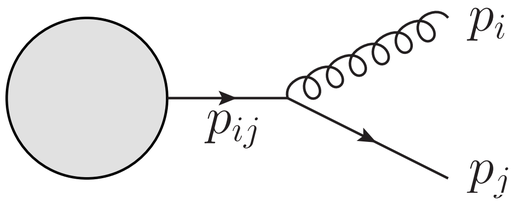
Competing Sudakov Veto Algorithms

Rob Verheyen

Shower Probabilities

Goal: Sample from the distribution



Ordering scale

Starting scale

$$p(t|u) = f(t)\exp\left(-\int_t^u d\tau f(\tau)\right) = f(t)\Delta_f(u,t)$$

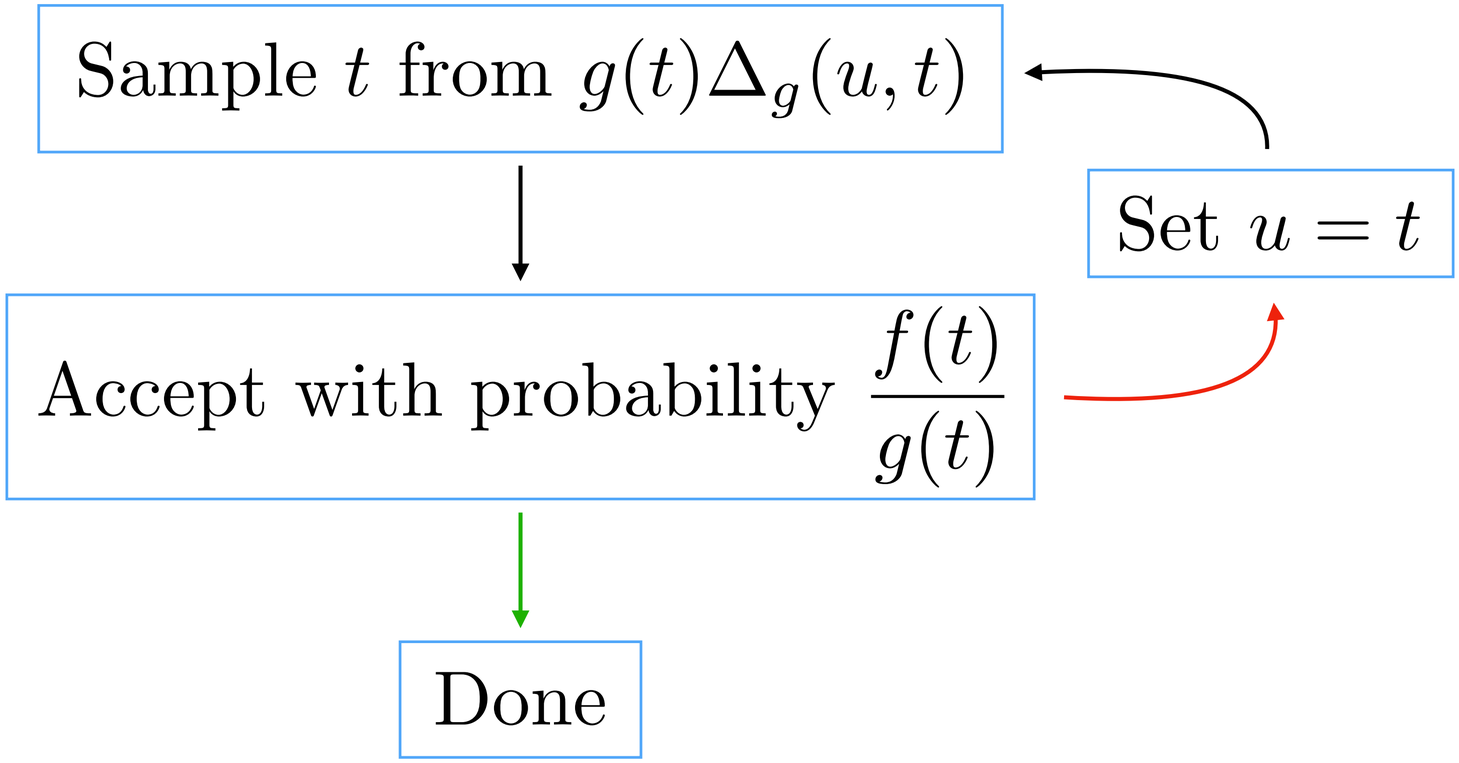
Branching kernel

Sudakov factor

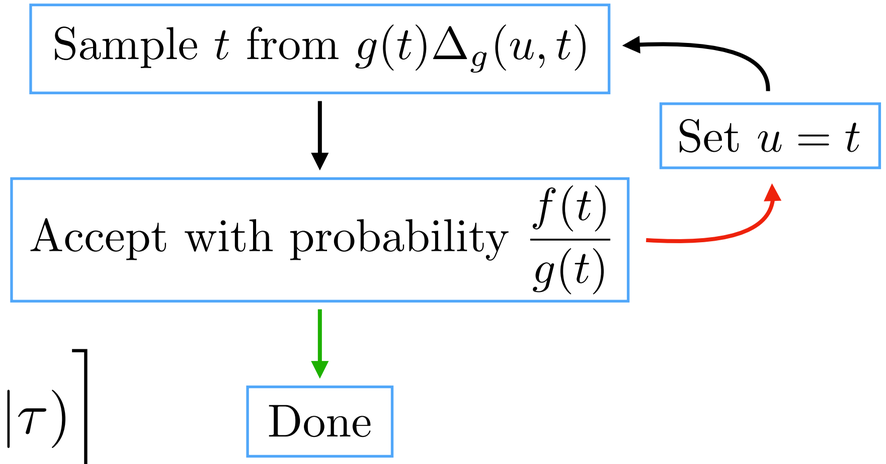
The branching kernel $f(t)$ is usually too complicated to sample directly

The Sudakov Veto Algorithm

Solution: Find a *simple* overestimate $g(t) > f(t)$ and do the following:



The Sudakov Veto Algorithm

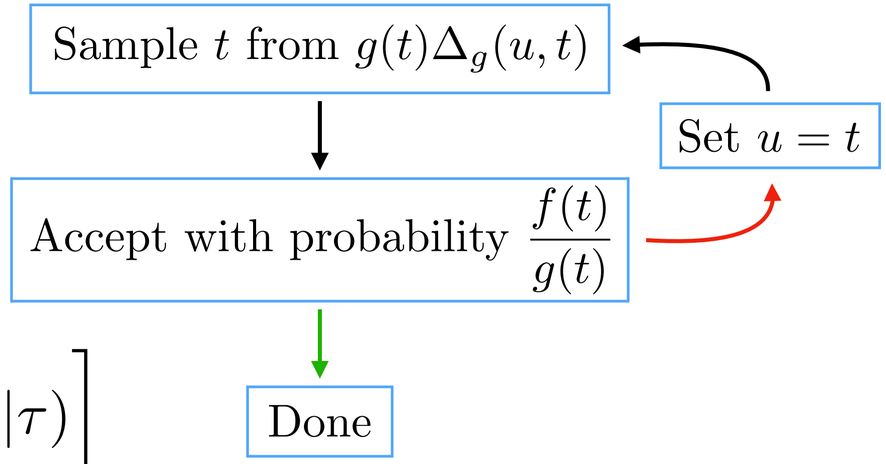


Why does that work? Write out probability recursively:

$$p(t|u) = \int_0^u d\tau g(\tau)\Delta_g(u,\tau)\left[\frac{f(\tau)}{g(\tau)}\delta(\tau - t) + \left(1 - \frac{f(\tau)}{g(\tau)}\right)p(t|\tau)\right]$$

