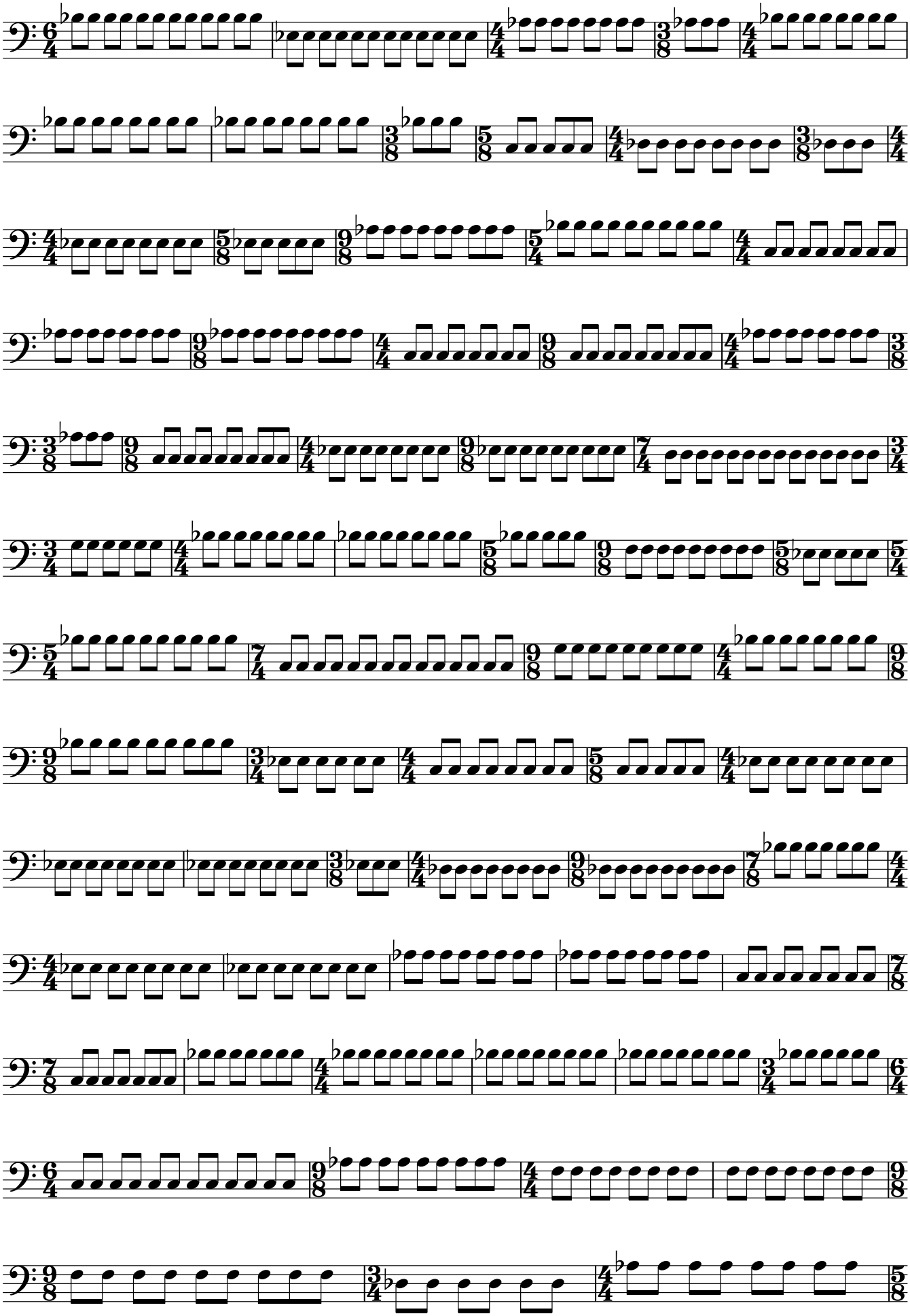
J. Simon van der Walt

♩ = 140

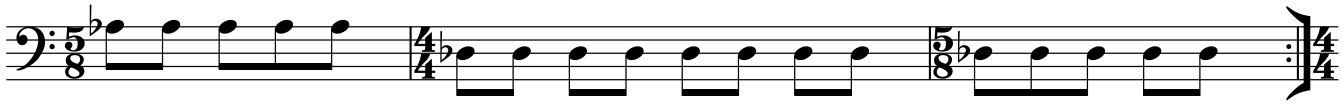
Start on any B♭: sempre stacc: alternate bowing always,
do not align downbows with barlines



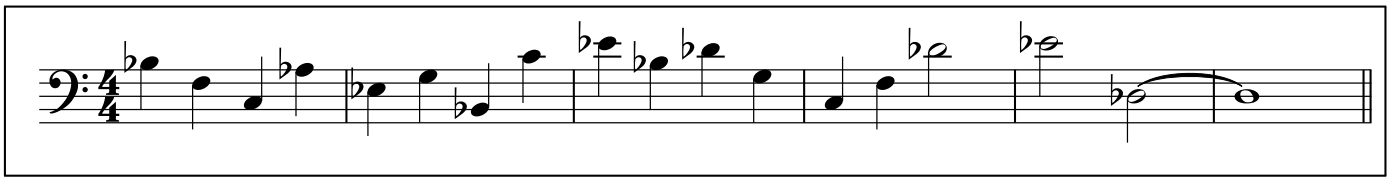
Bass

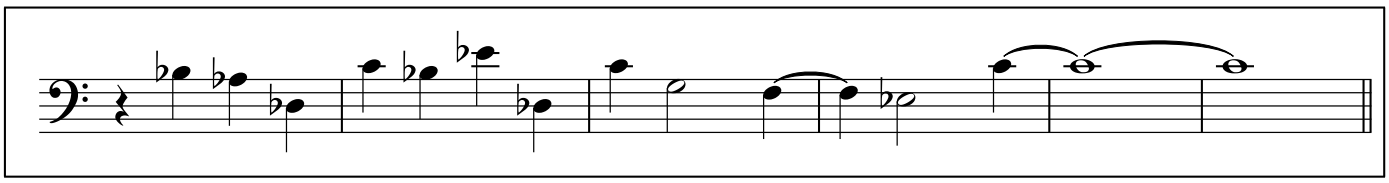


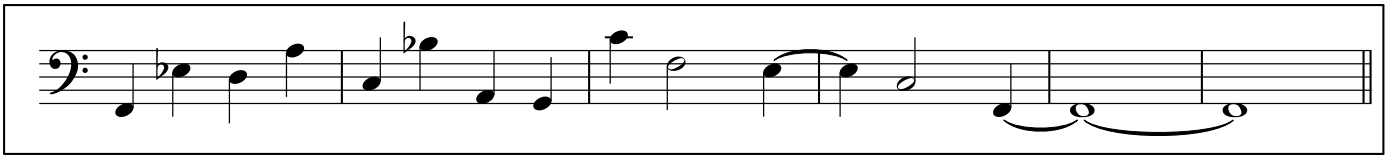
Bass

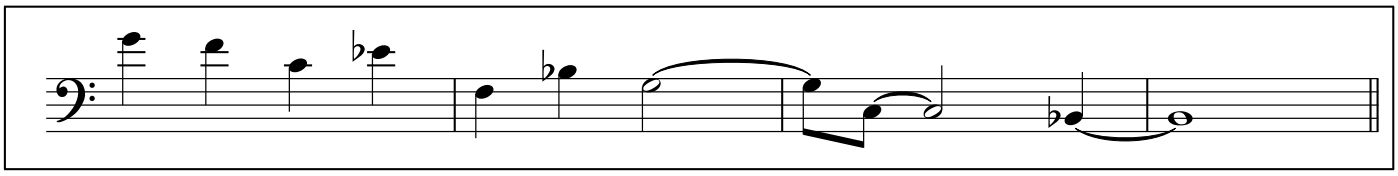


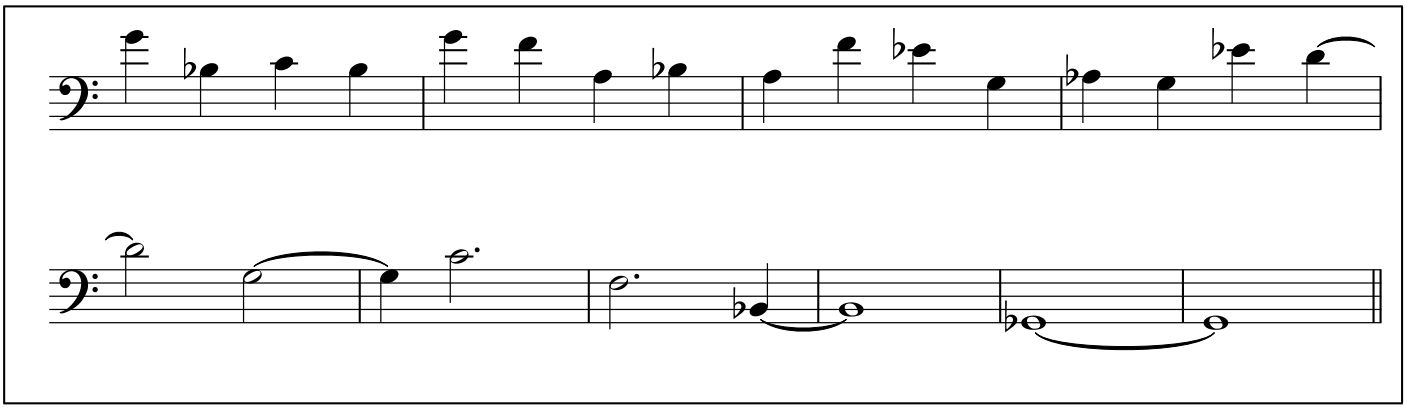
legato, in relievo

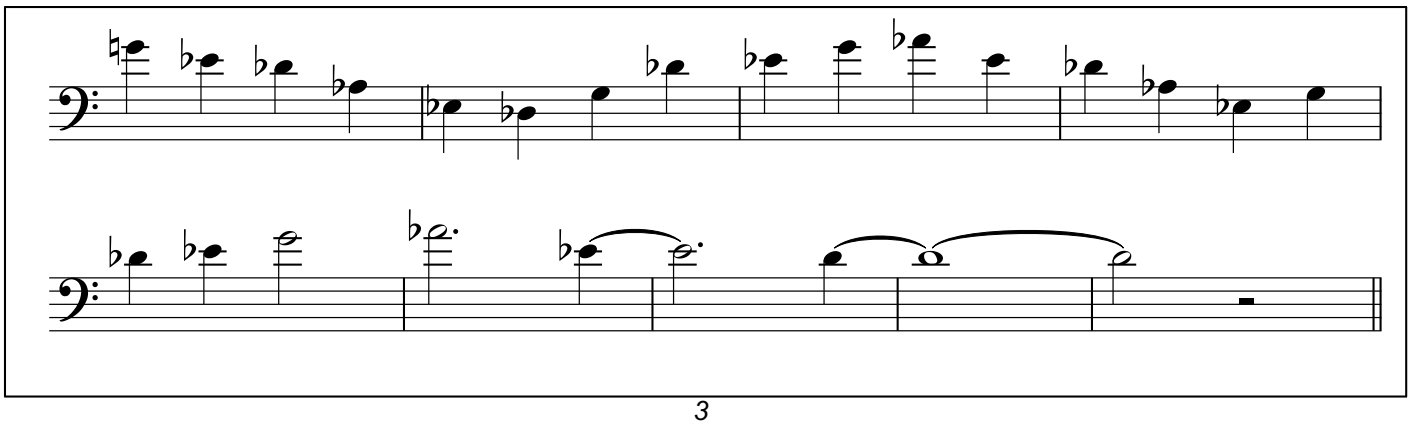












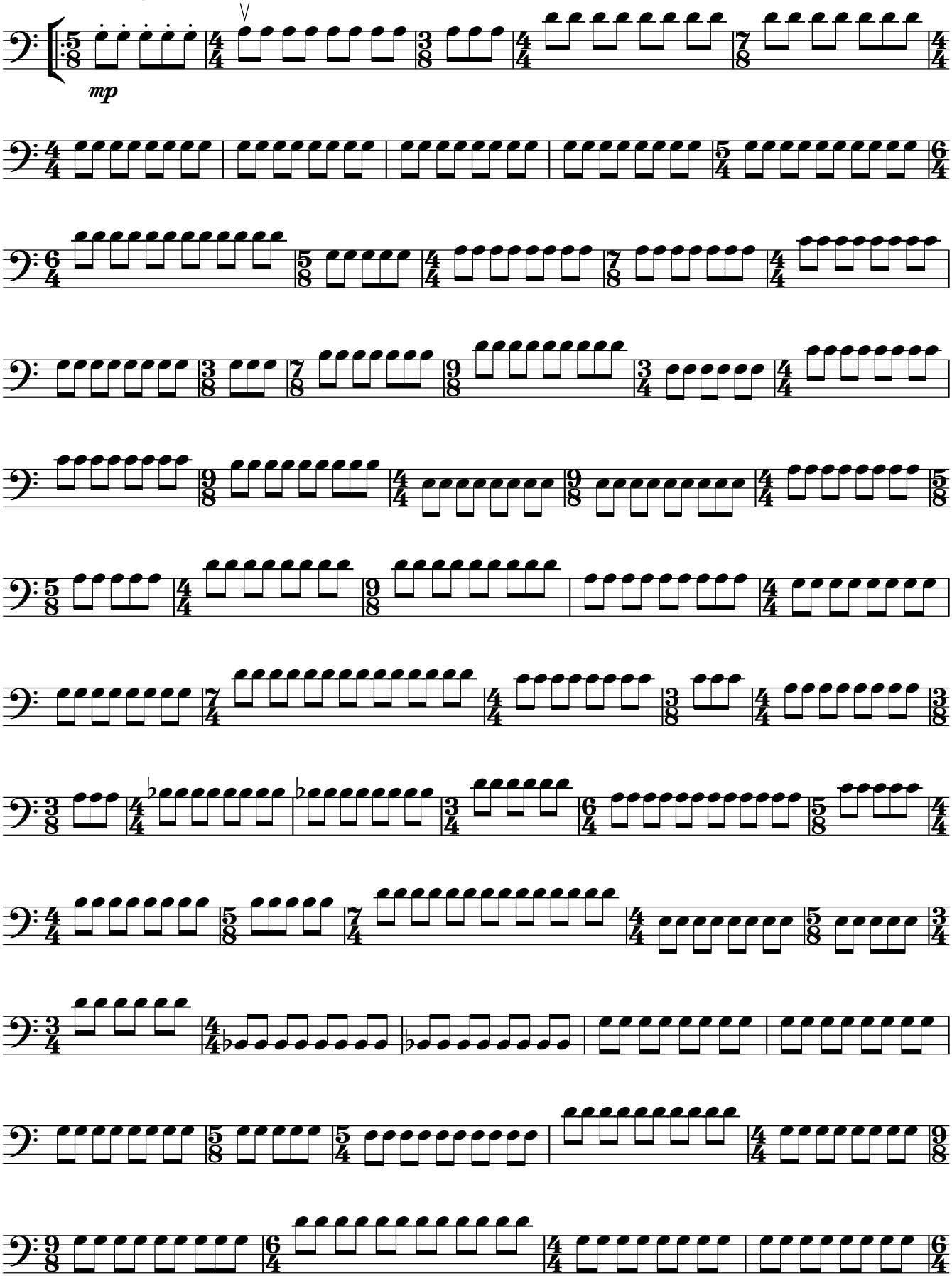
# The Black Rain

Cello

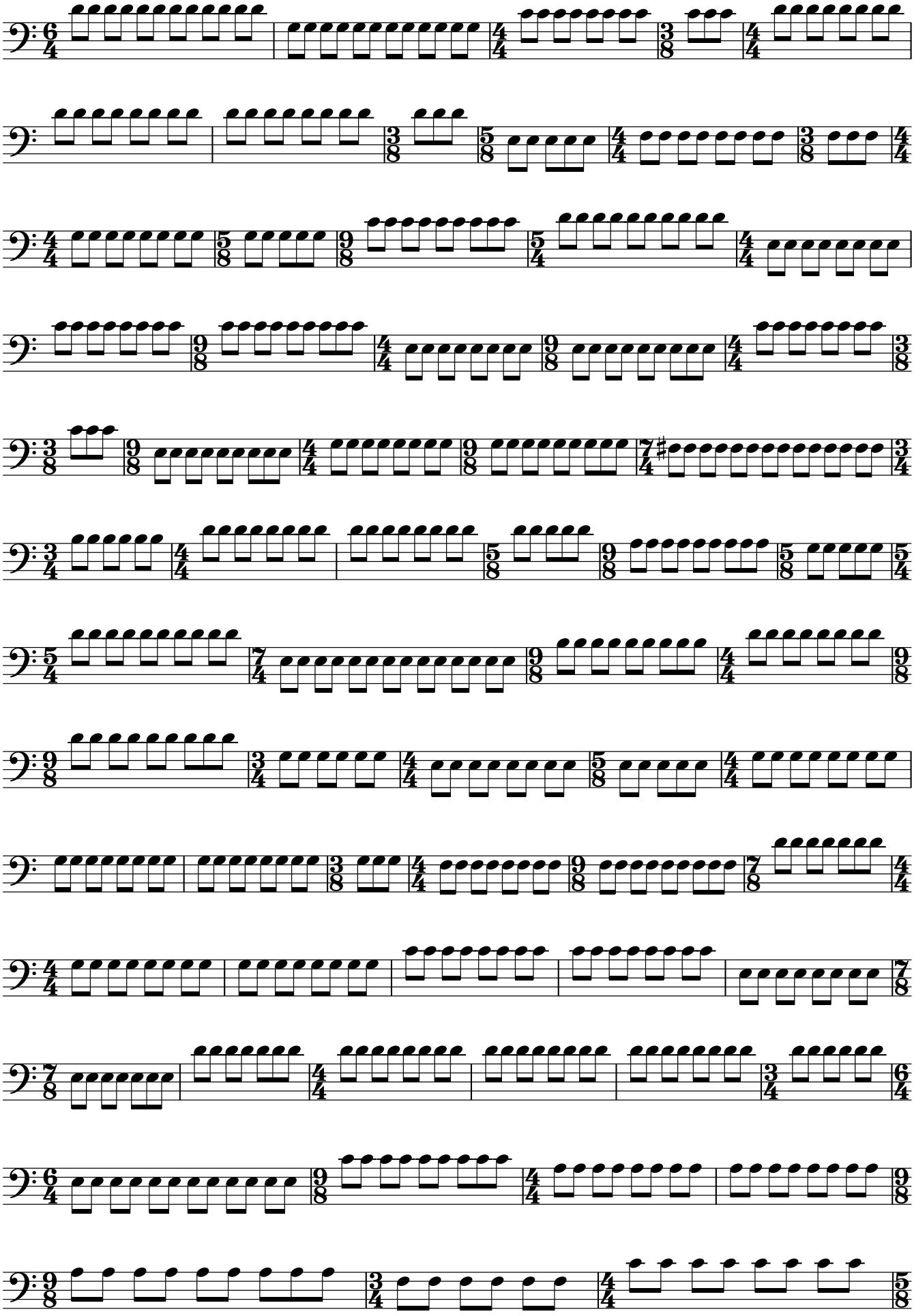
J. Simon van der Walt

♩ = 140

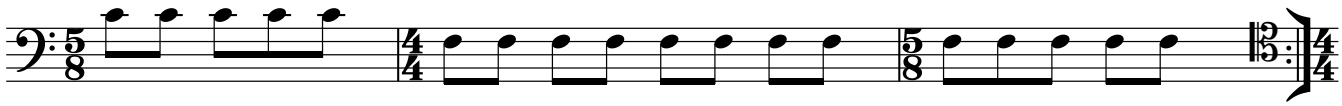
Start on any D: sempre stacc: alternate bowing always,
do not align downbows with barlines