Markovian

The Sudakov Veto Algorithm



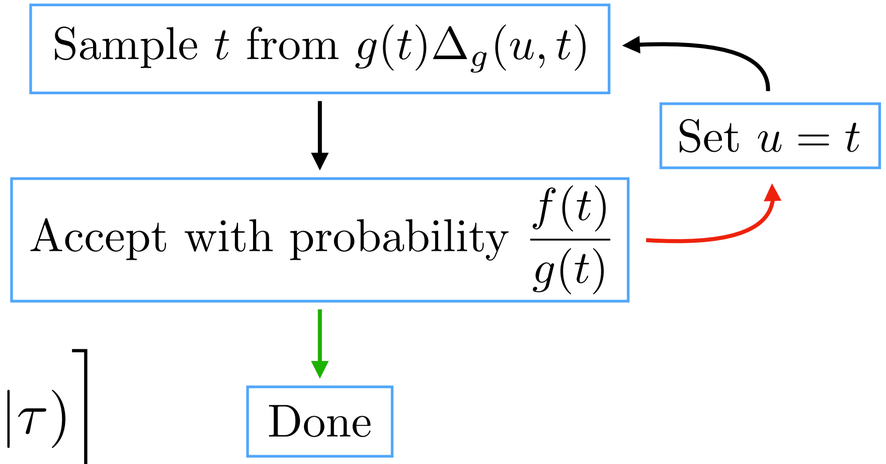
Why does that work? Write out probability recursively:

$$p(t|u) = \int_0^u d\tau g(\tau)\Delta_g(u,\tau)\left[\frac{f(\tau)}{g(\tau)}\delta(\tau - t) + \left(1 - \frac{f(\tau)}{g(\tau)}\right)p(t|\tau)\right]$$

Find differential equation

$$\frac{\partial}{\partial u}p(t|u) = f(t)\delta(t-u) - f(u)p(t|u)$$

The Sudakov Veto Algorithm



Why does that work? Write out probability recursively:

$$p(t|u) = \int_0^u d\tau g(\tau)\Delta_g(u,\tau)\left[\frac{f(\tau)}{g(\tau)}\delta(\tau - t) + \left(1 - \frac{f(\tau)}{g(\tau)}\right)p(t|\tau)\right]$$

Find differential equation

$$\frac{\partial}{\partial u}p(t|u) = f(t)\delta(t-u) - f(u)p(t|u)$$

General(ish) solution

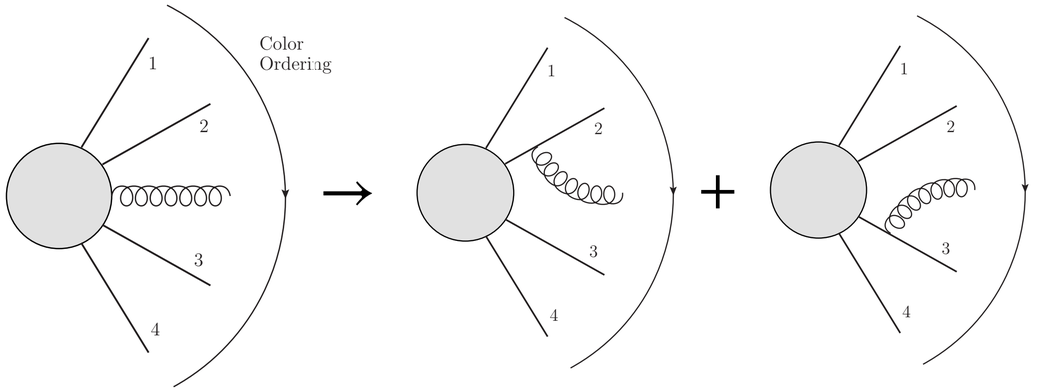
$$p(t|u) = f(t)\Delta_f(u,t)\theta(\sigma < t < u) + p_0(t,\sigma)$$

Cutoff term

Competition

Many extensions possible and required for practical implementation

- More variables
- Weighted veto algorithm
- Etc..



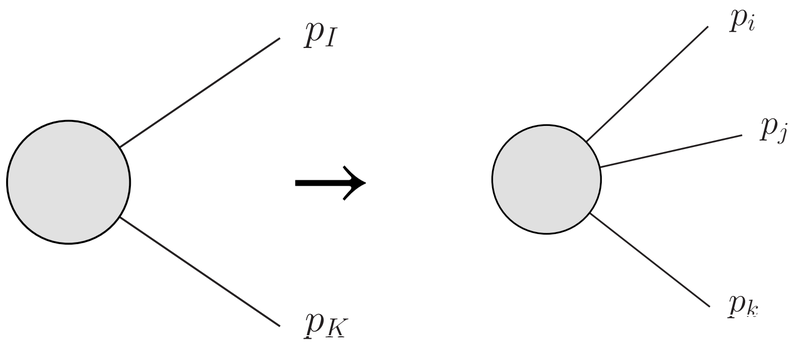
Focus this talk: *Competition*

$$\tilde{p}(t|u) = \tilde{f}(t)\exp\left(-\int_t^u d\tau \tilde{f}(\tau)\right) \quad \text{where} \quad \tilde{f}(t) = \sum_i^N f_i(t)$$

Partons/Dipoles/Antennae

Toy Shower

Vincia-based simple final-state toy shower

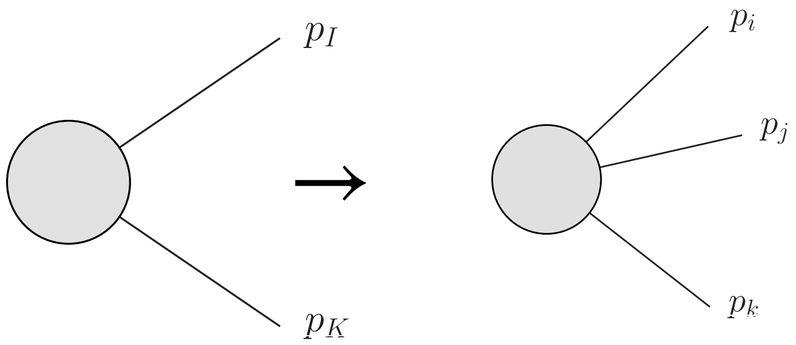


- Branching kernel: Soft eikonal $s_{ab} = 2p_a \cdot p_b$

$$f(s_{ij}, s_{jk}) = a\frac{s_{ik}}{s_{ij}s_{jk}} \qquad g(s_{ij}, s_{jk}) = a\frac{s_{IK}}{s_{ij}s_{jk}}$$

Toy Shower

Vincia-based simple final-state toy shower



- Branching kernel: Soft eikonal

$$s_{ab} = 2p_a \cdot p_b$$

$$f(s_{ij}, s_{jk}) = a\frac{s_{ik}}{s_{ij}s_{jk}} \qquad g(s_{ij}, s_{jk}) = a\frac{s_{IK}}{s_{ij}s_{jk}}$$

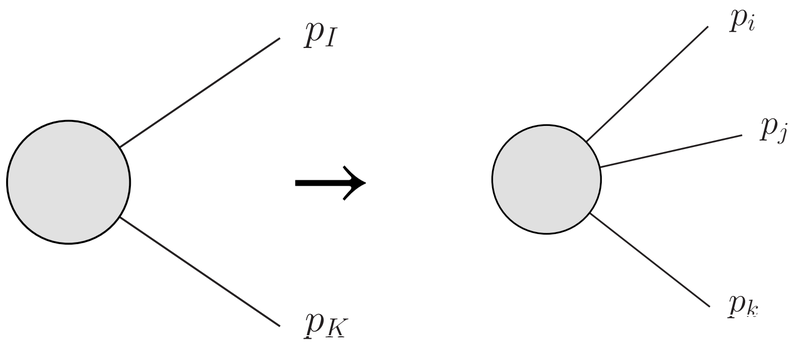
- Phase space factorisation: Vincia

$$\begin{aligned} d\Phi_{n+1} &= d\Phi_n \times ds_{ij}\, ds_{jk}\, d\varphi \\ &= d\Phi_n \times dp_\perp^2\, dz\, d\varphi \end{aligned}$$

$$p_\perp^2 = \frac{s_{ij}s_{jk}}{s_{IK}} \qquad z = \frac{s_{jk}}{s_{IK}}$$

Toy Shower

Vincia-based simple final-state toy shower



- Branching kernel: Soft eikonal

$$s_{ab} = 2p_a \cdot p_b$$

$$f(s_{ij}, s_{jk}) = a\frac{s_{ik}}{s_{ij}s_{jk}} \qquad g(s_{ij}, s_{jk}) = a\frac{s_{IK}}{s_{ij}s_{jk}}$$