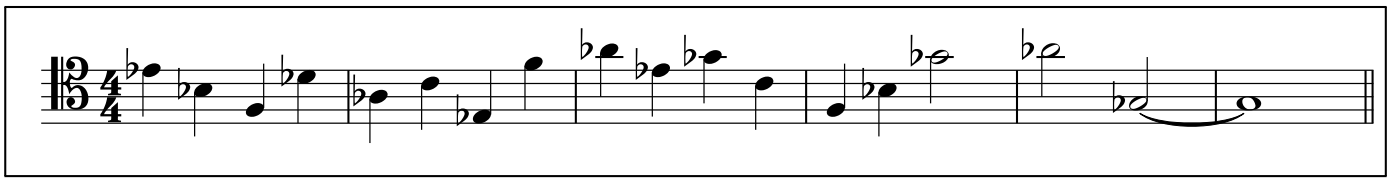
Cello

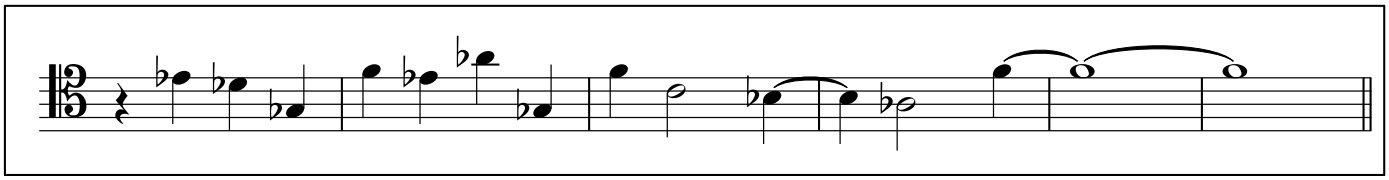


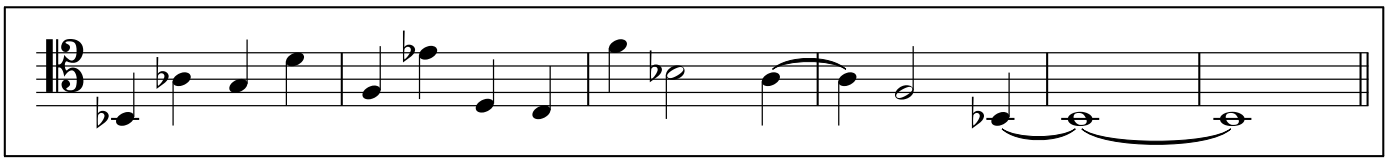
Cello

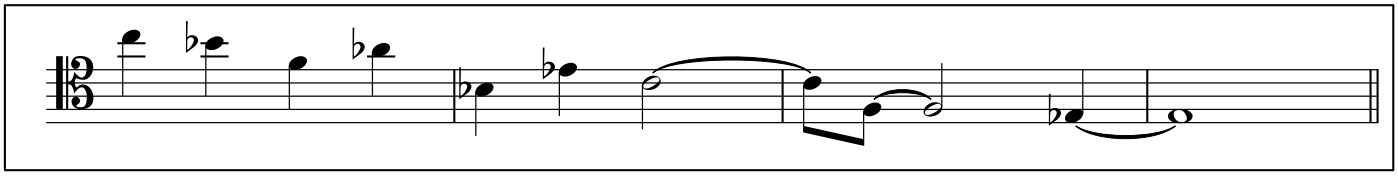


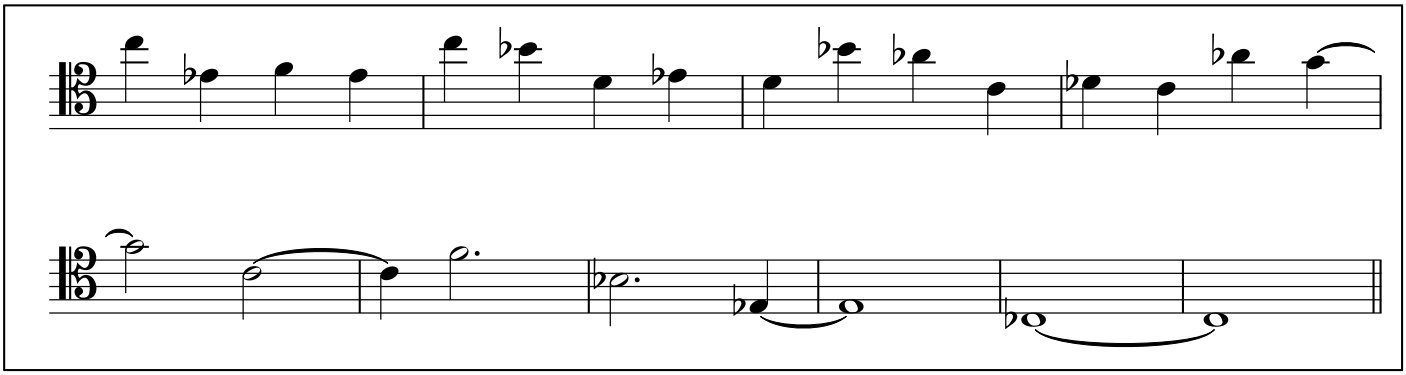
legato, in relievo

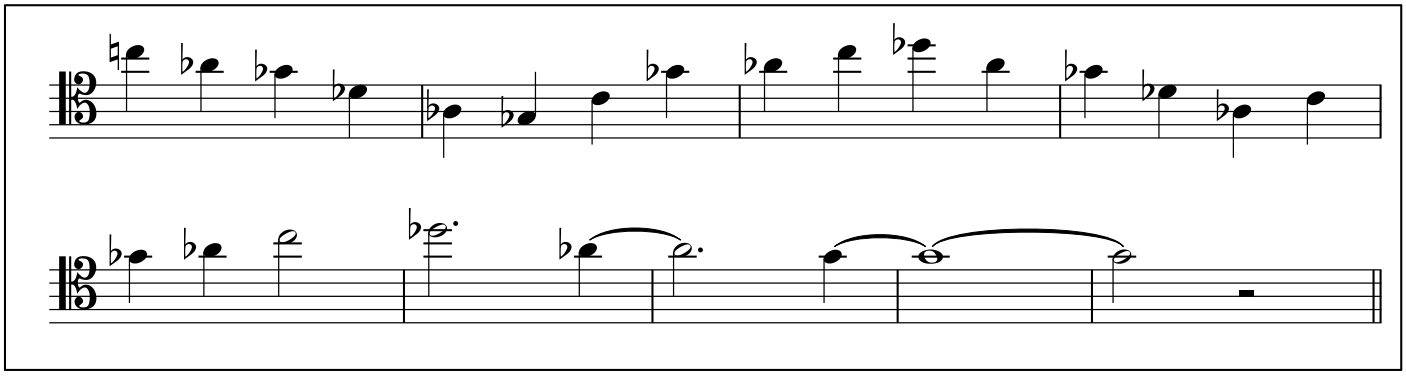












# The Black Rain

Viola

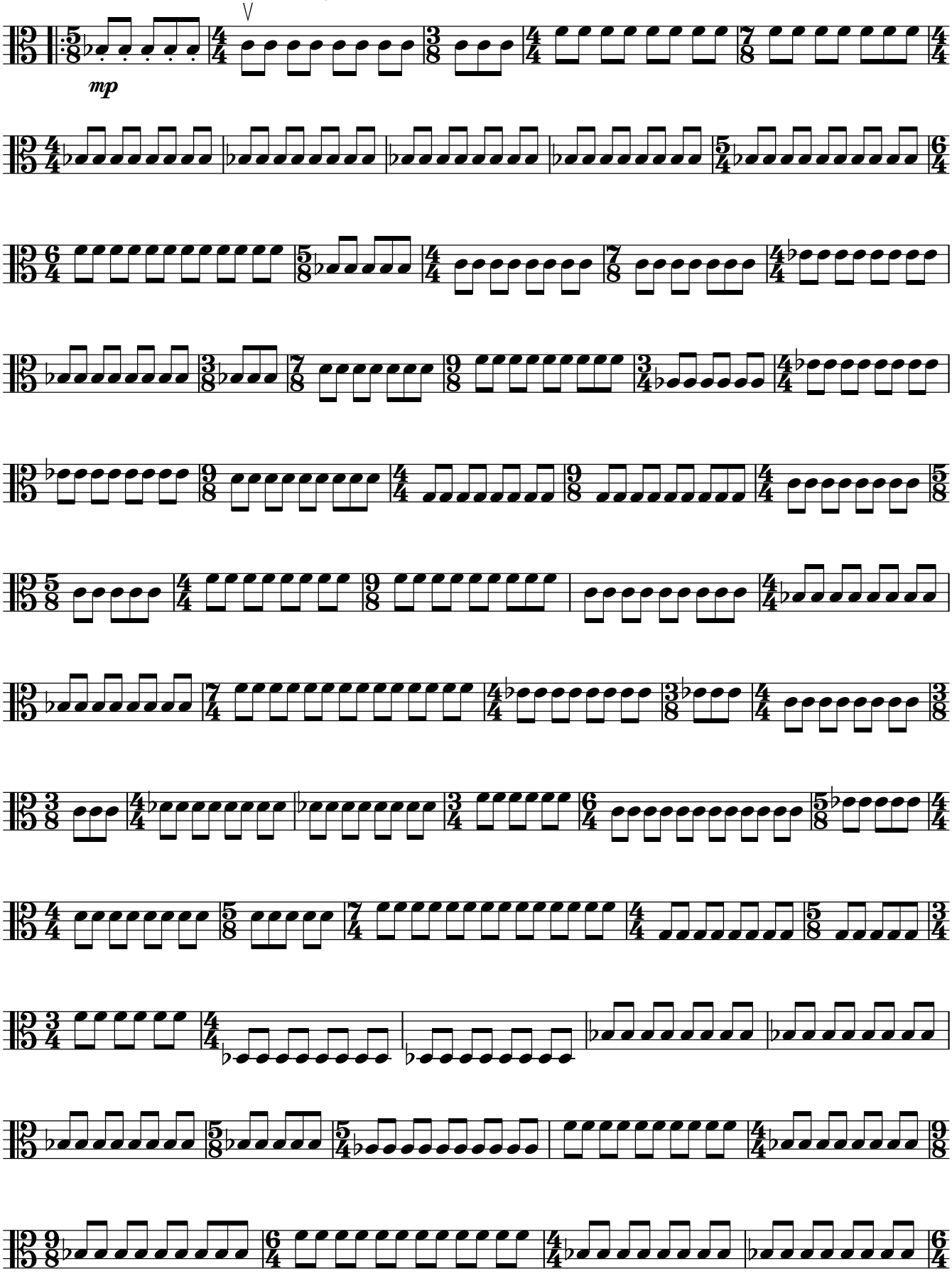
Start on any F

J. Simon van der Walt

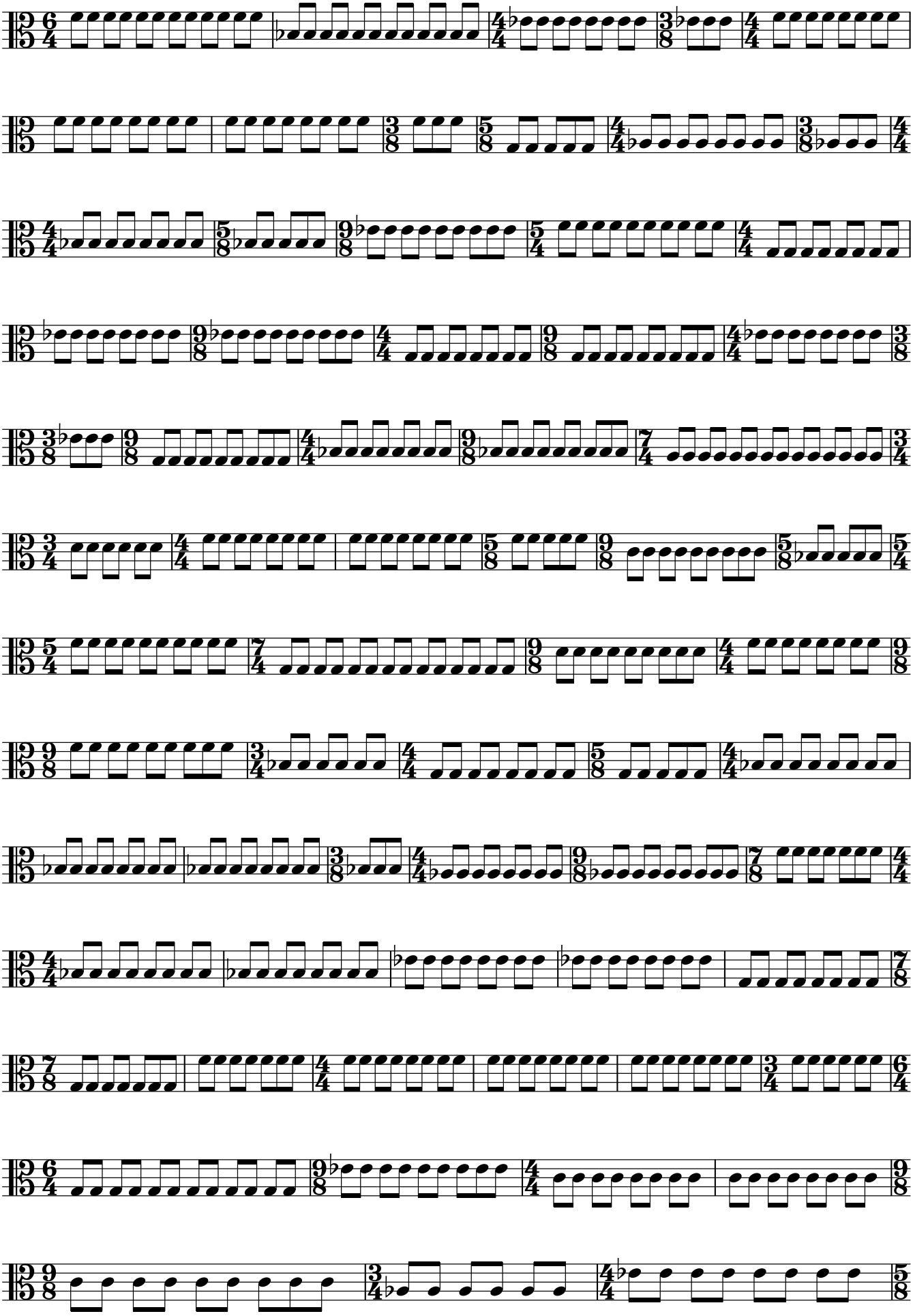
♩ = 140

sempre stacc: alternate bowing always,
do not align downbows with barlines

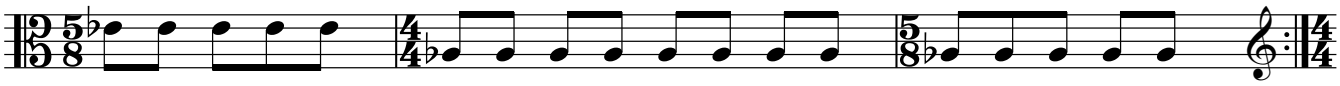
*mp*