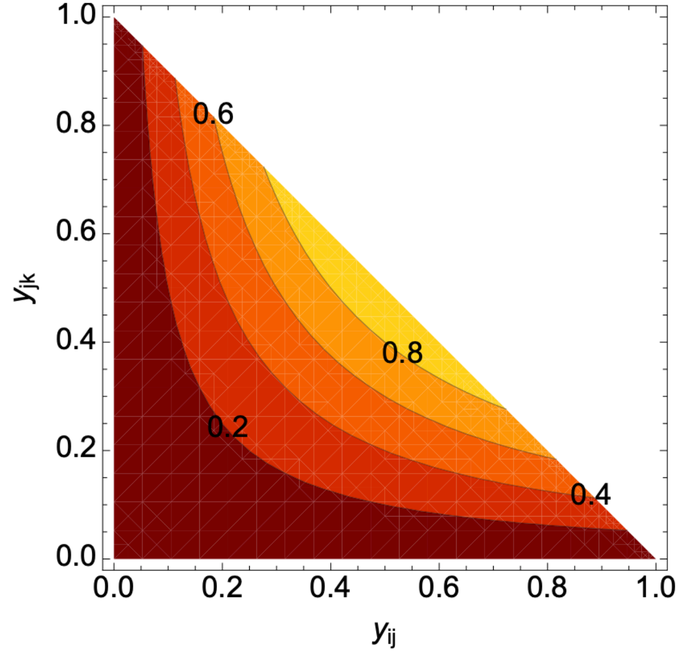
- Phase space factorisation: Vincia

$$d\Phi_{n+1} = d\Phi_n \times ds_{ij}\, ds_{jk}\, d\varphi$$
$$= d\Phi_n \times dp_\perp^2\, dz\, d\varphi$$

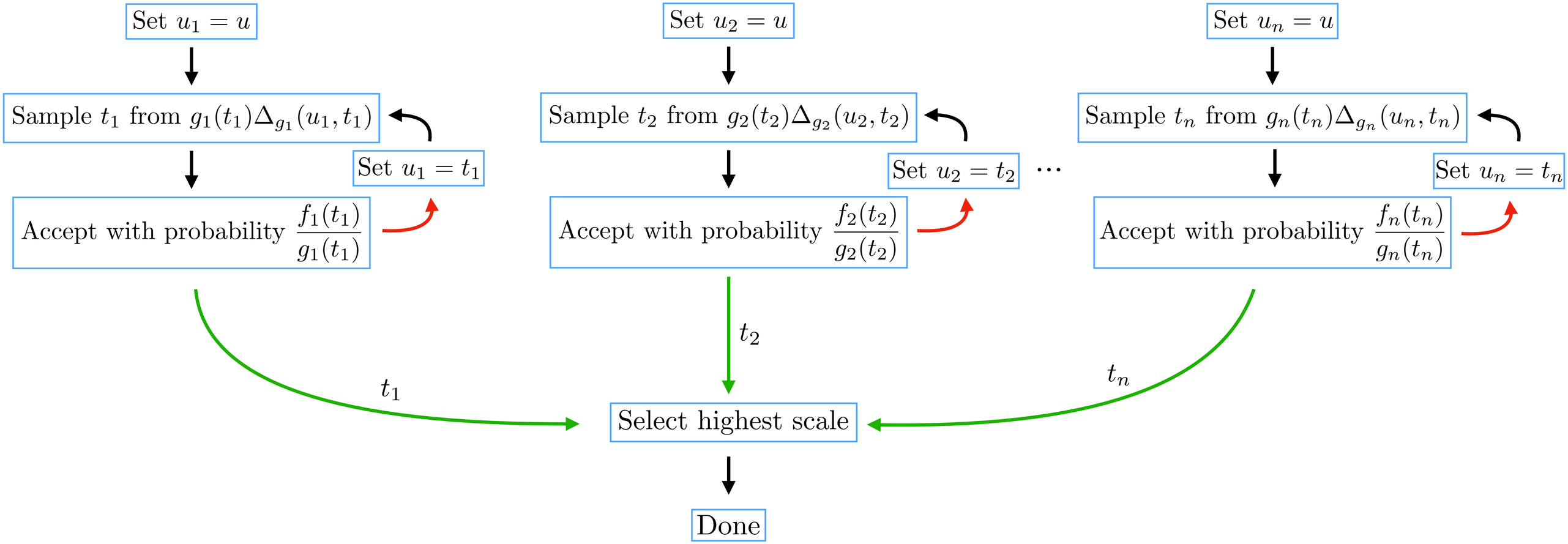
$$p_\perp^2 = \frac{s_{ij}s_{jk}}{s_{IK}} \qquad z = \frac{s_{jk}}{s_{IK}}$$

- Phase space boundaries

$$z_\pm = \frac{1}{2}\left(1 \pm \sqrt{1 - 4\frac{p_\perp^2}{s_{IK}}}\right)$$



Veto-Max


Set $u_1 = u$
Sample t_1 from $g_1(t_1)\Delta_{g_1}(u_1, t_1)$
Set $u_1 = t_1$
Accept with probability $\frac{f_1(t_1)}{g_1(t_1)}$
Set $u_2 = u$
Sample t_2 from $g_2(t_2)\Delta_{g_2}(u_2, t_2)$
Set $u_2 = t_2$
Accept with probability $\frac{f_2(t_2)}{g_2(t_2)}$
$\cdots$
Set $u_n = u$
Sample t_n from $g_n(t_n)\Delta_{g_n}(u_n, t_n)$
Set $u_n = t_n$
Accept with probability $\frac{f_n(t_n)}{g_n(t_n)}$
t_1
t_2
t_n
Select highest scale
Done

Veto-Max

Why does that work?

$$\left[\prod_{i=1}^{n}\int_{0}^{u} dt_i f_i(t_i)\Delta_{f_i}(u,t_i)\right]\sum_{j=1}^{n}\left[\prod_{k\neq j}\theta\left(t_j > t_k\right)\right]\delta\left(t-t_j\right)$$

Veto-Max

Why does that work?

$$\left[\prod_{i=1}^{n}\int_{0}^{u} dt_i f_i(t_i)\Delta_{f_i}(u,t_i)\right]\sum_{j=1}^{n}\left[\prod_{k\neq j}\theta\left(t_j > t_k\right)\right]\delta\left(t-t_j\right)$$

$$=\sum_{i=1}^{n}\left[\prod_{j\neq i}\int_{0}^{t_i} dt_j f(t_j)\Delta_{f_j}(u,t_j)\right]\int_{0}^{u} dt_i f_i(t_i)\Delta_{f_i}(u,t_i)\,\delta\left(t-t_i\right)$$