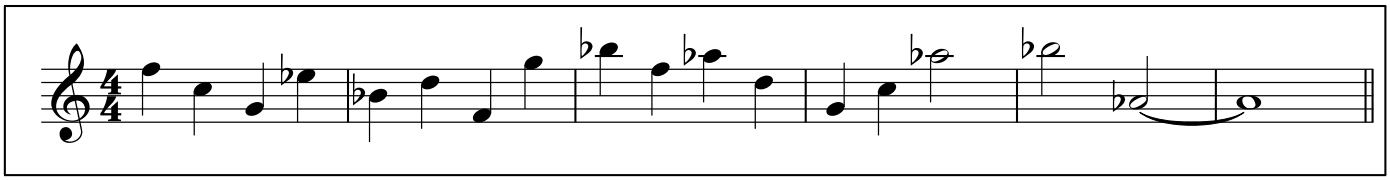
Viola

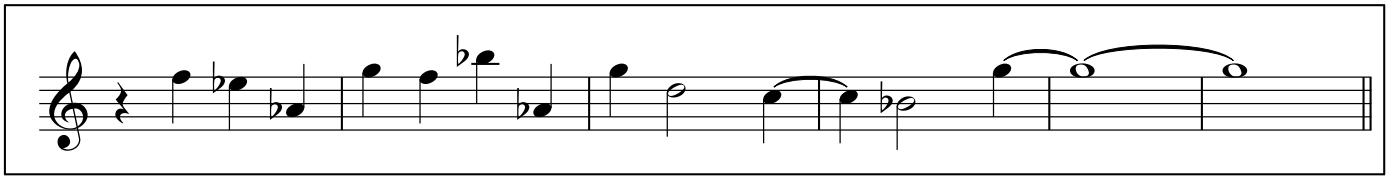


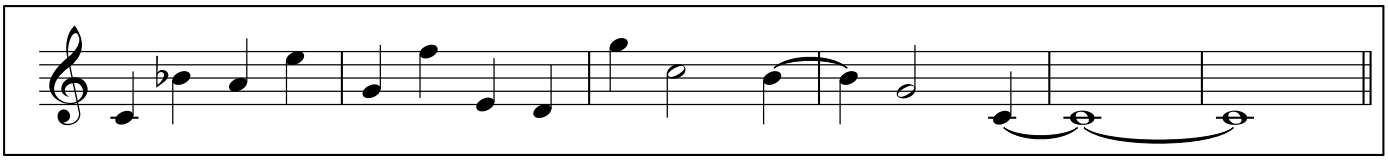
Viola

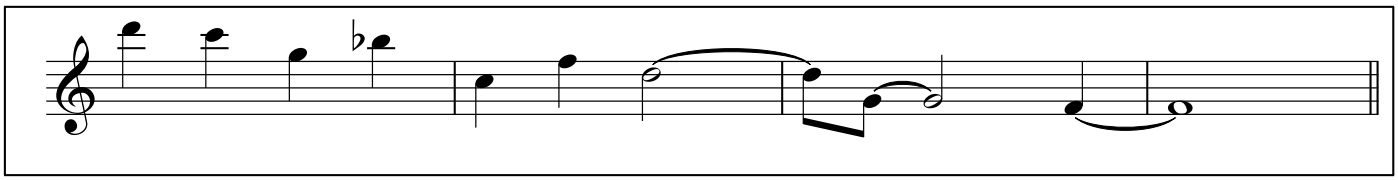


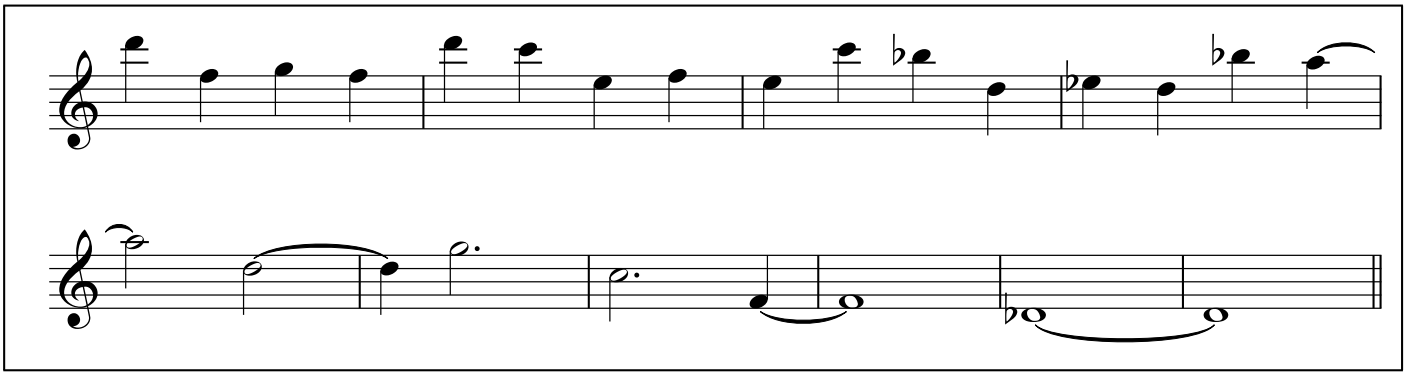
legato, in relievo

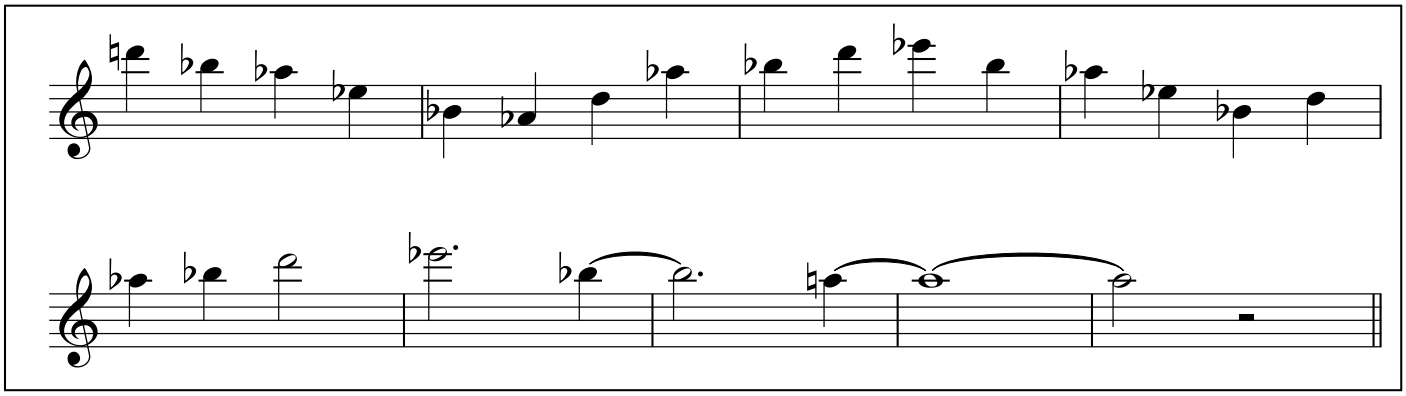












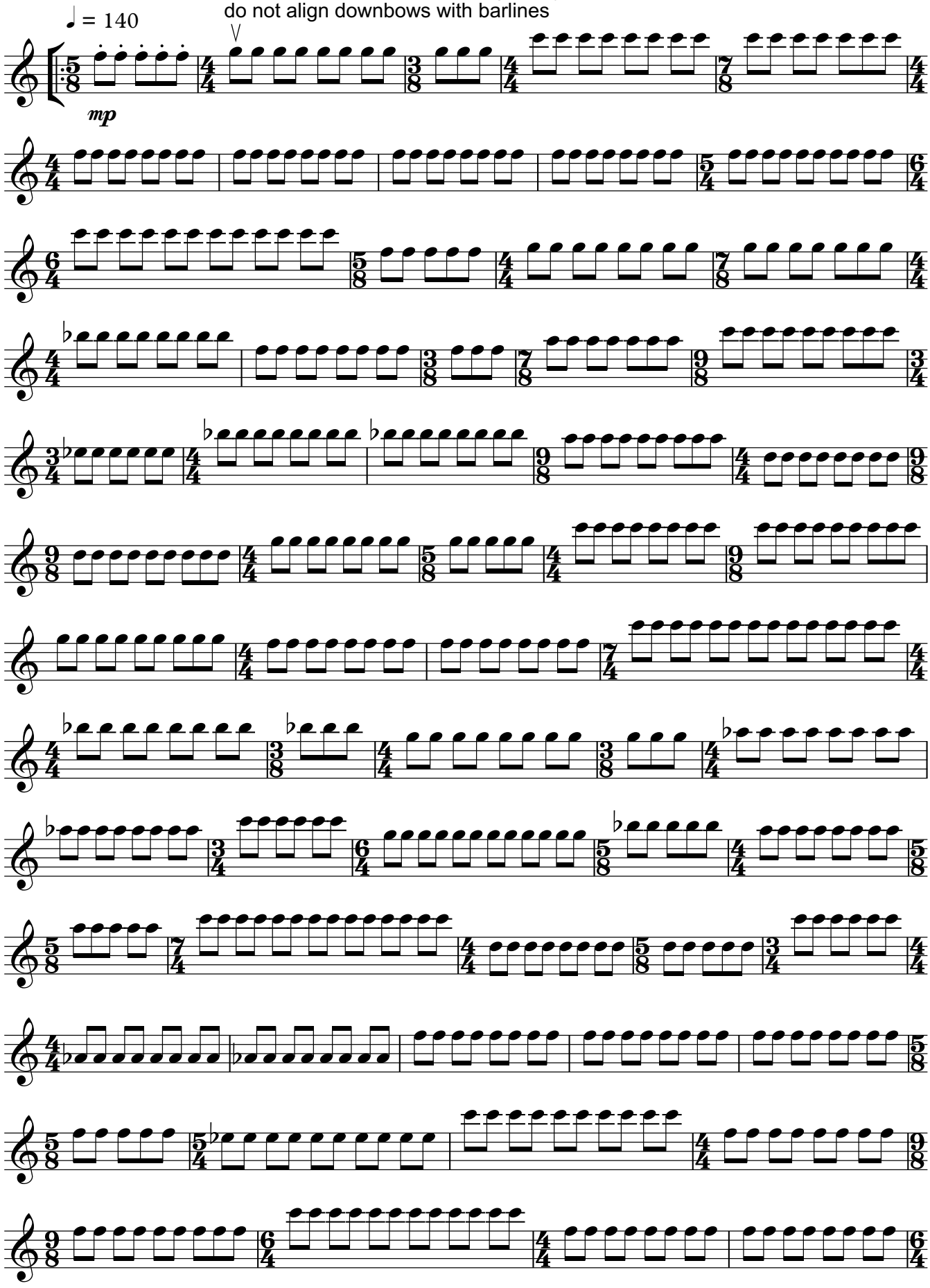
# The Black Rain

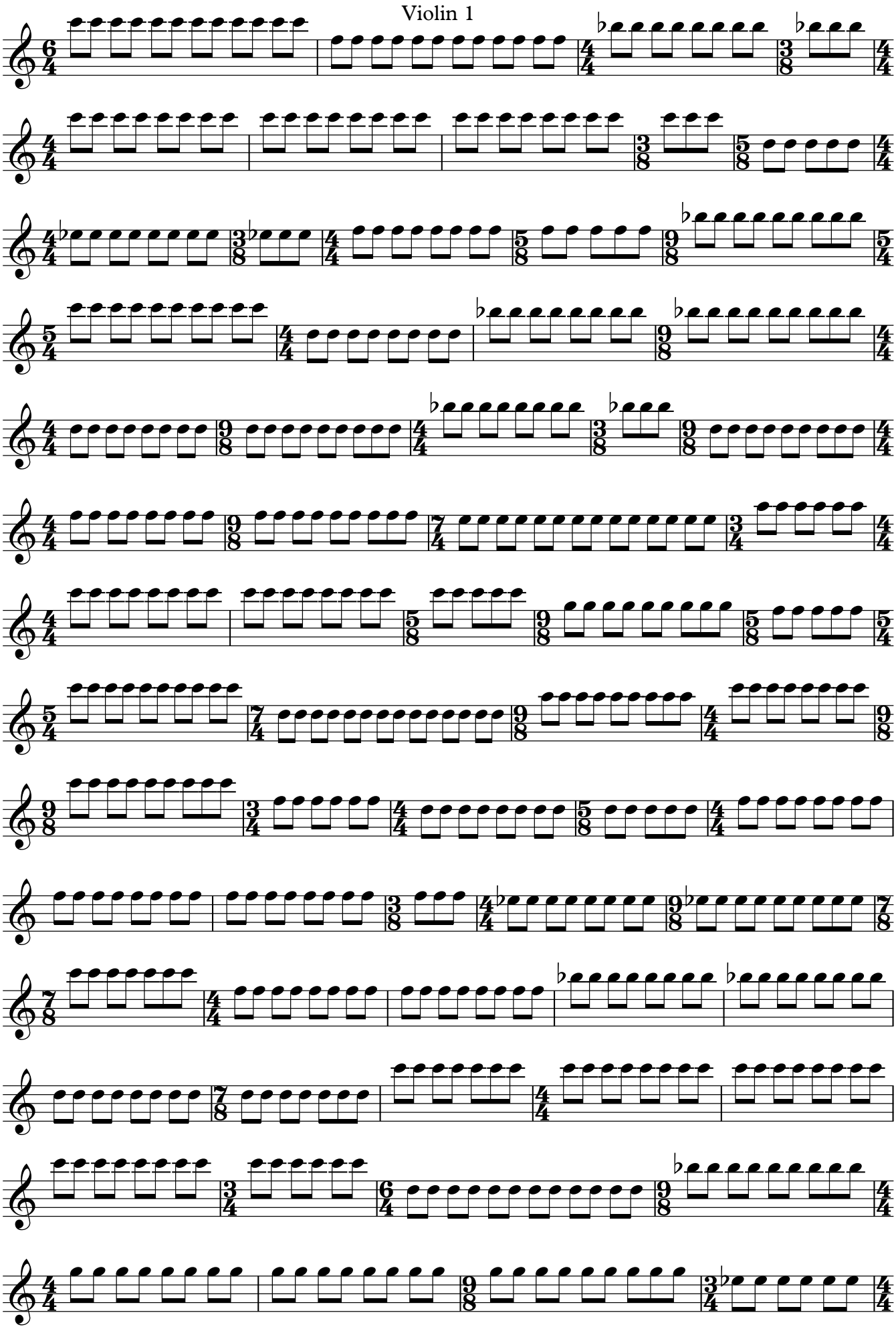
J. Simon van der Walt

Violin 1

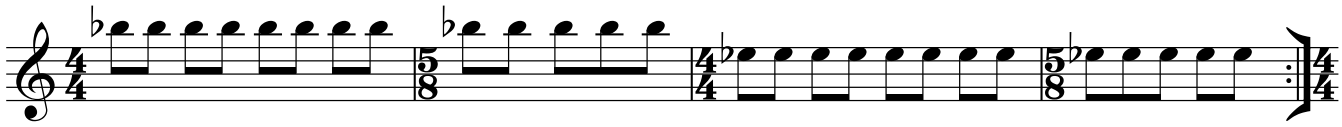
Start on any C

sempre stacc: alternate bowing always,
do not align downbows with barlines