Veto-Max

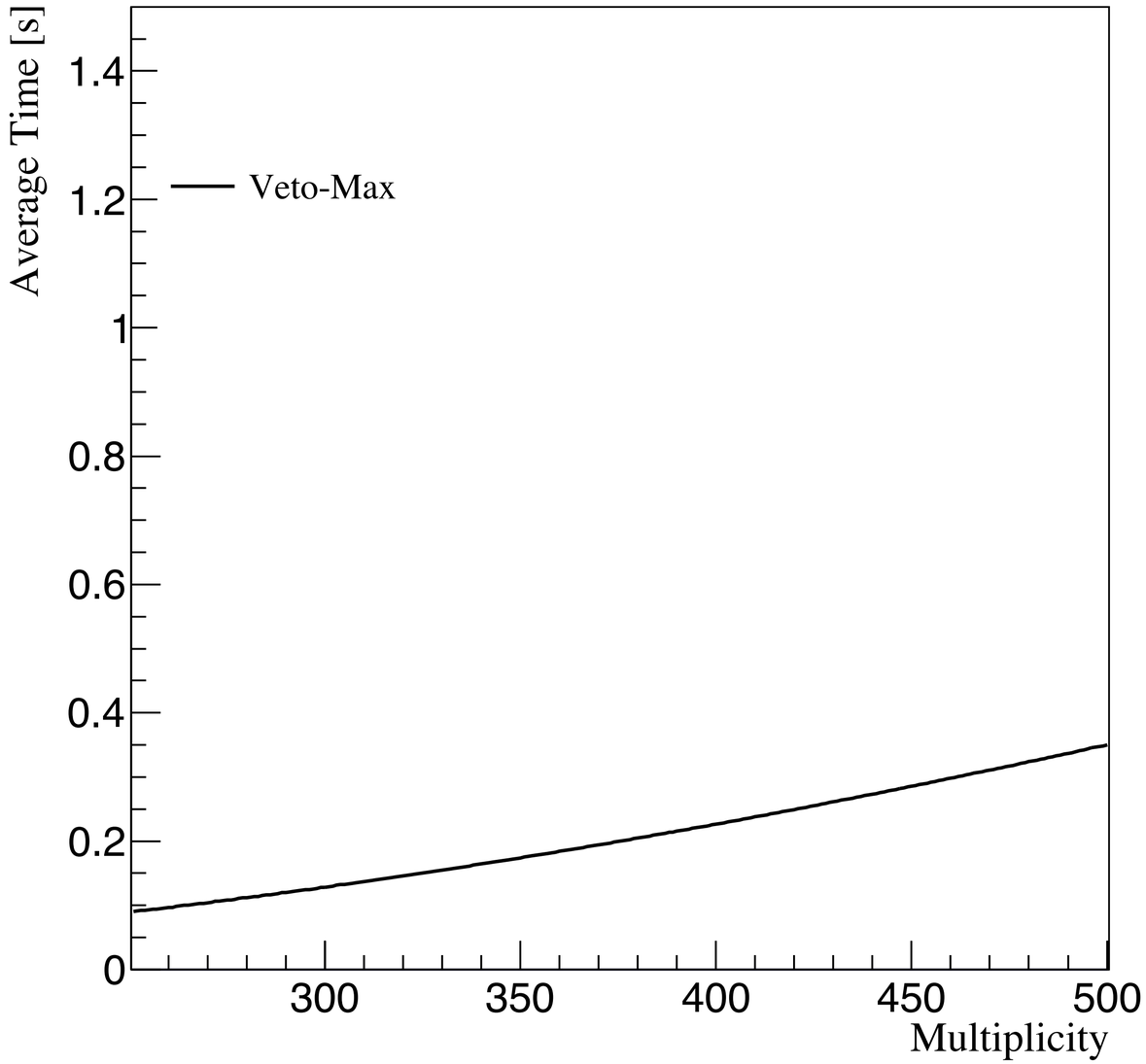
Why does that work?

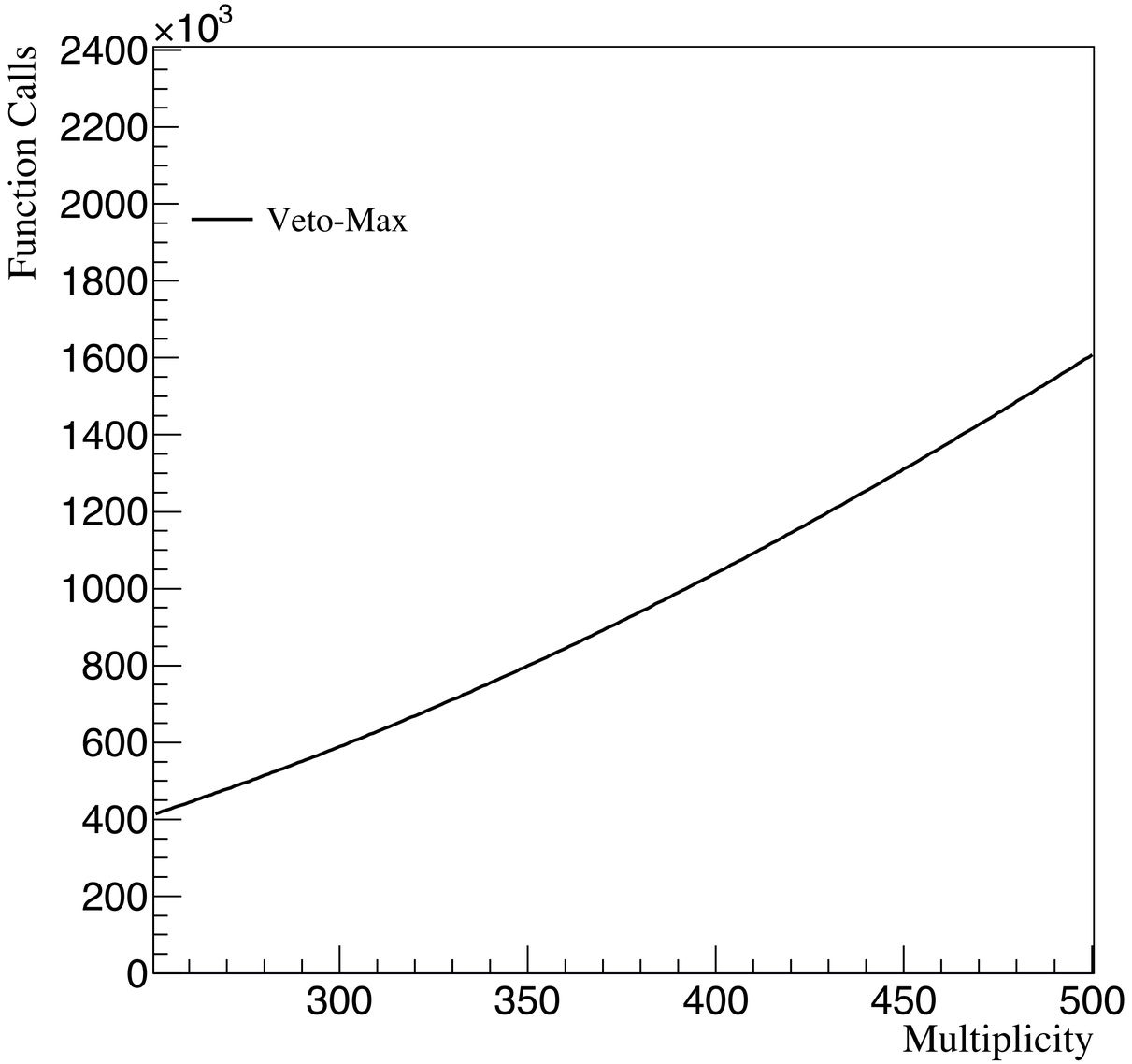
$$\left[\prod_{i=1}^{n}\int_{0}^{u} dt_i f_i(t_i)\Delta_{f_i}(u,t_i)\right]\sum_{j=1}^{n}\left[\prod_{k\neq j}\theta\left(t_j > t_k\right)\right]\delta\left(t-t_j\right)$$

$$=\sum_{i=1}^{n}\left[\prod_{j\neq i}\int_{0}^{t_i} dt_j f(t_j)\Delta_{f_j}(u,t_j)\right]\int_{0}^{u} dt_i f_i(t_i)\Delta_{f_i}(u,t_i)\,\delta\left(t-t_i\right)$$

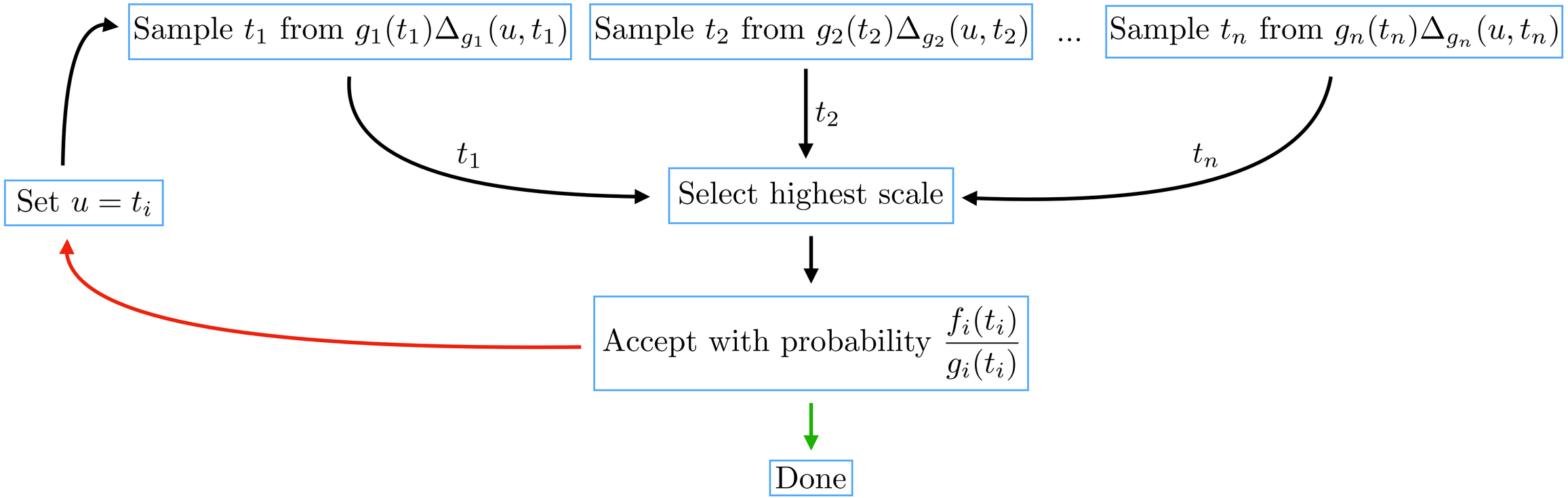
$$=\sum_{i=1}^{n} f_i(t)\Delta_{f_i}(u,t)\prod_{j\neq i}\Delta_{f_j}(u,t)=\tilde{p}(t|u)$$

Timings

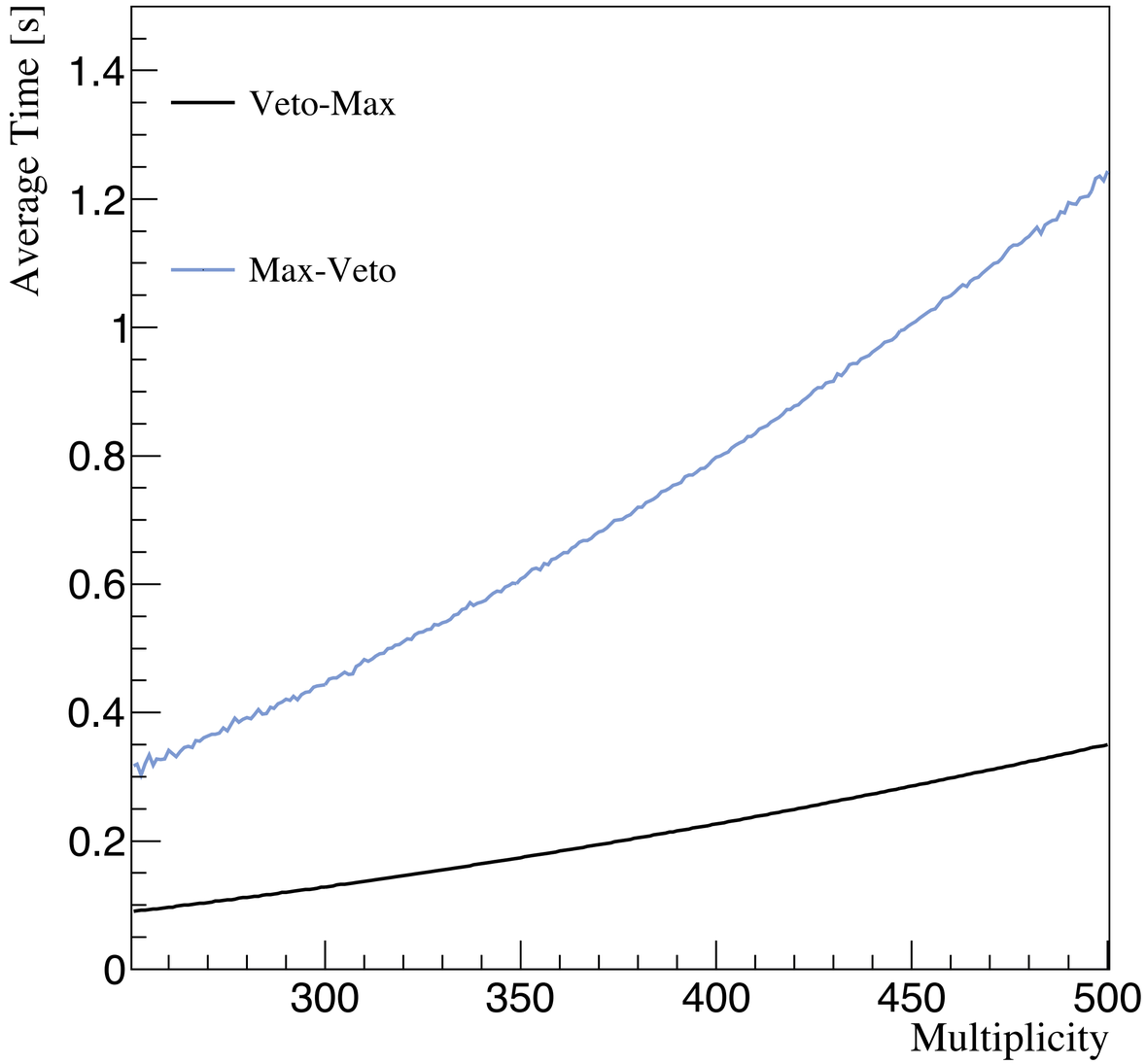

Average Time [s]
Veto-Max
1.4
1.2
1
0.8
0.6
0.4
0.2
0
300
350
400
450
500
Multiplicity