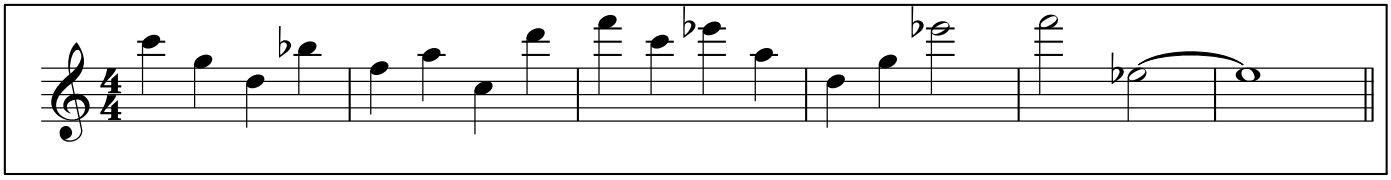


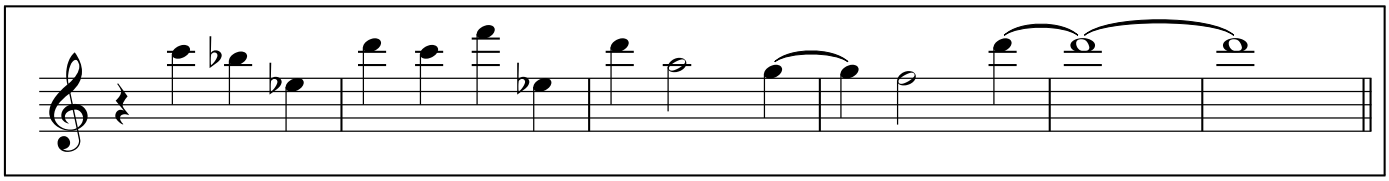

Violin 1

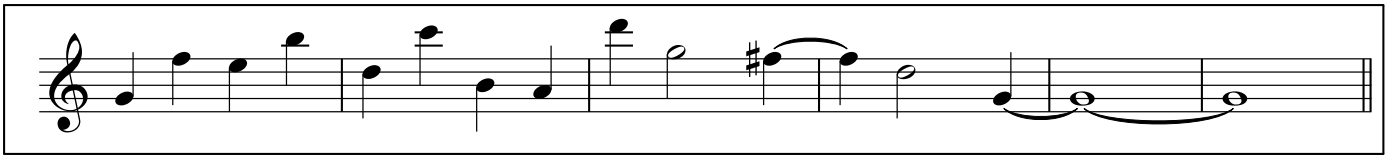
Violin 1

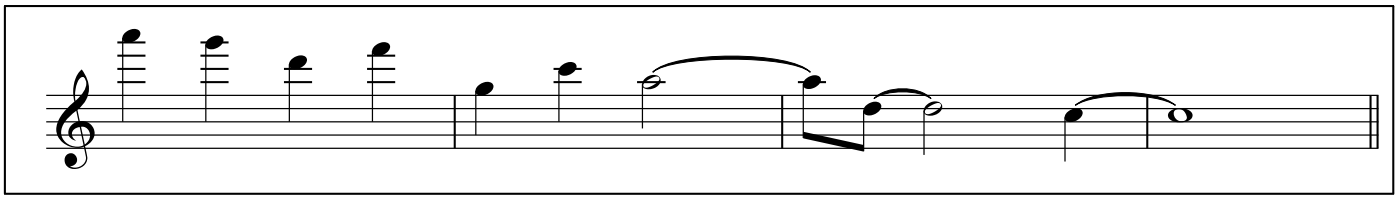


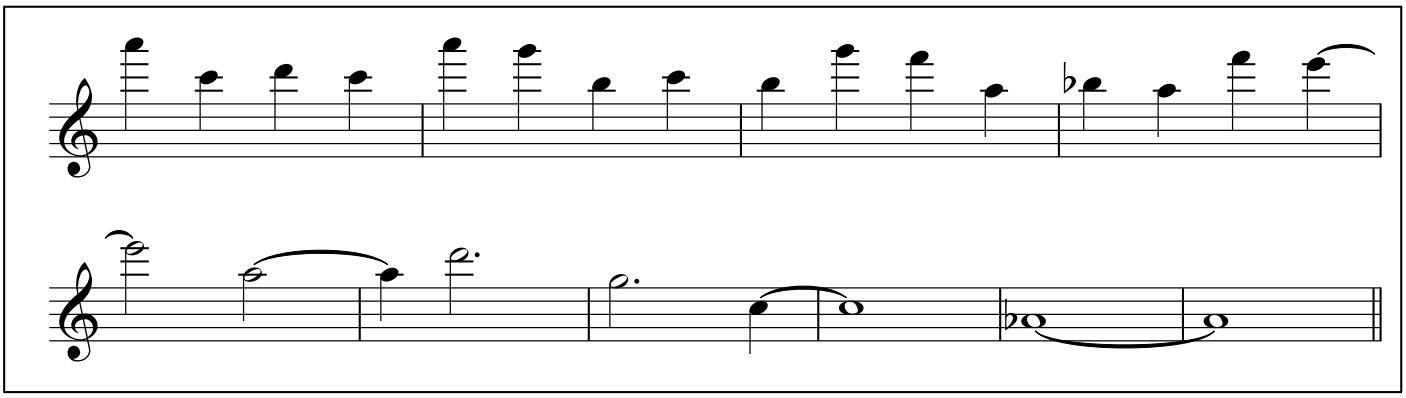
legato, in relievo

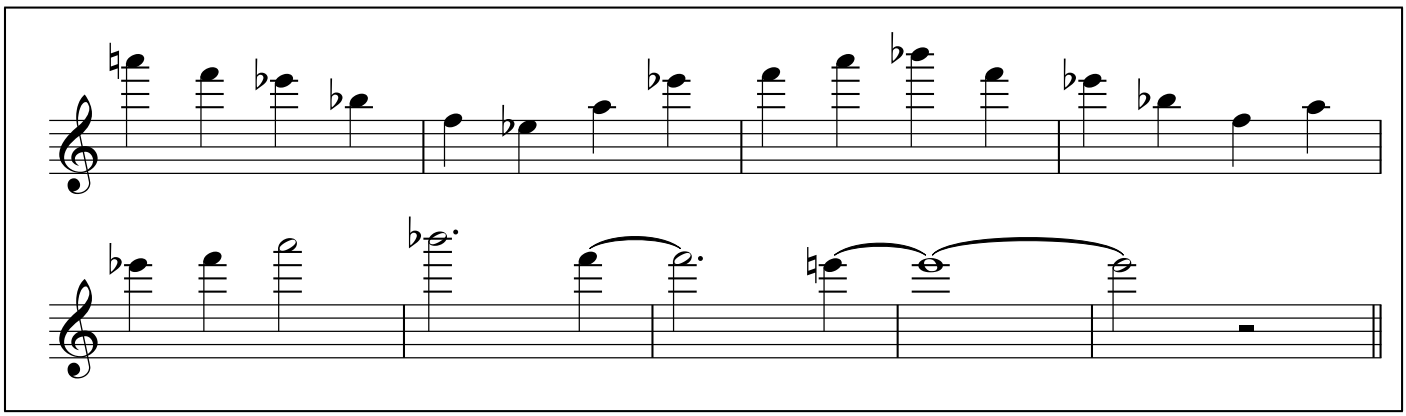












# The Black Rain

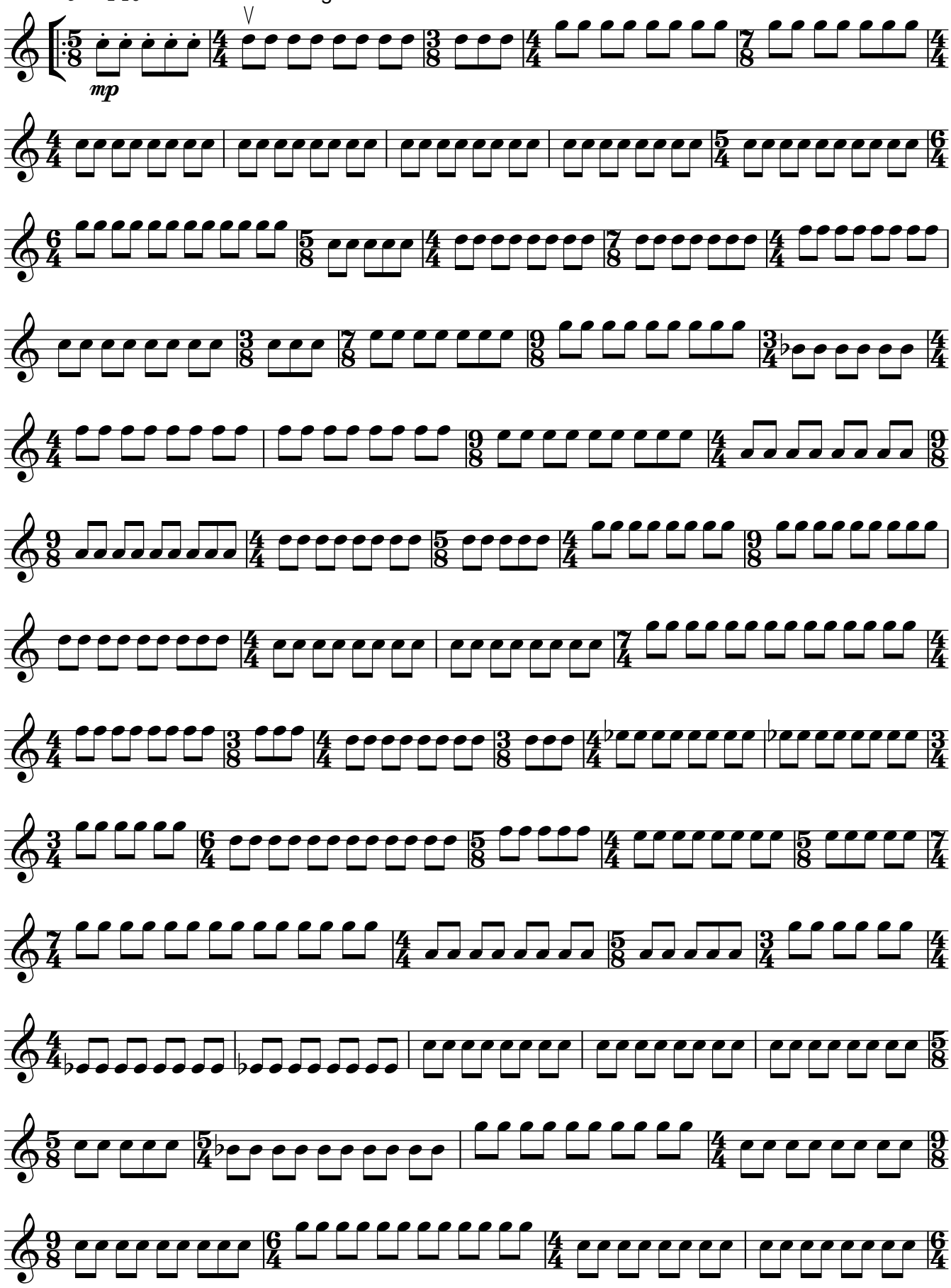
Violin 2

Start on any G

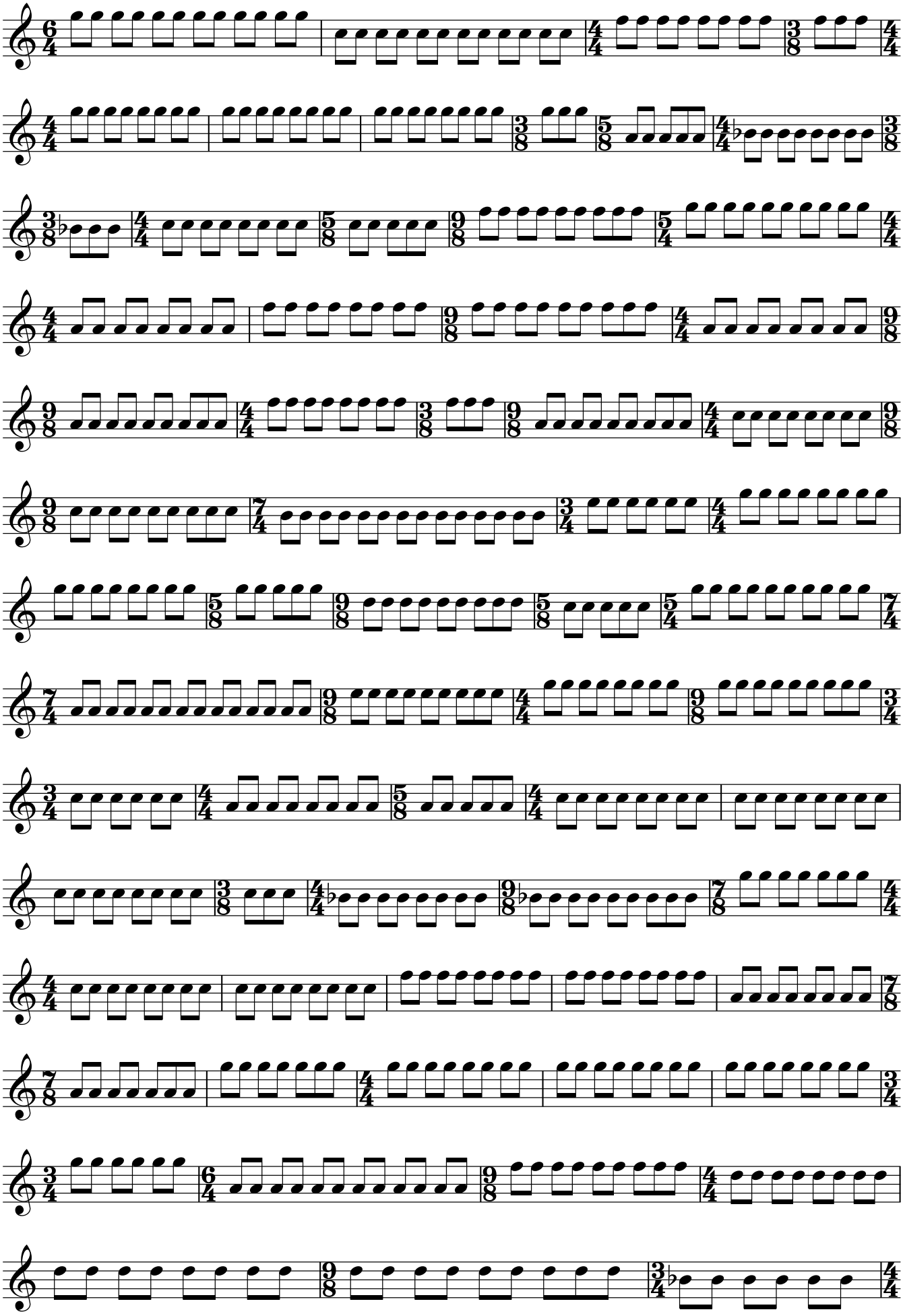
J. Simon van der Walt

♩ = 140

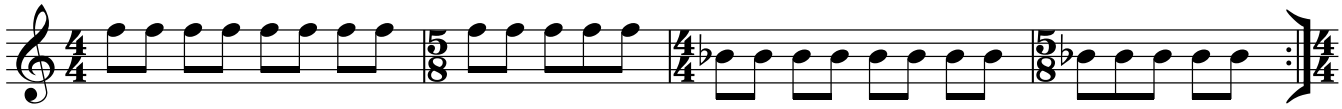
sempre stacc: alternate bowing always,
do not align downbows with barlines