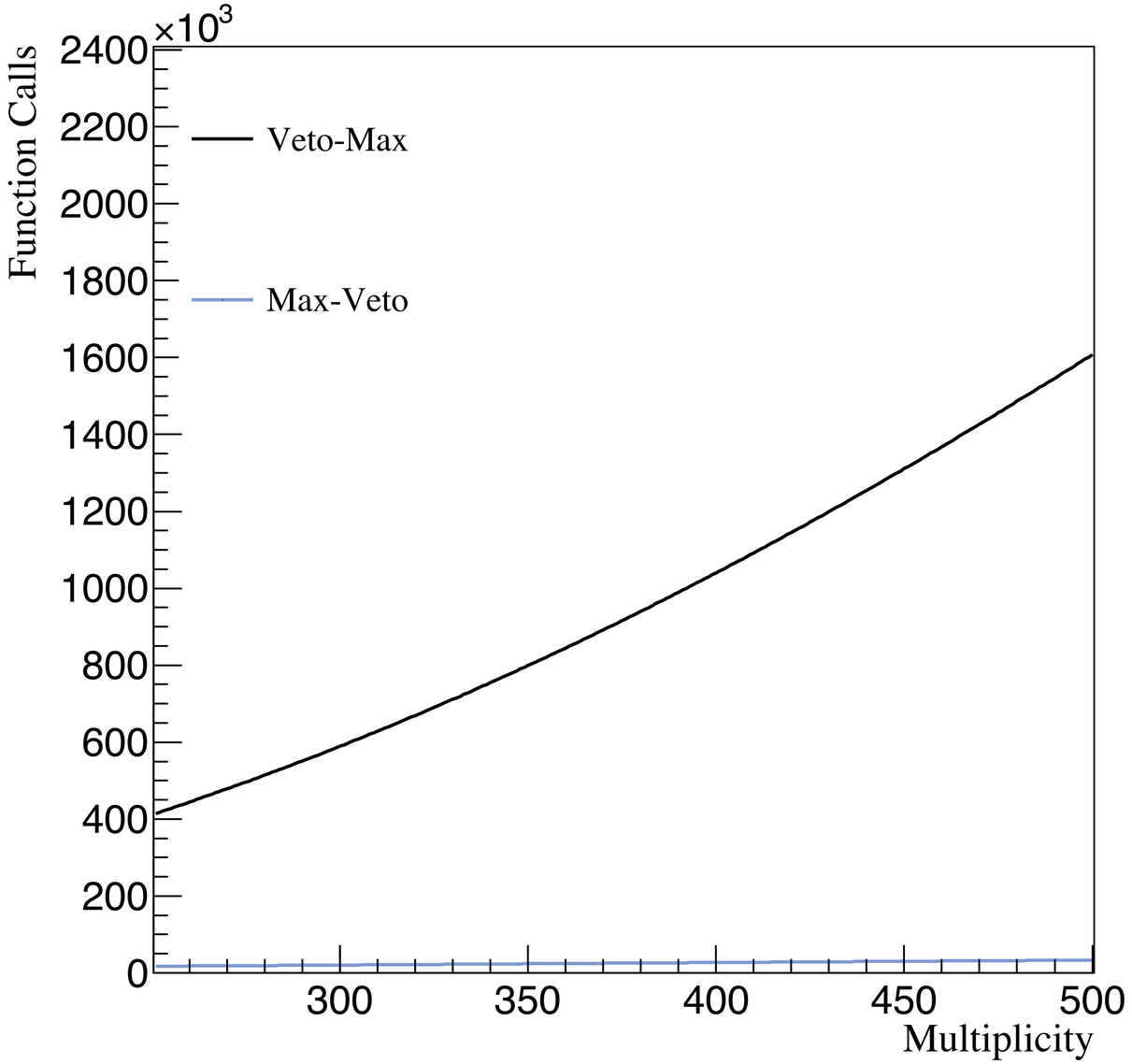

Function Calls
×10^3
Veto-Max
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
300
350
400
450
500
Multiplicity

Max-Veto

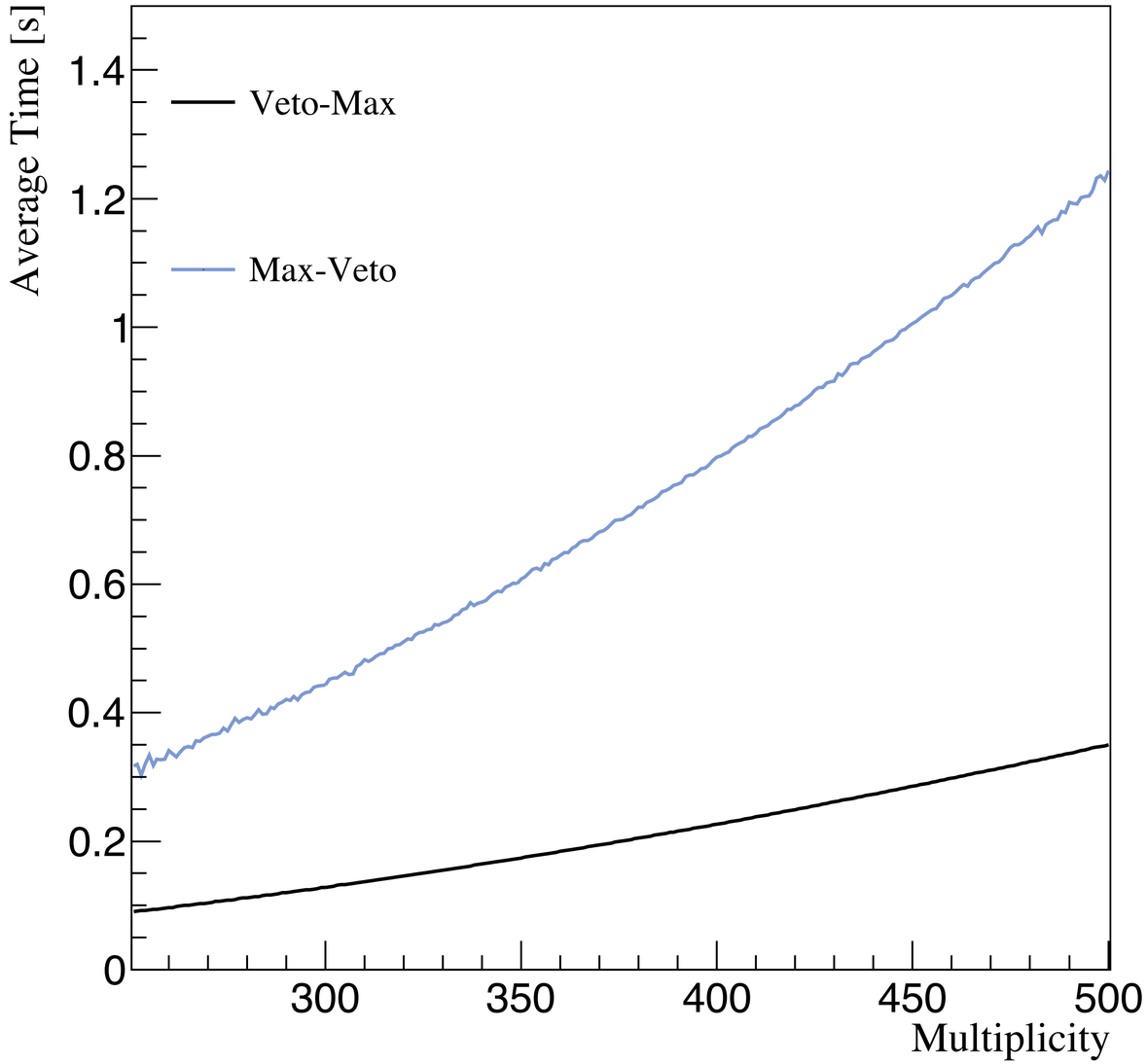

Sample t_1 from $g_1(t_1)\Delta_{g_1}(u, t_1)$
Sample t_2 from $g_2(t_2)\Delta_{g_2}(u, t_2)$
...
Sample t_n from $g_n(t_n)\Delta_{g_n}(u, t_n)$
t_1
t_2
t_n
Select highest scale
Set $u = t_i$
Accept with probability $\frac{f_i(t_i)}{g_i(t_i)}$
Done

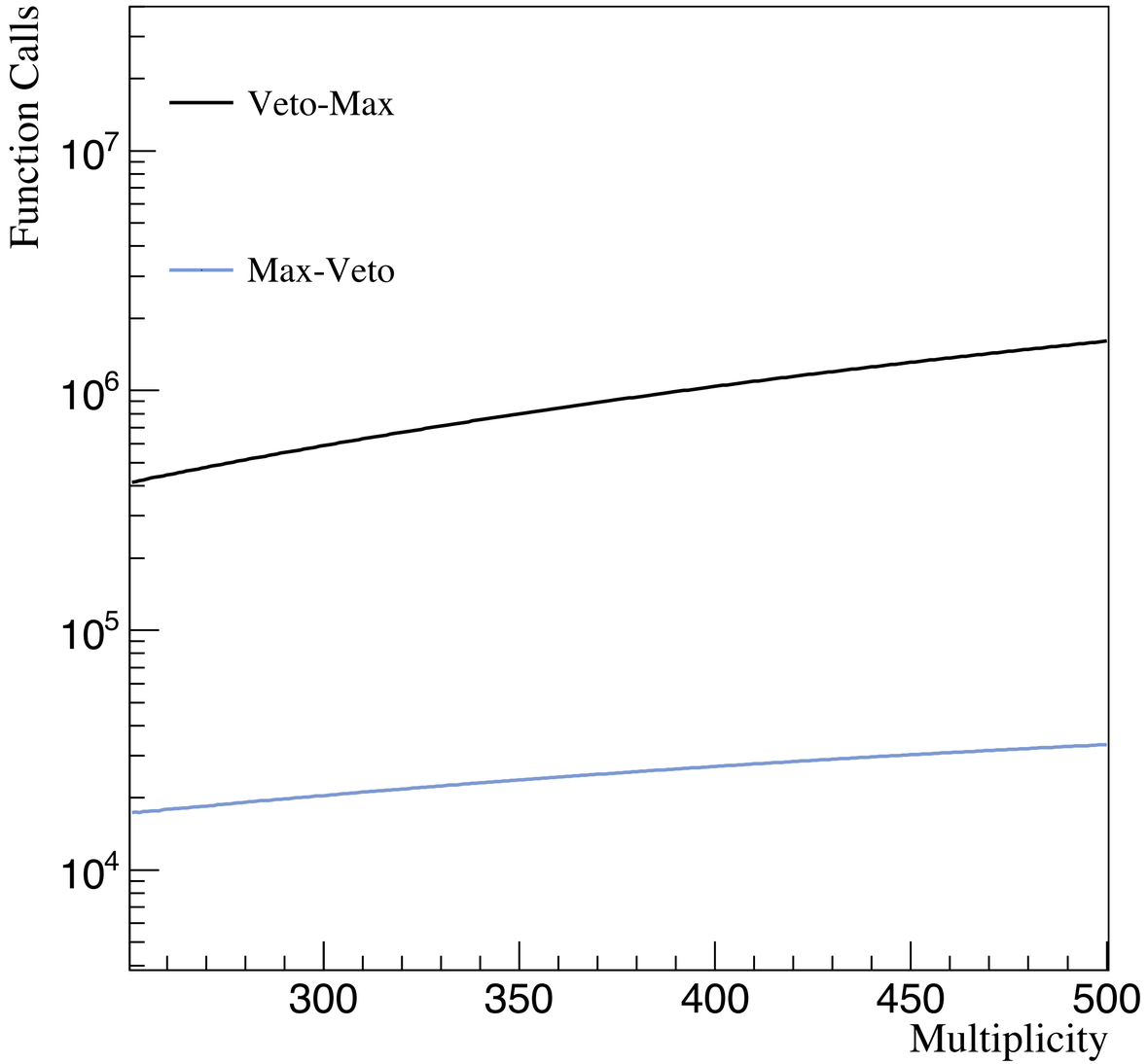
Timings


Average Time [s]
Veto-Max
Max-Veto
1.4
1.2
1
0.8
0.6
0.4
0.2
0
300
350
400
450
500
Multiplicity

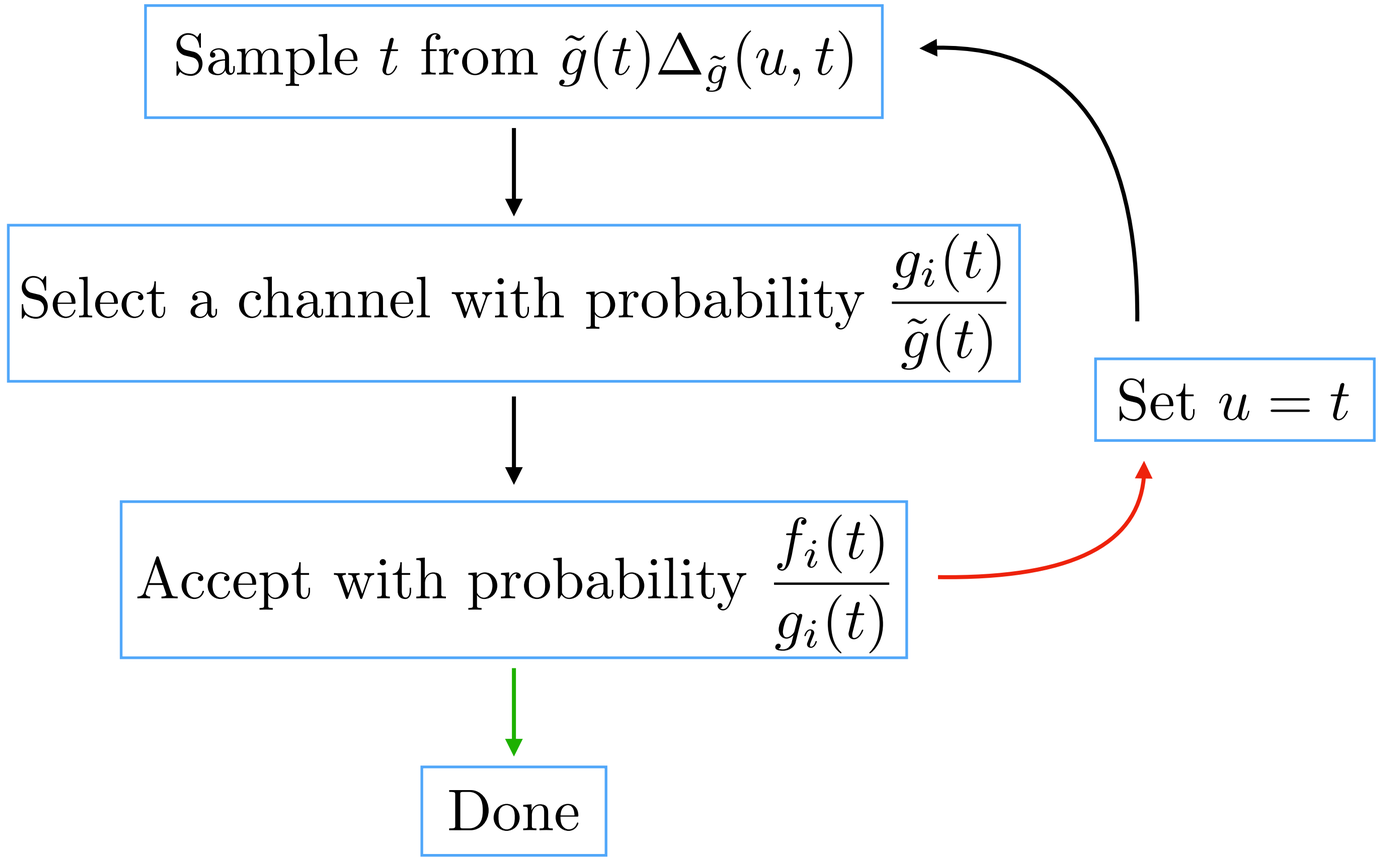

Function Calls
×10^3
Veto-Max
Max-Veto
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
300
350
400
450
500
Multiplicity