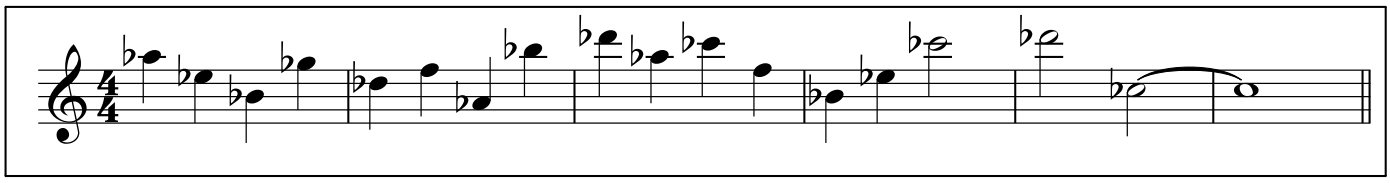
Violin 2

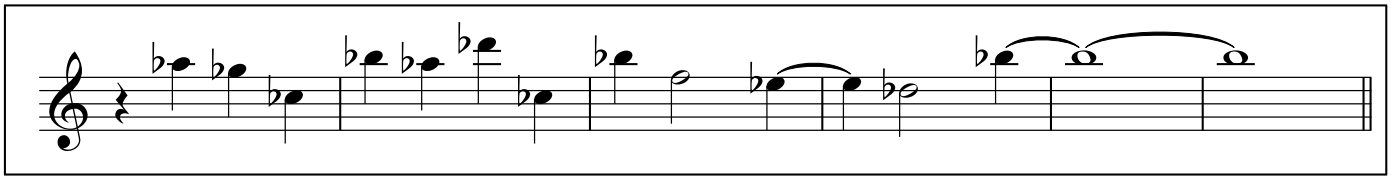


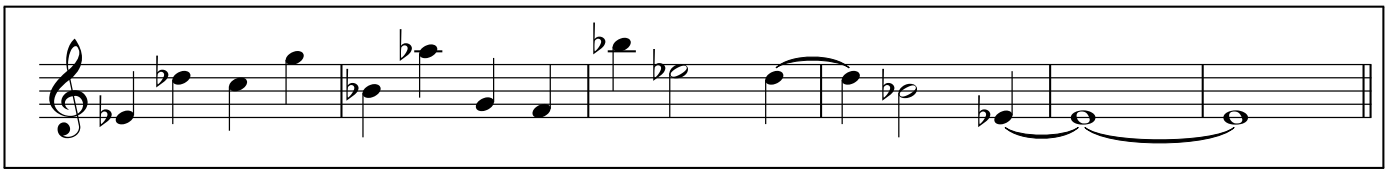
Violin 2

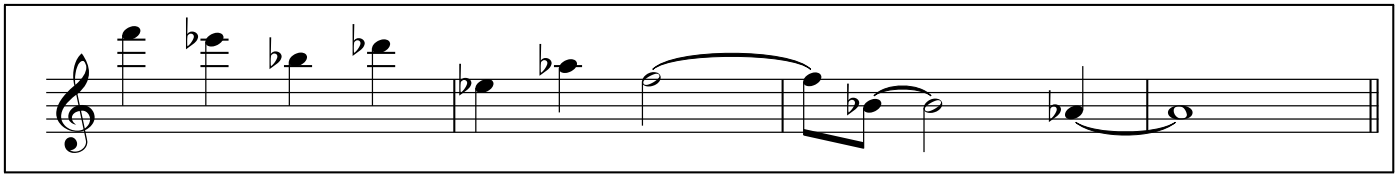


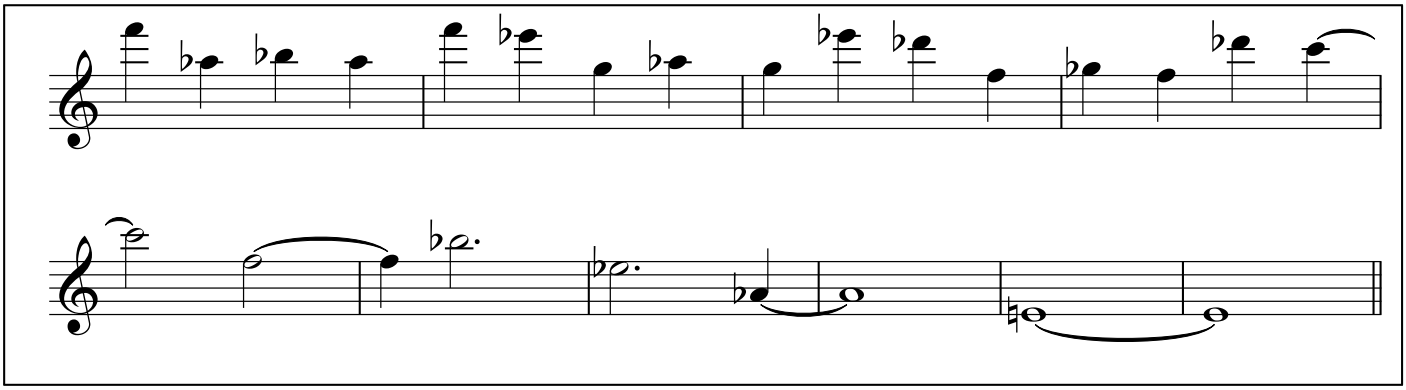
legato, in relievo

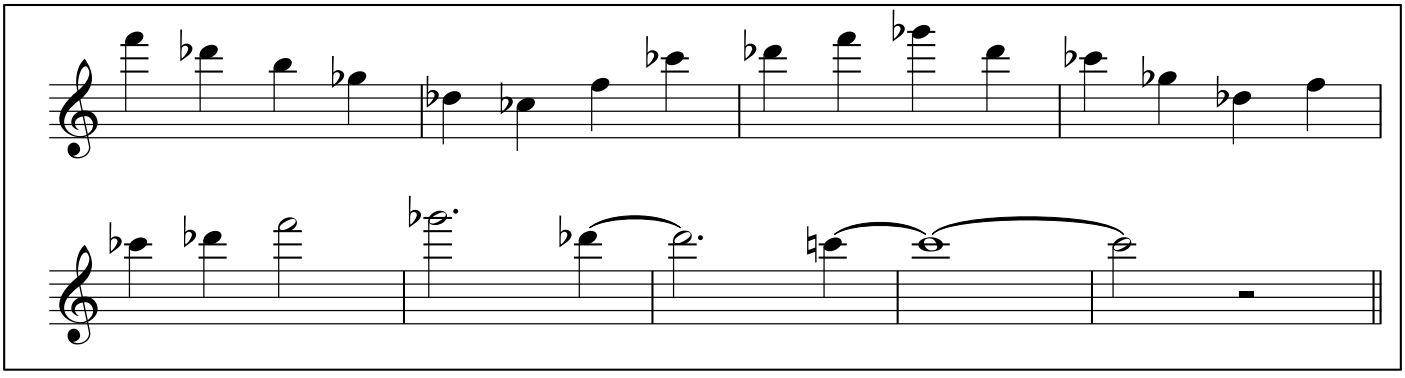










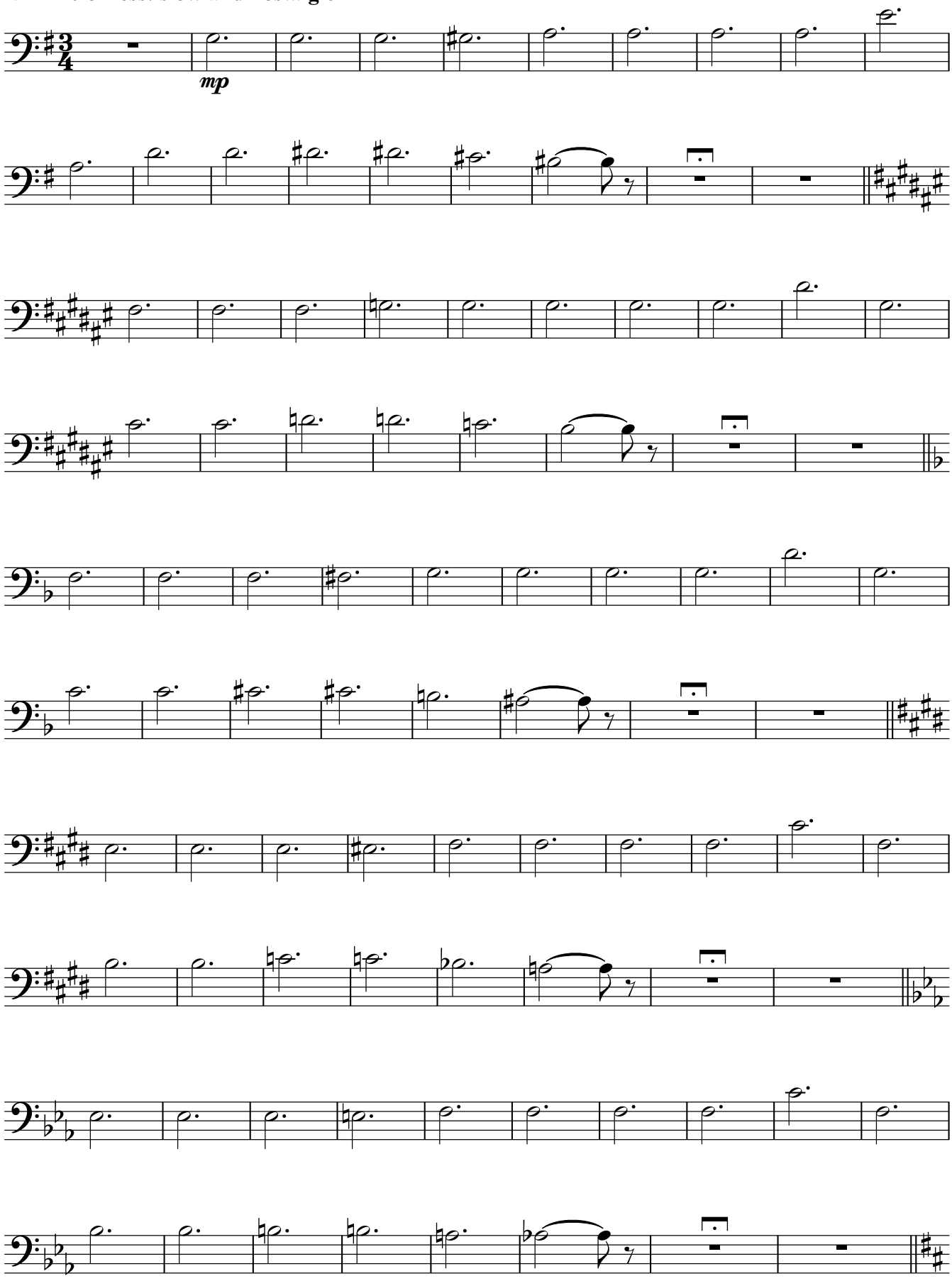


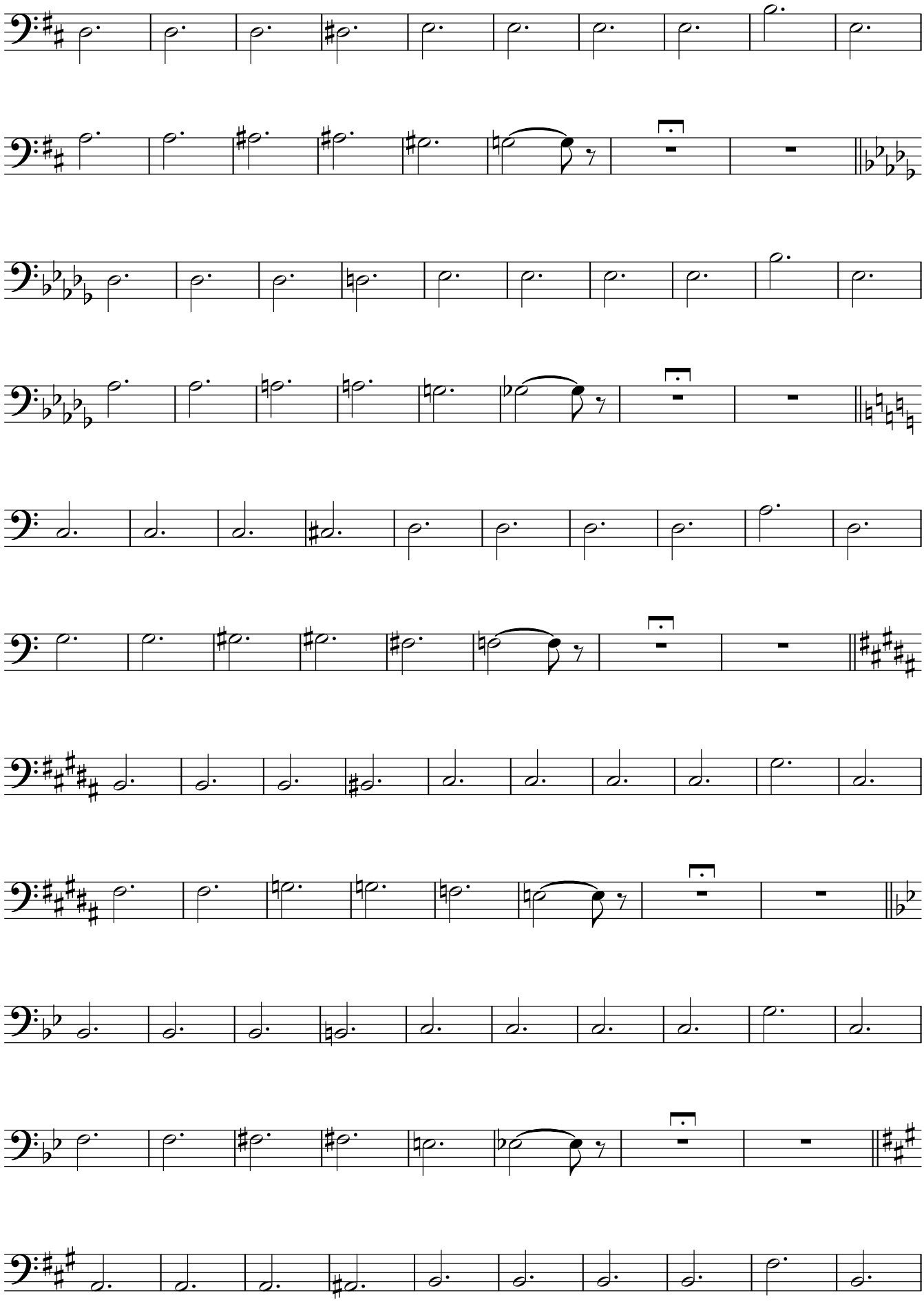
Bass

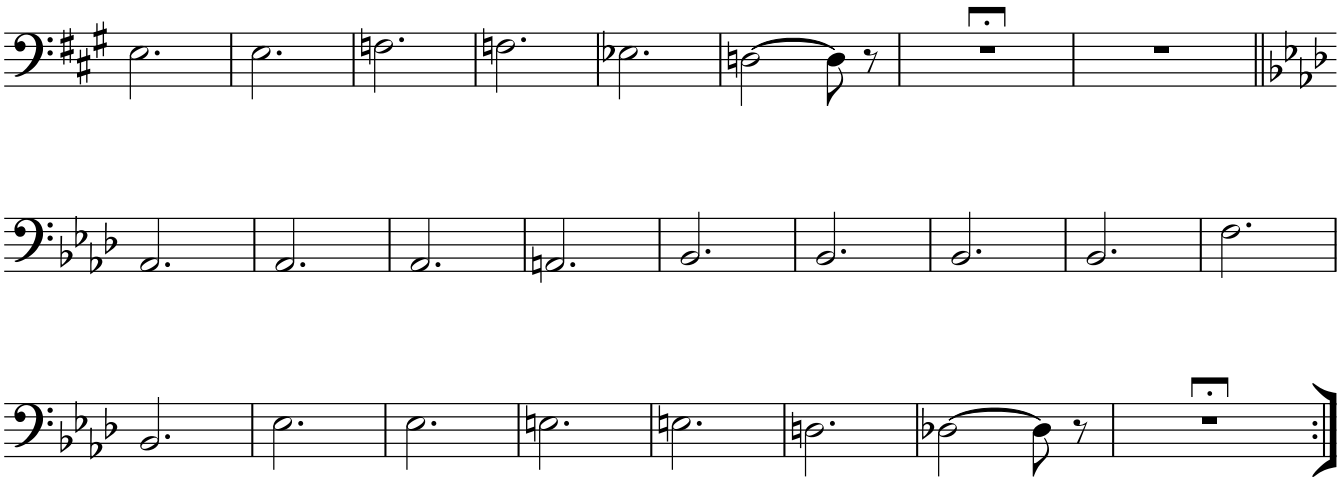
# SLIPPING AWAY

J. Simon van der Walt

♩ = 140 or less: slow and nostalgic





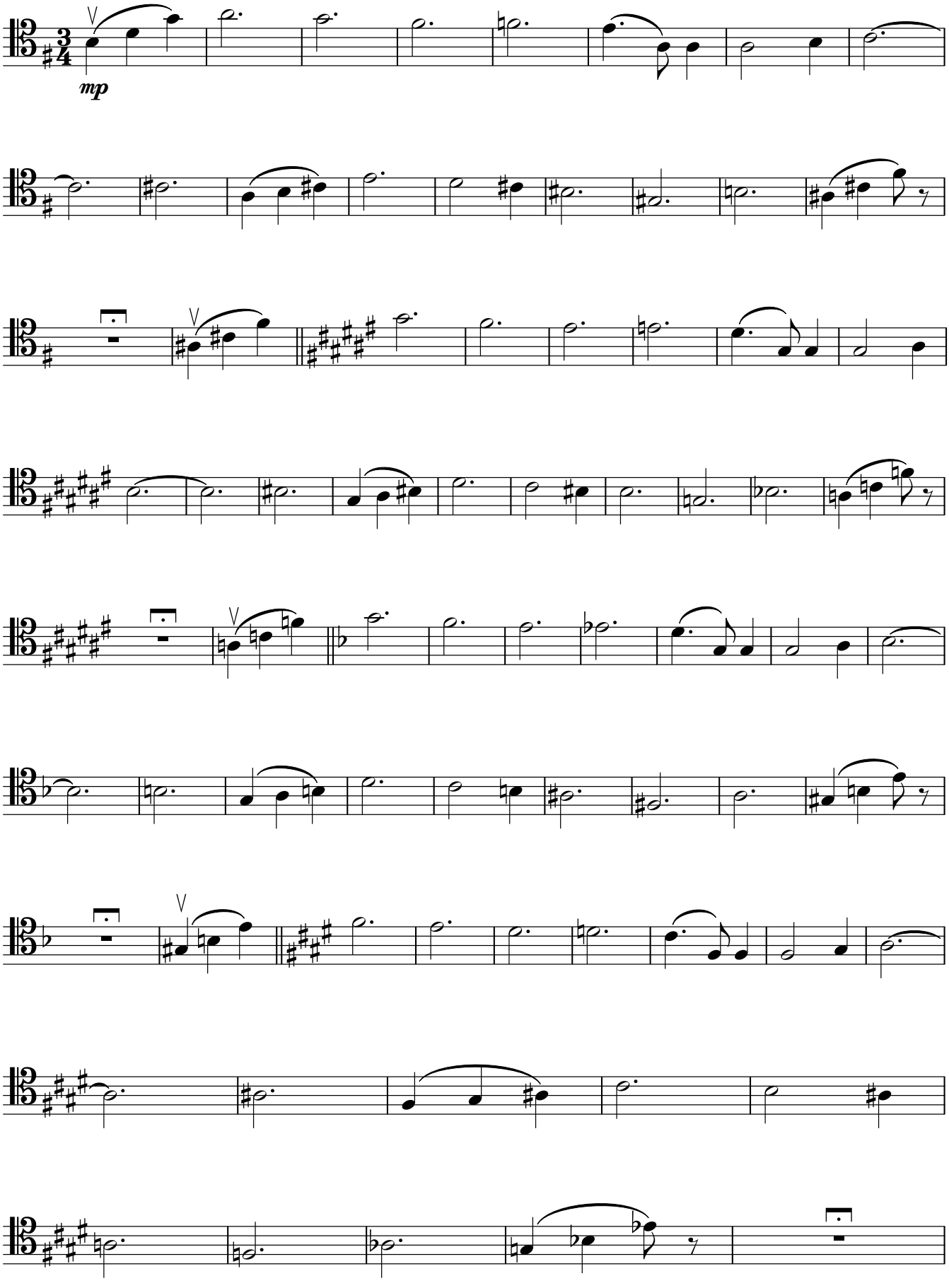


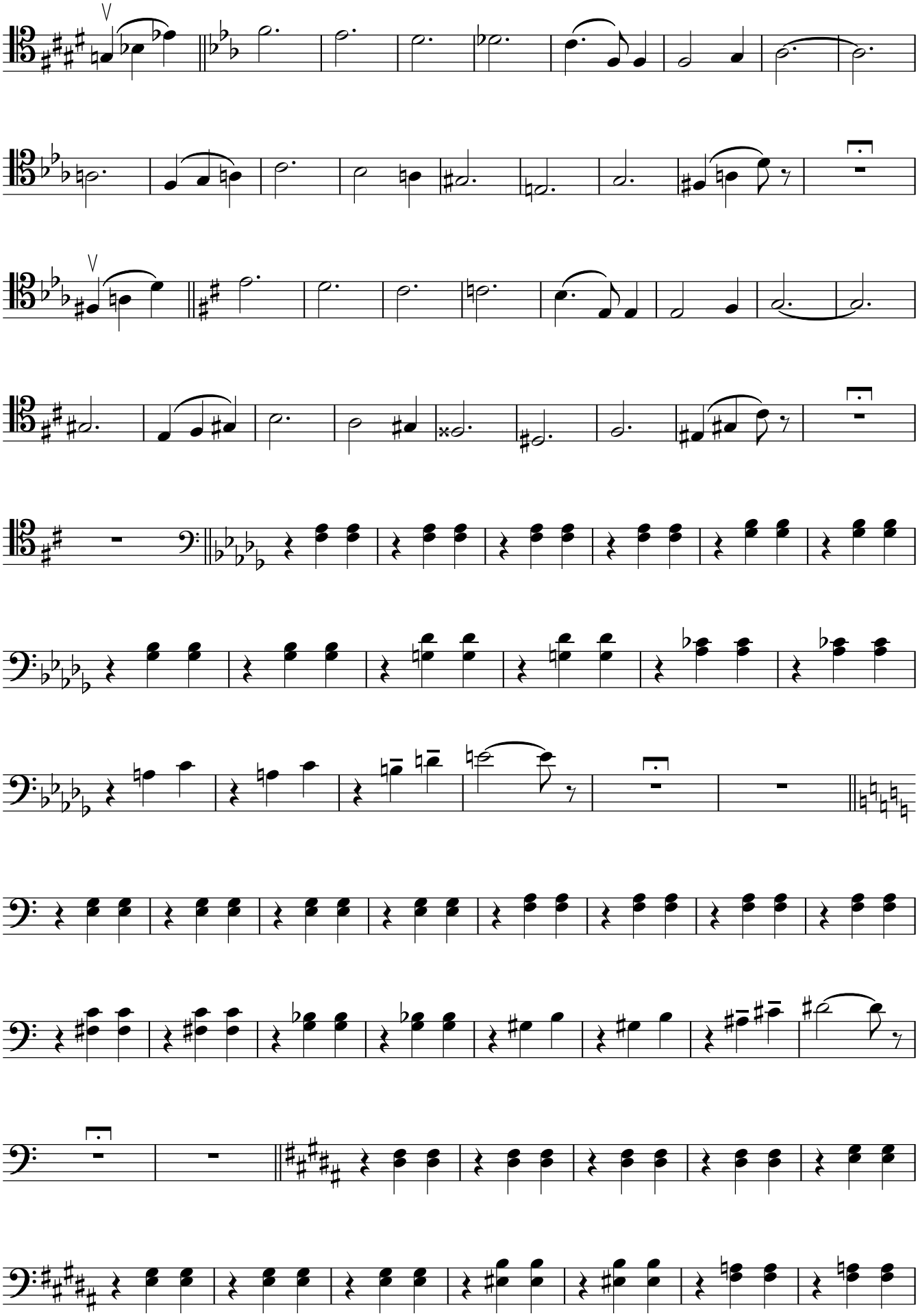
Cello

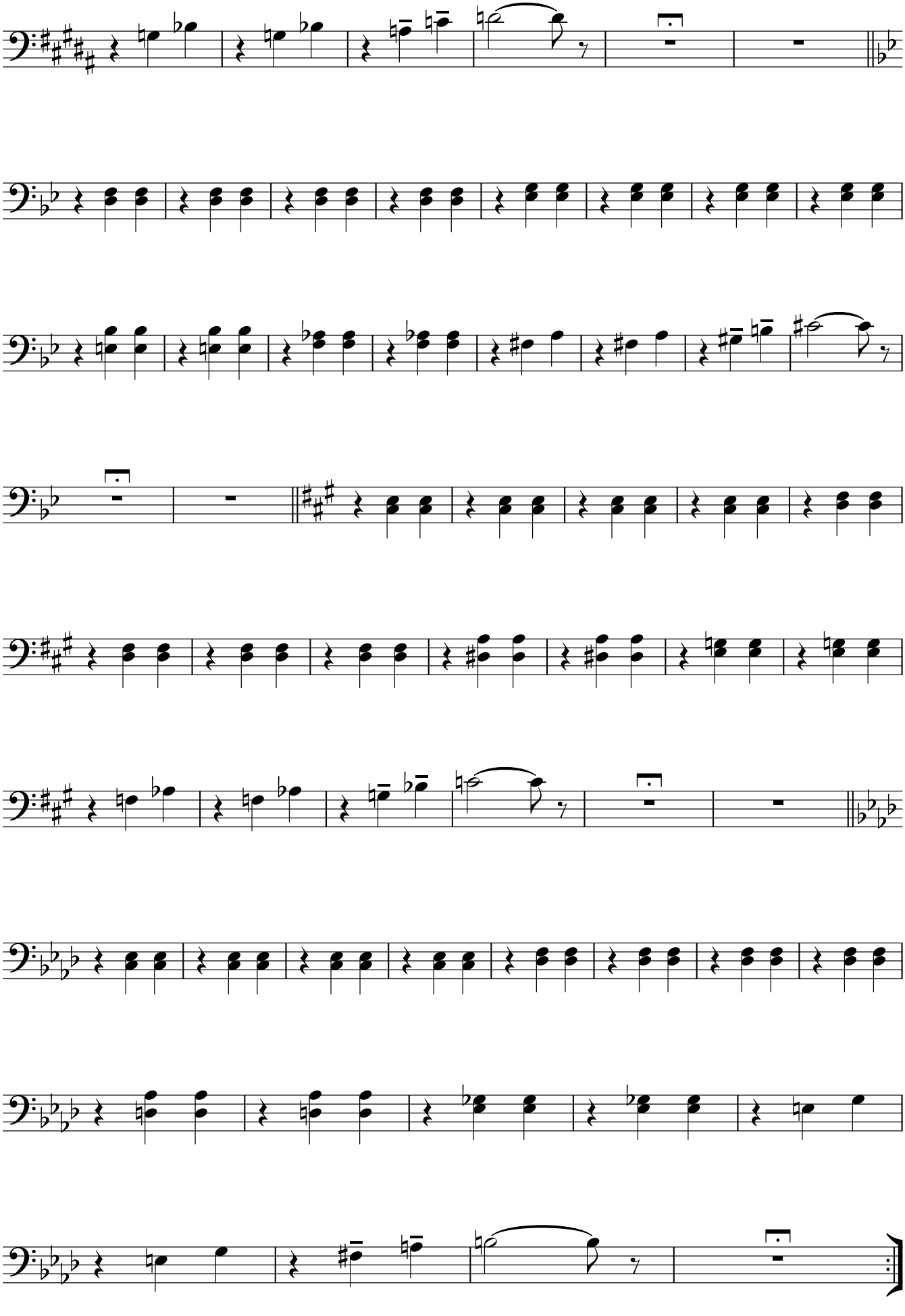
# SLIPPING AWAY

J. Simon van der Walt

**♩ = 140 or less: slow and nostalgic**





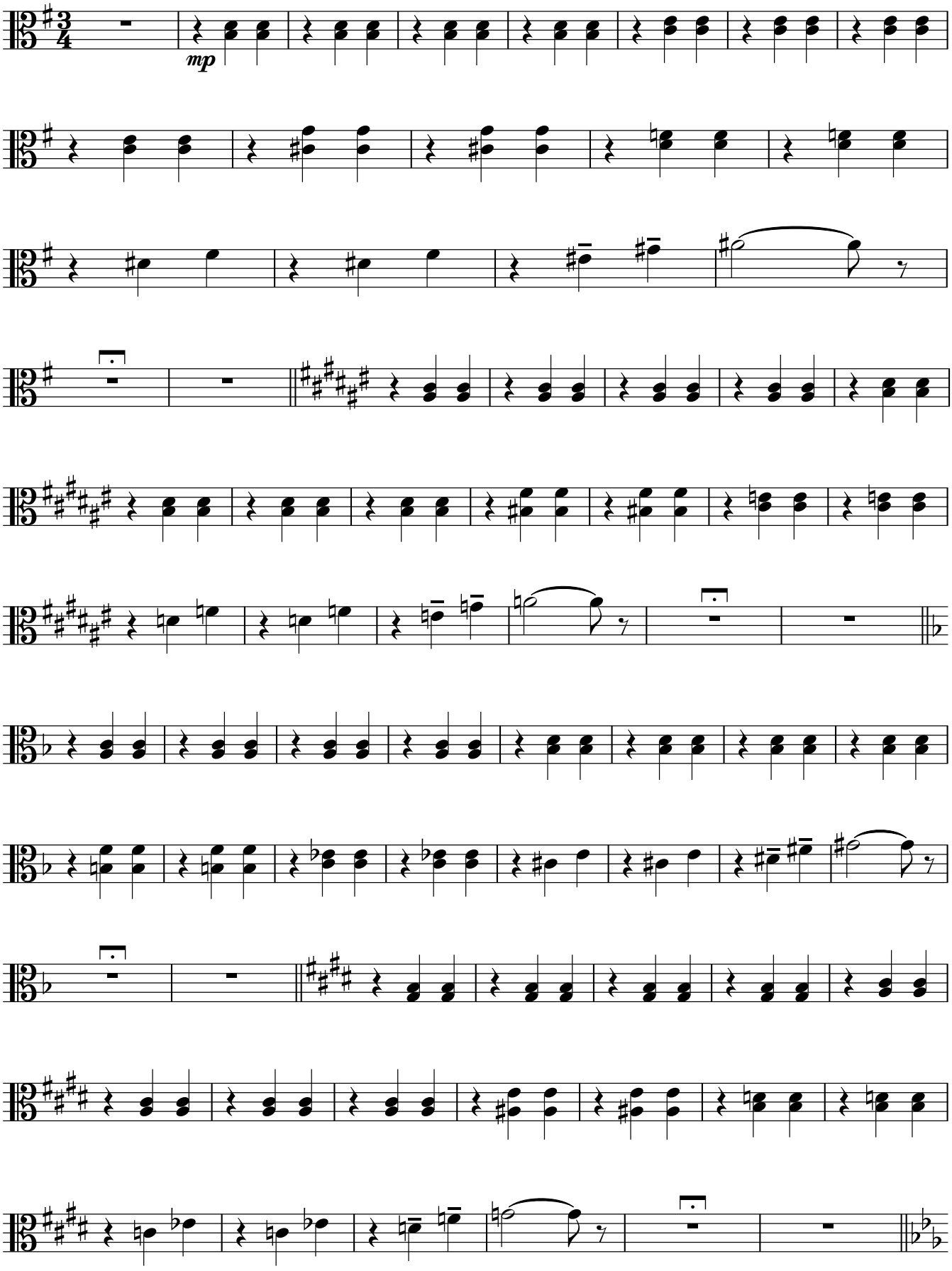


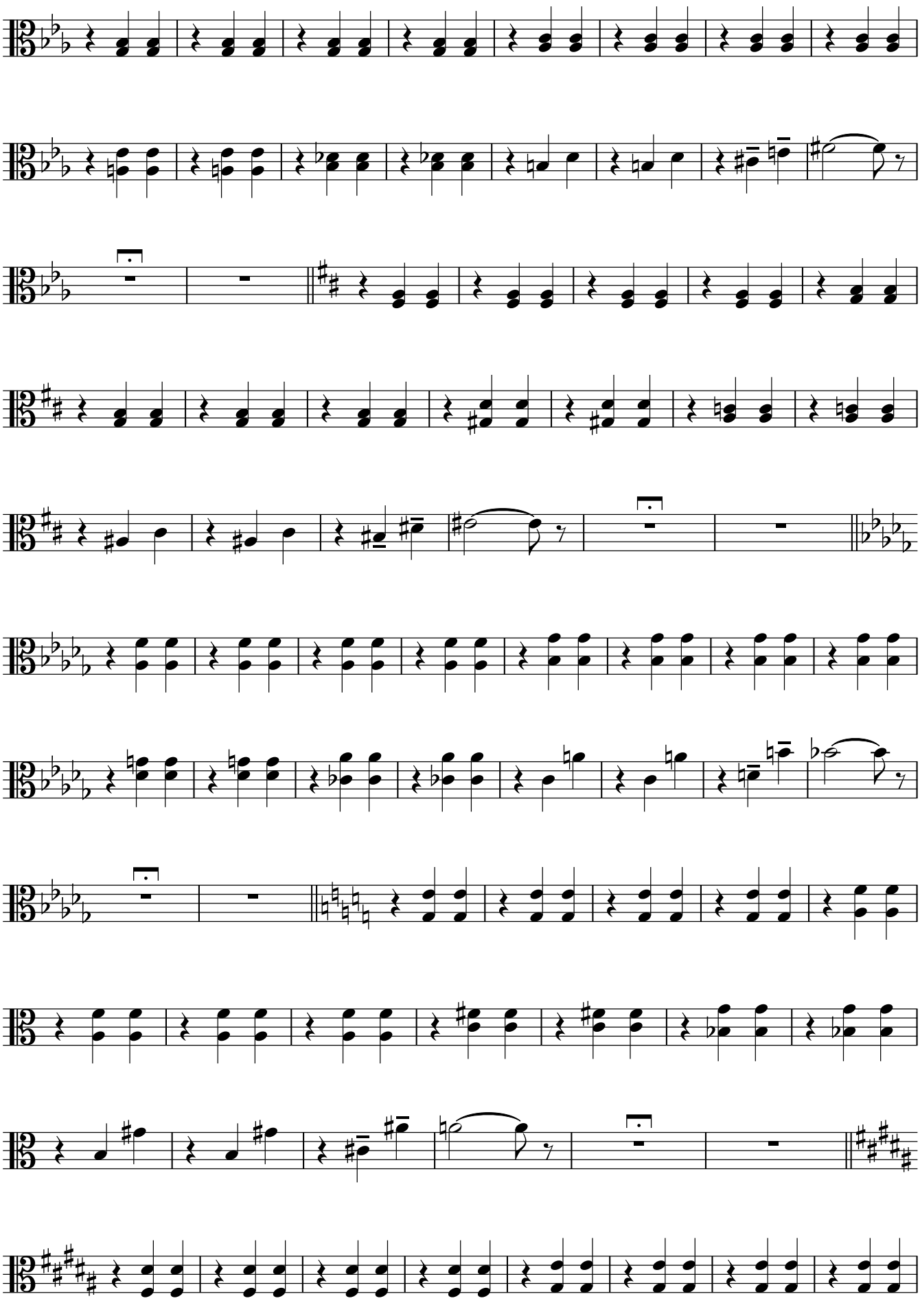
Viola

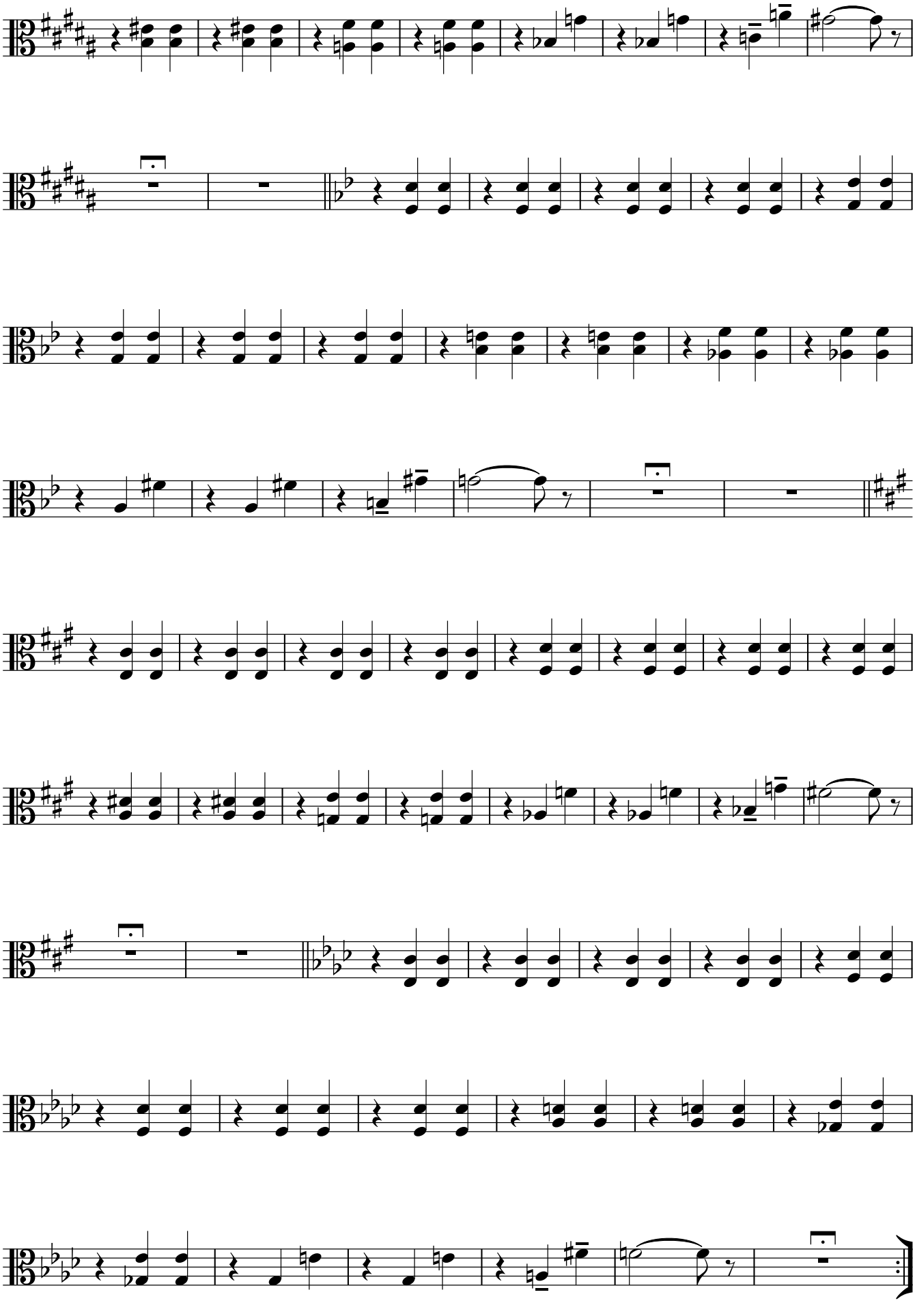
# SLIPPING AWAY

J. Simon van der Walt

♩ = 140 or less: slow and nostalgic





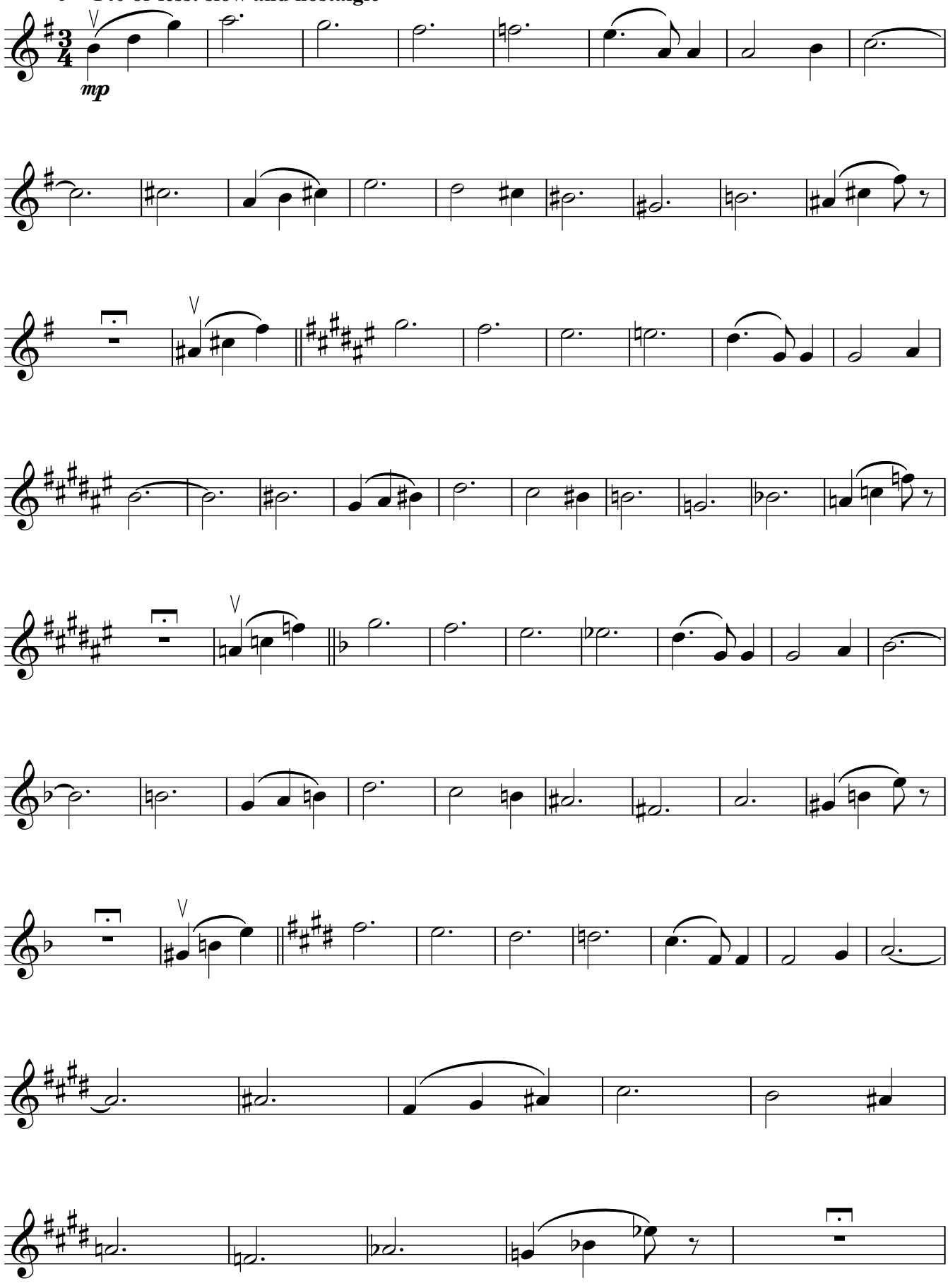


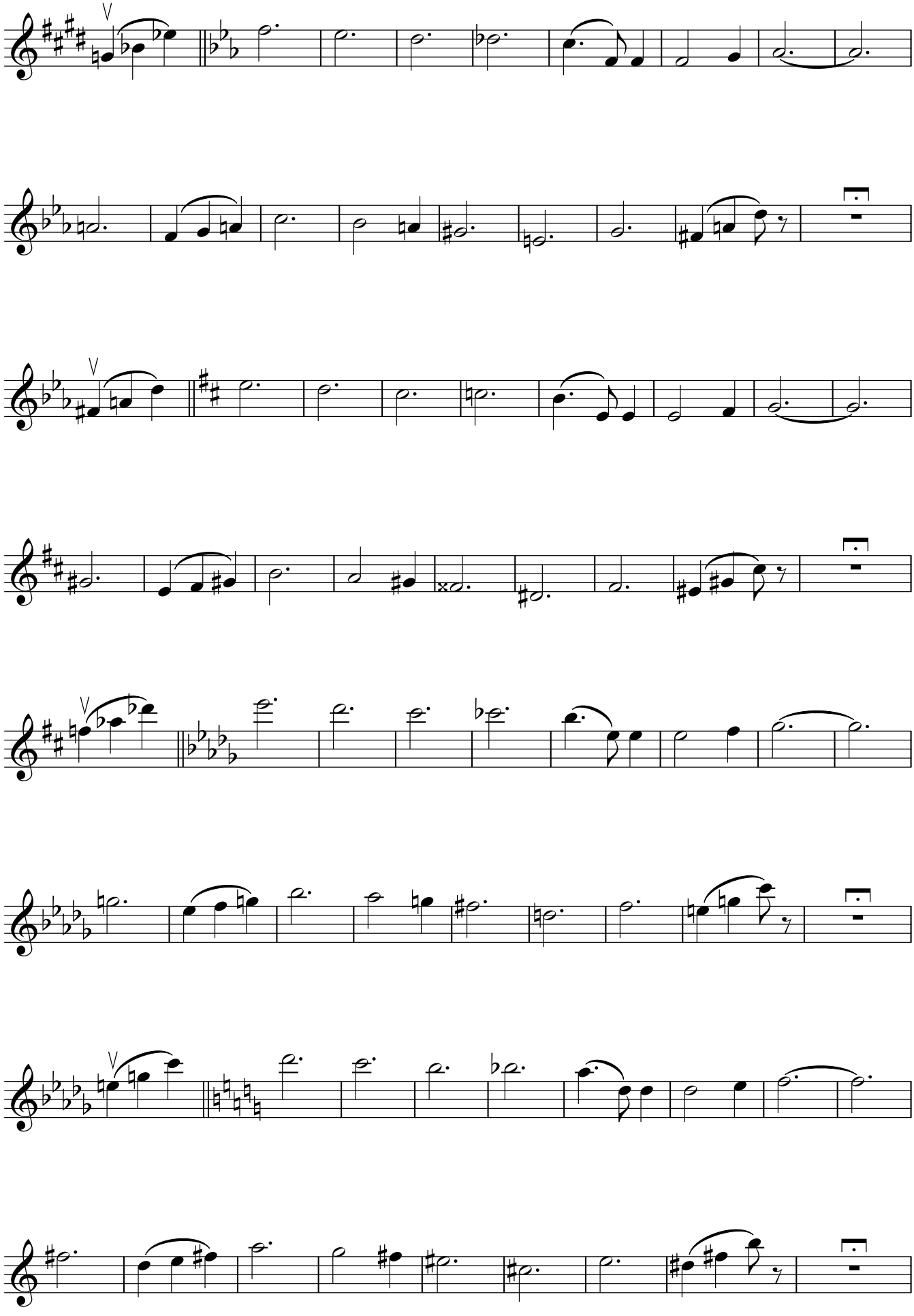
Violin 1

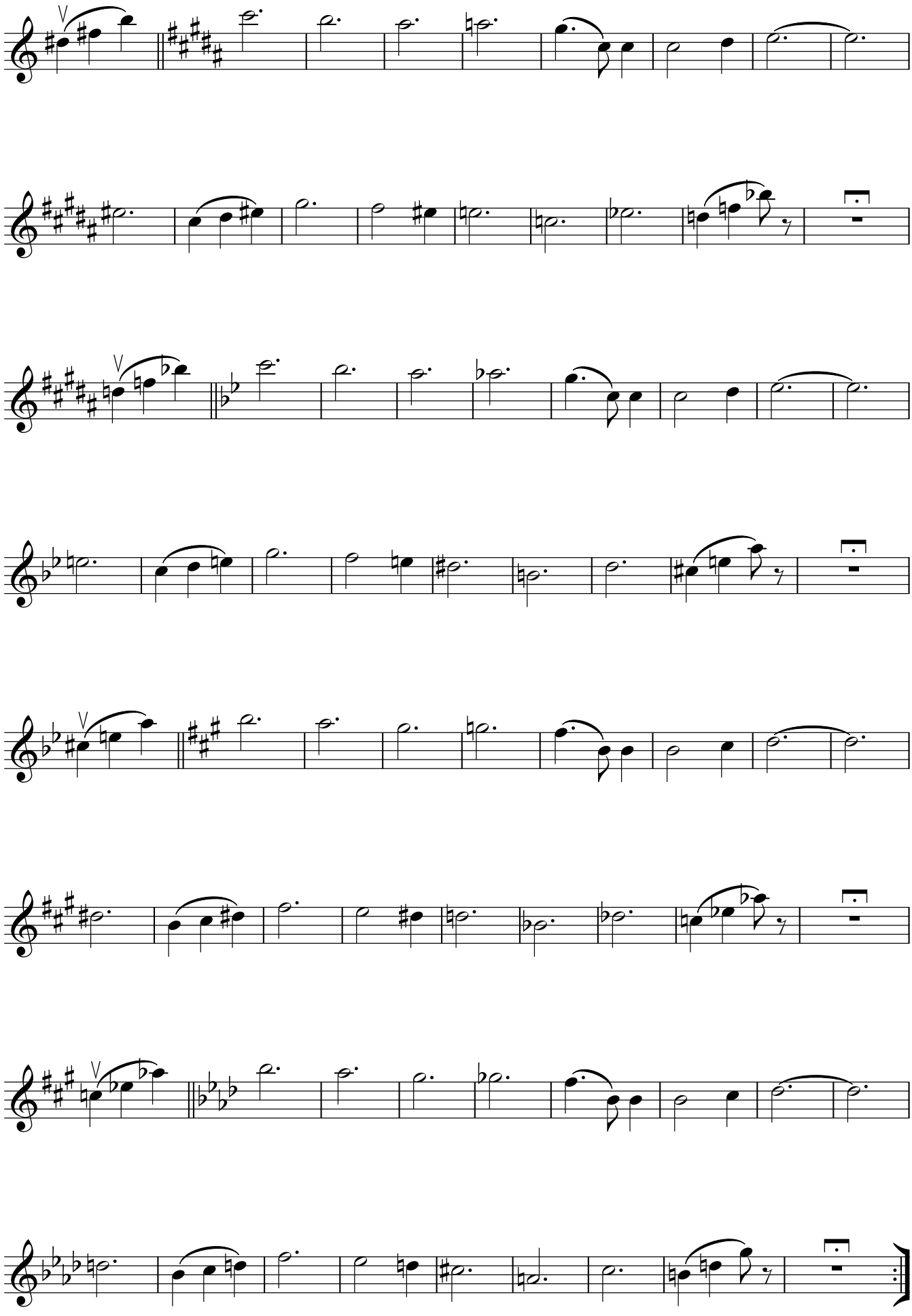
# SLIPPING AWAY

J. Simon van der Walt

♩ = 140 or less: slow and nostalgic





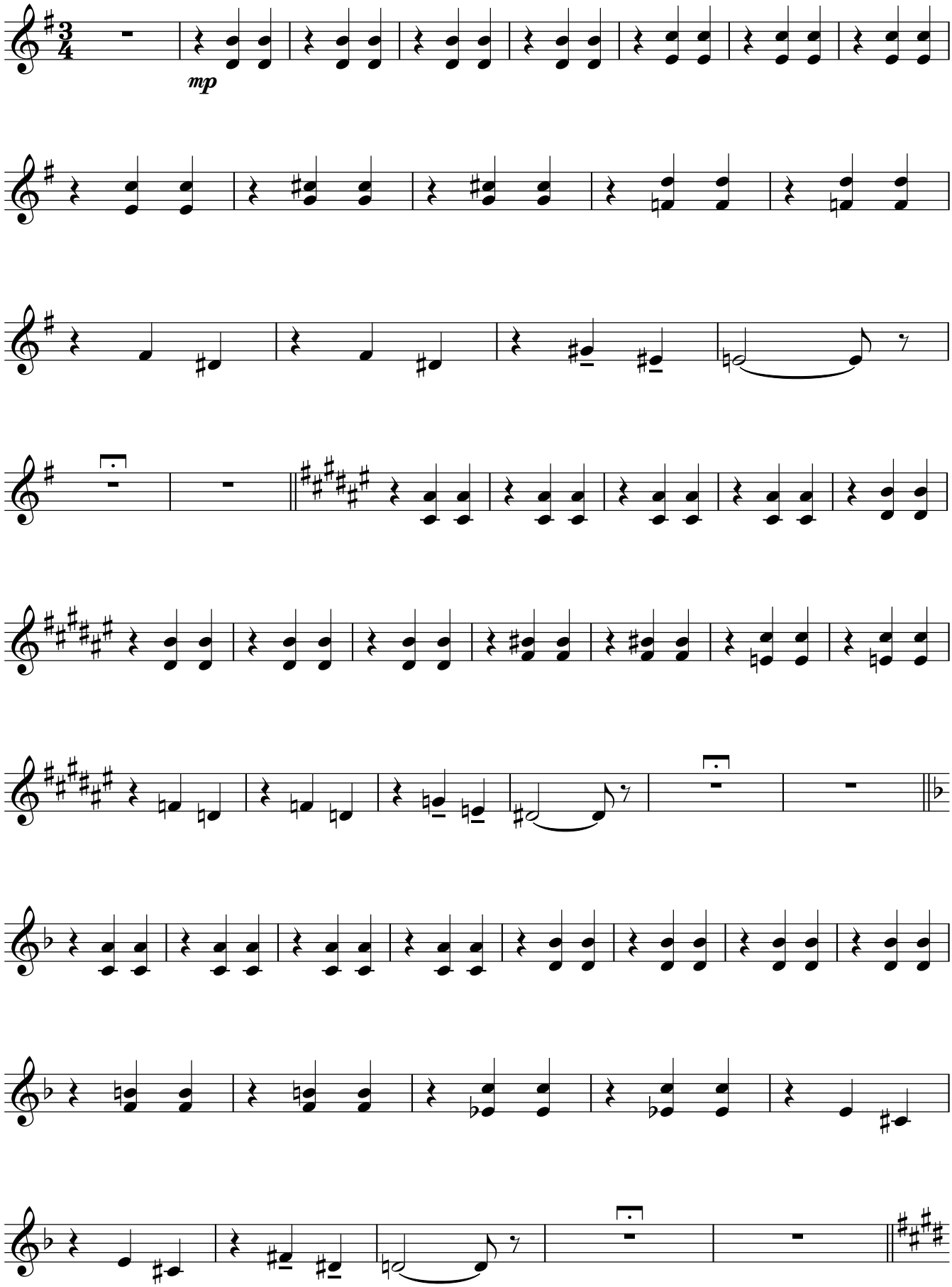


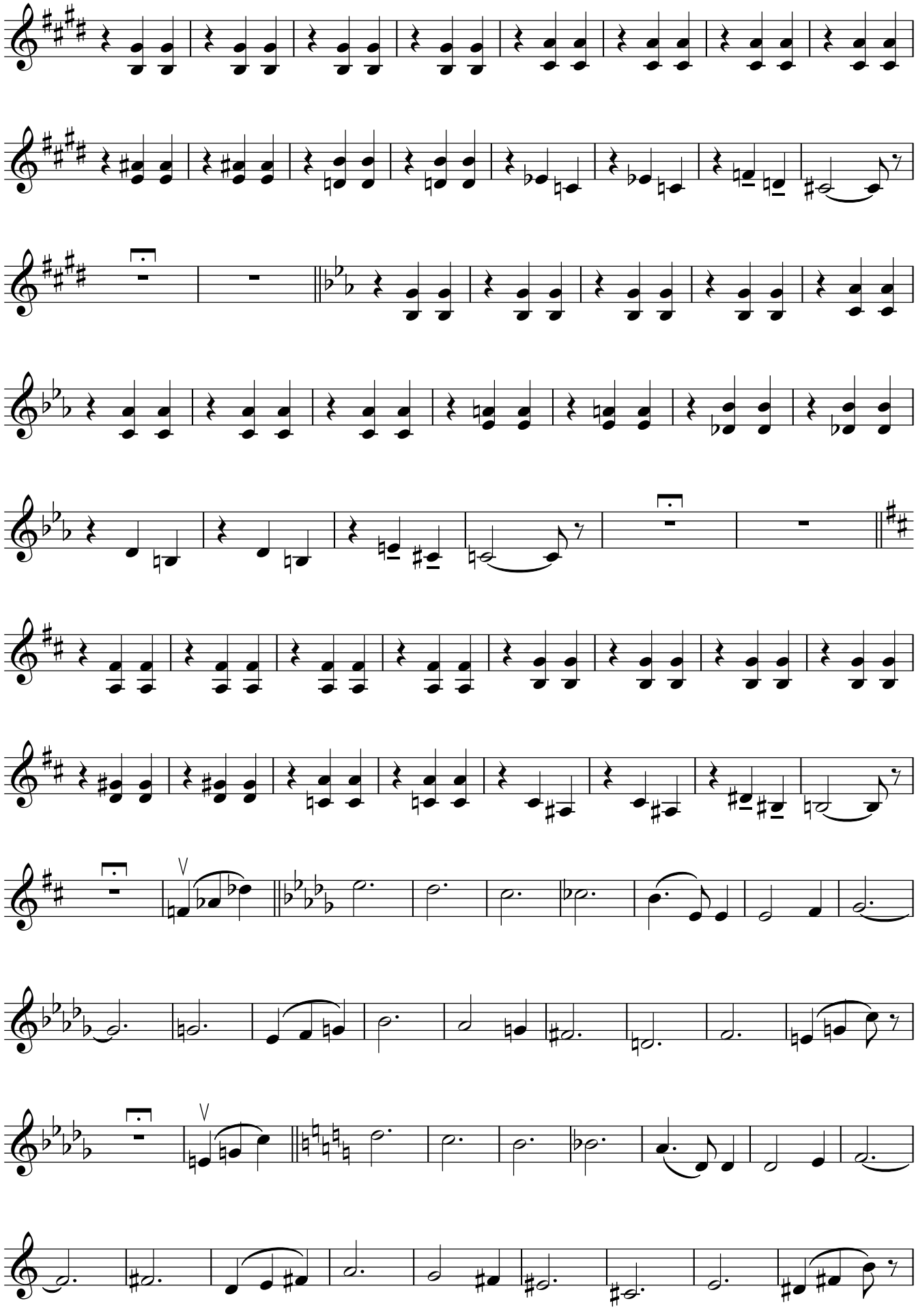
Violin 2

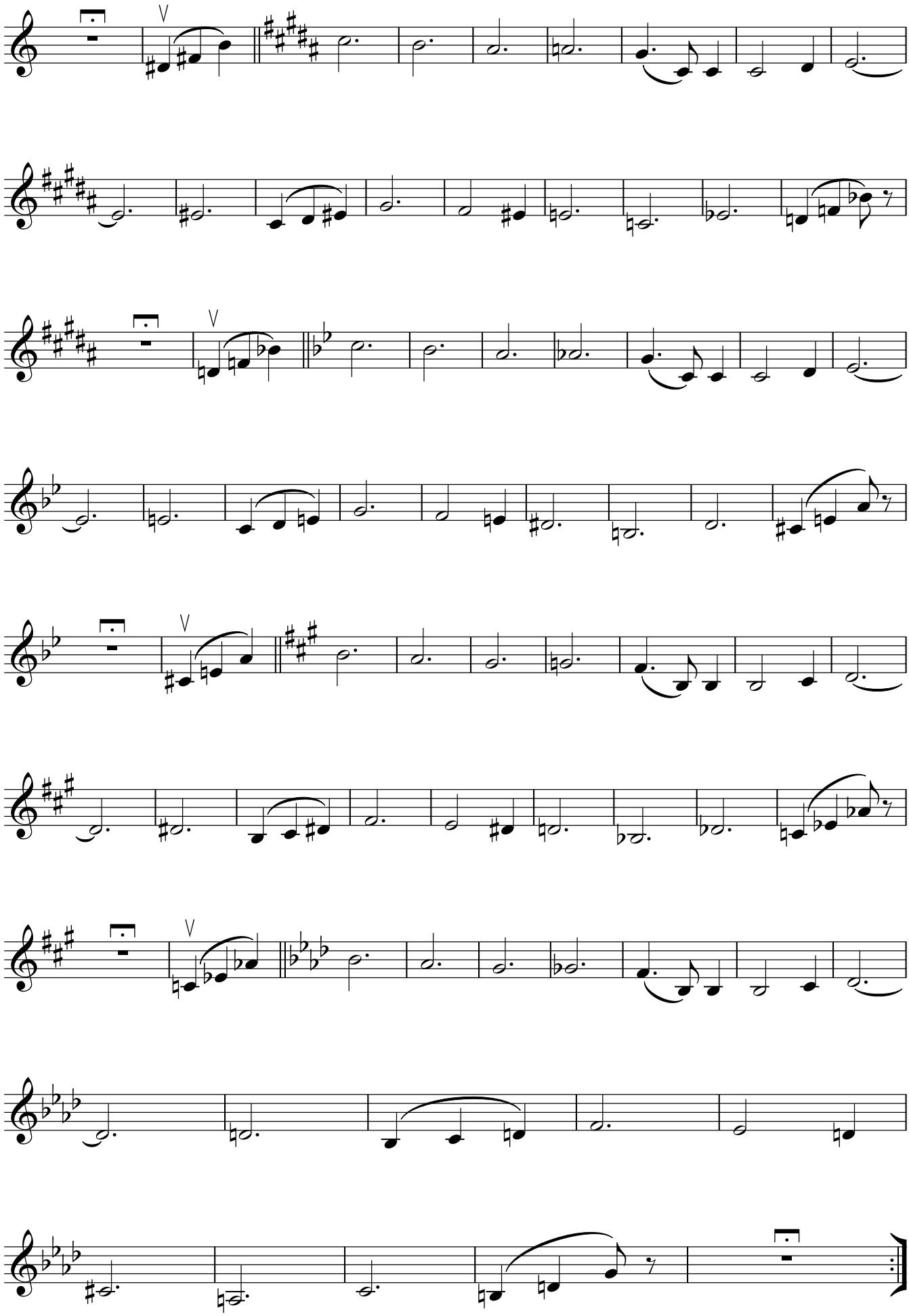
# SLIPPING AWAY

J. Simon van der Walt

♩ = 140 or less: slow and nostalgic







```
// setup and testing
s.quit;
Server.local.options.device = "JackRouter";
s.boot;
// Server.killAll;
// s.meter;
().play;
// evil tritone metronome
(
Pbind( \midinote, Pseq([90,84,84,84],inf),
       \legato, 0.005,
).play(TempoClock(140/60));
)

////////////////////// initialise, ready to go

(

{
~slipbuf = Buffer.read(s, Document.current.dir.asString++"/
slippingsynth5ch.aiff");
0.5.wait;
~outrobuf = Buffer.read(s, Document.current.dir.asString++"/outro.aif");
0.5.wait;
}.fork(AppClock);

// ~slipbuf.play;
// ~outrobuf.play;

SynthDef(\patbuf, {
    | freq, slices=16, slice=16, gate=1, pan=0, buf, gain=0.5 |
    var env, start, len, sig, rate, splay;
    rate = freq/(60.midicps);
    len = BufFrames.kr(buf);
    start = (len / slices * slice);
    sig = PlayBuf.ar(5, buf, BufRateScale.kr(buf) * rate, startPos: start,
loop: 1);
    splay = Splay.ar(sig, 0.7);
    env = Linen.kr(gate, attackTime: 0.5, releaseTime: 0.5, doneAction: 2);
    // Out.ar(0, [sig[0],sig[4]] * env  * gain) //choose channels to play
0-4
    Out.ar(0, sig * env  * gain) // five channel
    // Out.ar(0, splay * env * gain) // for stereo testing

}).add;


a=SynthDef(\playthrough, {| inputs=#[0,1,2,3,4], through=0.5 |
           var sig = SoundIn.ar(inputs);
           Out.ar(0, sig * through) // five channel
           // Out.ar(0, (Splay.ar(sig) * through)) // stereo testing
}).play;
```

```
//a.set(\through, 0.5);
//a.set(\through, 0.0);
//a.set(\through, 0.2); // set this lower down the page

SynthDef(\yap0, {
            | input=0, gate=1, thru=0 |
            var livesound, env, ad=0.05, pan;
            livesound = SoundIn.ar(input);
            env = EnvGen.kr(Env.asr(ad,1.0,ad,'linear'), gate, doneAction:
2);
            Out.ar(0, (livesound * env * thru)) //channel 0
            // Out.ar(0, Pan2.ar((livesound * env * thru),-1)) //panned
}).add;

SynthDef(\yap1, {
            | input=1, gate=1, thru=0 |
            var livesound, env, ad=0.05;
            livesound = SoundIn.ar(input);
            env = EnvGen.kr(Env.asr(ad,1.0,ad,'linear'), gate, doneAction:
2);
            Out.ar(1, (livesound * env * thru)) //channel 1
            // Out.ar(0, Pan2.ar((livesound * env * thru),-0.5)) //panned
}).add;

SynthDef(\yap2, {
            | input=2, gate=1, thru=0 |
            var livesound, env, ad=0.05;
            livesound = SoundIn.ar(input);
            env = EnvGen.kr(Env.asr(ad,1.0,ad,'linear'), gate, doneAction:
2);
            Out.ar(2, (livesound * env * thru)) //channel 1
            // Out.ar(0, Pan2.ar((livesound * env * thru),0)) //panned
}).add;

SynthDef(\yap3, {
            | input=3, gate=1, thru=0 |
            var livesound, env, ad=0.05;
            livesound = SoundIn.ar(input);
            env = EnvGen.kr(Env.asr(ad,1.0,ad,'linear'), gate, doneAction:
2);
            Out.ar(3, (livesound * env * thru)) //channel 1
            // Out.ar(0, Pan2.ar((livesound * env * thru),0.5)) //panned
}).add;

SynthDef(\yap4, {
            | input=4, gate=1, thru=0 |
            var livesound, env, ad=0.05;
            livesound = SoundIn.ar(input);
            env = EnvGen.kr(Env.asr(ad,1.0,ad,'linear'), gate, doneAction:
2);
            Out.ar(4, (livesound * env * thru)) //channel 1
```

```
            // Out.ar(0, Pan2.ar((livesound * env * thru),1)) //panned
}).add;

)

// go! - on leader's downbeat
(

a.set(\through, 0.2); // maybe

{
t=TempoClock(140/60);
~tsyn = PmonoArtic(\yap0,
        \thru, Pser([1,0],inf),
        \dur, 0.5,
        \legato, 0.2
).play(t);
~start = t.seconds;

u=TempoClock(140/60);
~usyn = PmonoArtic(\yap1,
        \thru, Pser([1,0],inf),
        \dur, 0.5
).play(u);

v=TempoClock(140/60);
~vsyn = PmonoArtic(\yap2,
        \thru, Pser([1,0],inf),
        \dur, 0.5
).play(v);

w=TempoClock(140/60);
~wsyn = PmonoArtic(\yap3,
        \thru, Pser([1,0],inf),
        \dur, 0.5
).play(w);

x=TempoClock(140/60);
~xsyn = PmonoArtic(\yap4,
        \thru, Pser([1,0],inf),
        \dur, 0.5
).play(x);



t.sched(t.timeToNextBeat, {t.sync((140/60)*(3/2), 8); nil});
9.wait;
("01 "++(t.seconds - ~start).asString).postln;
u.sched(u.timeToNextBeat, {u.sync((140/60)*(2/3), 8); nil});
9.wait;
("02 "++(t.seconds - ~start).asString).postln;
v.sched(v.timeToNextBeat, {v.sync((140/60)*(4/3), 8); nil});
```

```
9.wait;
("03 "++(t.seconds - ~start).asString).postln;
w.sched(w.timeToNextBeat, {w.sync((140/60)*(3/4), 8); nil});
9.wait;
("04 "++(t.seconds - ~start).asString).postln;
x.sched(x.timeToNextBeat, {x.sync((140/60)*(5/3), 8); nil});
9.wait;
("05 "++(t.seconds - ~start).asString).postln;
t.sched(t.timeToNextBeat, {t.sync((140/60)*(4/5), 8); nil});
9.wait;
("06 "++(t.seconds - ~start).asString).postln;
u.sched(u.timeToNextBeat, {u.sync((140/60)*(3/2), 8); nil});
9.wait;
("07 "++(t.seconds - ~start).asString).postln;
v.sched(v.timeToNextBeat, {v.sync((140/60)*(2/3), 8); nil});
9.wait;
("08 "++(t.seconds - ~start).asString).postln;
w.sched(w.timeToNextBeat, {w.sync((140/60)*(4/3), 8); nil});
9.wait;
("09 "++(t.seconds - ~start).asString).postln;
x.sched(x.timeToNextBeat, {x.sync((140/60)*(3/4), 8); nil});
9.wait;
("10 "++(t.seconds - ~start).asString).postln;
t.sched(t.timeToNextBeat, {t.sync((140/60)*(5/3), 8); nil});
9.wait;
("11 "++(t.seconds - ~start).asString).postln;
u.sched(u.timeToNextBeat, {u.sync((140/60)*(4/5), 8); nil});
9.wait;
("12 "++(t.seconds - ~start).asString).postln;
v.sched(v.timeToNextBeat, {v.sync((140/60)*(3/4), 8); nil});
9.wait;
("13 "++(t.seconds - ~start).asString).postln;
w.sched(w.timeToNextBeat, {w.sync((140/60)*(5/3), 8); nil});
9.wait;
("14 "++(t.seconds - ~start).asString).postln;
x.sched(x.timeToNextBeat, {x.sync((140/60)*(4/5), 8); nil});
9.wait;
("15 "++(t.seconds - ~start).asString).postln;
t.sched(t.timeToNextBeat, {t.sync((140/60)*(1), 8); nil});
u.sched(u.timeToNextBeat, {u.sync((140/60)*(1), 9); nil});
v.sched(v.timeToNextBeat, {v.sync((140/60)*(1), 10); nil});
w.sched(w.timeToNextBeat, {w.sync((140/60)*(1), 11); nil});
x.sched(x.timeToNextBeat, {x.sync((140/60)*(1), 12); nil});
60.wait;
("16 "++(t.seconds - ~start).asString).postln;
t.sched(t.timeToNextBeat, {t.sync((140/60)*(3/2), 8); nil});
u.sched(u.timeToNextBeat, {u.sync((140/60)*(2/3), 8); nil});
v.sched(v.timeToNextBeat, {v.sync((140/60)*(4/3), 8); nil});
w.sched(w.timeToNextBeat, {w.sync((140/60)*(3/4), 8); nil});
x.sched(x.timeToNextBeat, {x.sync((140/60)*(5/3), 8); nil});
20.wait;
("17 "++(t.seconds - ~start).asString).postln;
```

```
t.sched(t.timeToNextBeat, {t.sync((140/60)*(1), 3); nil});
u.sched(u.timeToNextBeat, {u.sync((140/60)*(1), 5); nil});
v.sched(v.timeToNextBeat, {v.sync((140/60)*(1), 6); nil});
w.sched(w.timeToNextBeat, {w.sync((140/60)*(1), 7); nil});
x.sched(x.timeToNextBeat, {x.sync((140/60)*(1), 8); nil});
20.wait;
("18 "++(t.seconds - ~start).asString).postln;

~tsyn.stop;
~usyn.stop;
~vsyn.stop;
~wsyn.stop;
~xsyn.stop;

j=TempoClock(30/60);

Pbind( \instrument, \patbuf,
       \buf, ~slipbuf,
       \slice, Prand((1..16), 1), // just one random slice
       \dur, 10
       ).trace.play(j);

4.wait;
a.set(\through, 0.5); //maybe
4.wait;

// made subsequent interjections quieter

Pbind( \instrument, \patbuf,
       \buf, ~slipbuf,
       \slice, Pxrand((1..16), inf), // now inf
       \dur, Pseq((10..1).mirror, inf),
       \legato, Pseq((1..10).mirror/10, 1),
       \gain, 0.2
       ).trace.play(j);

180.wait;
("outro "++(t.seconds - ~start).asString).postln;

{Out.ar(0, ((PlayBuf.ar(1, ~outrobuf))* 0.4) ! 5)}.play; // outro 5ch

// mute string playthrough and free buffers for tidy
56.wait;
a.set(\through, 0.0);
~slipbuf.free;
~outrobuf.free;
}.fork;
)
```