Timings

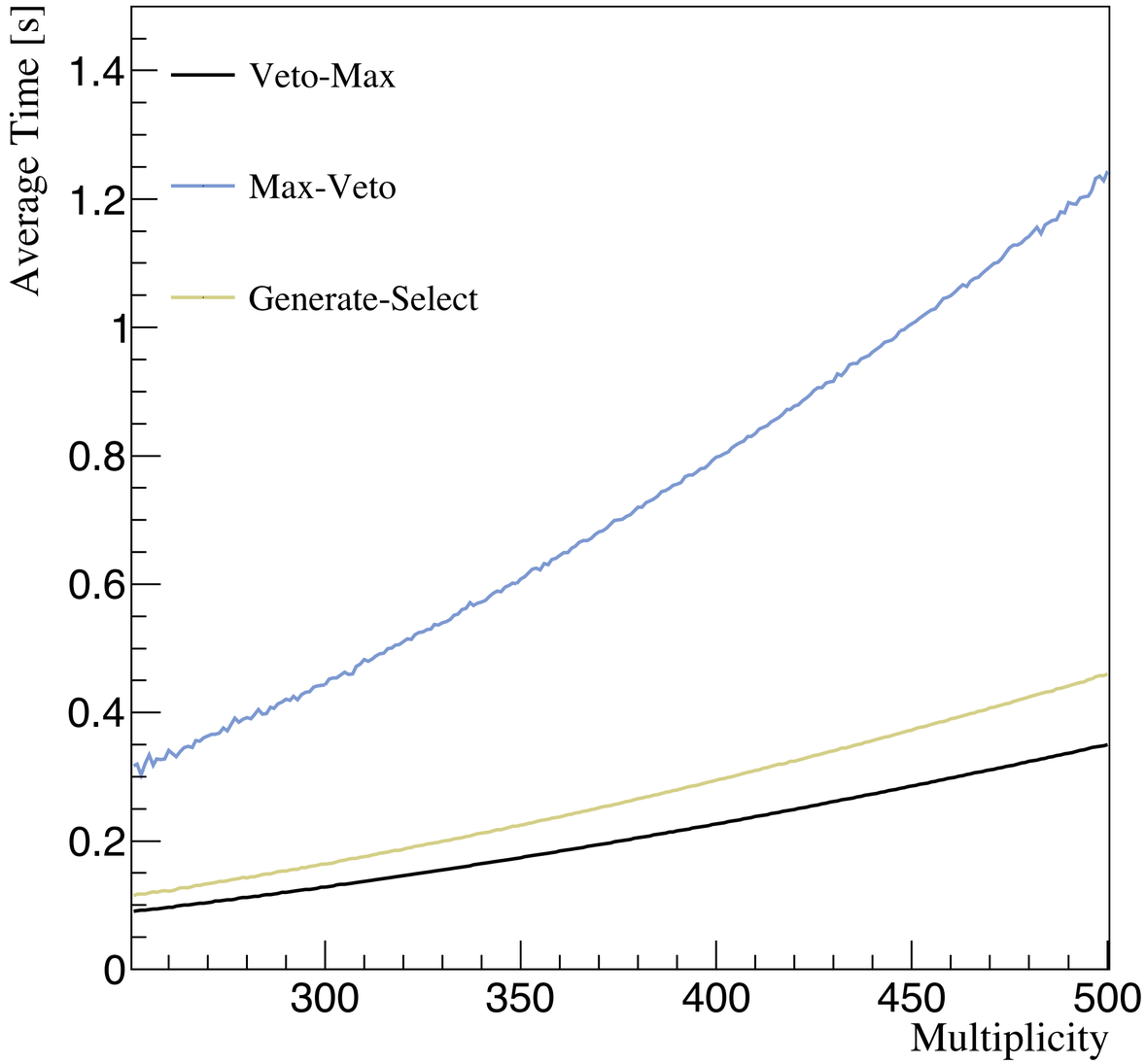

Average Time [s]
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Veto-Max
Max-Veto
300
350
400
450
500
Multiplicity

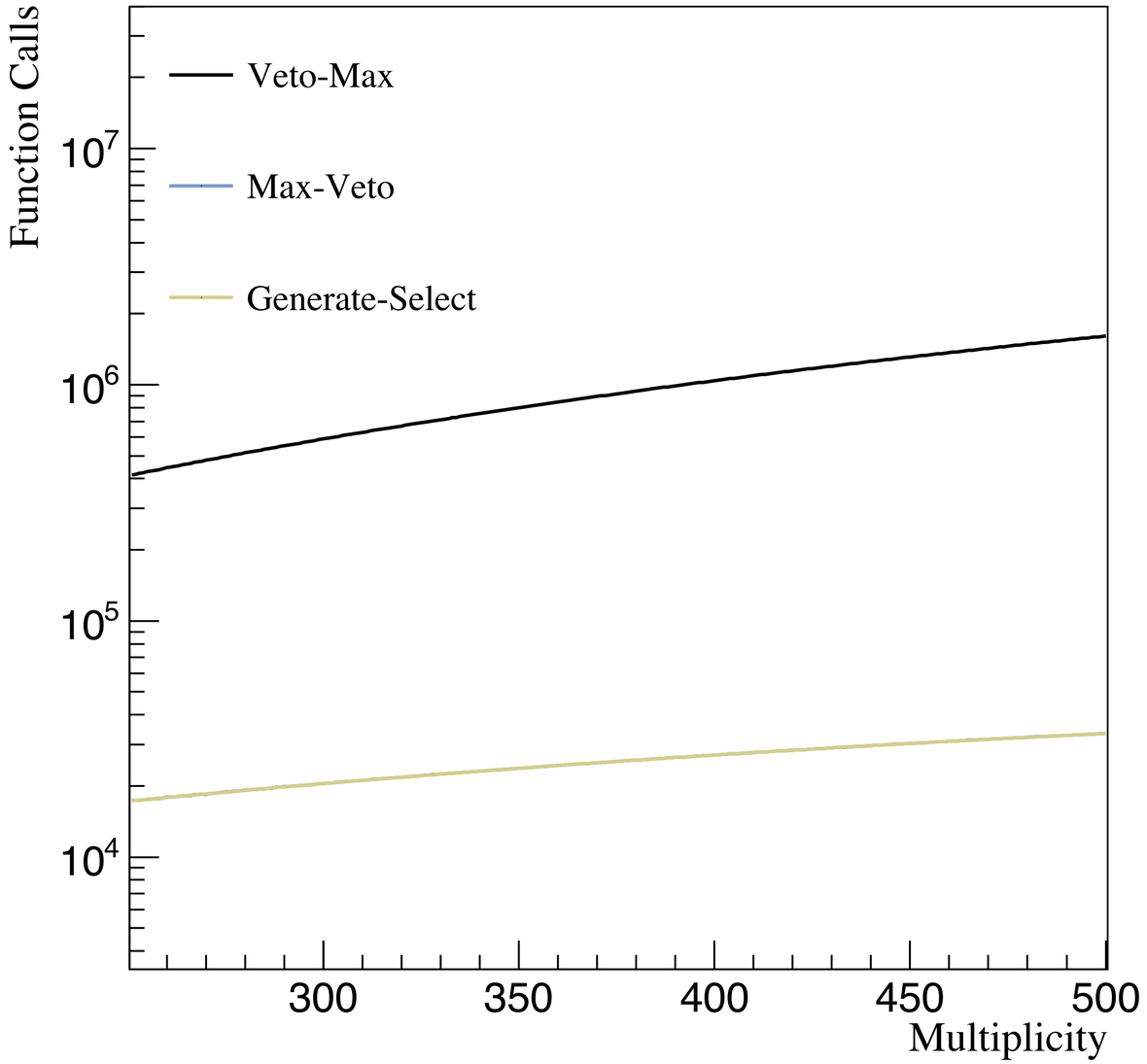

Function Calls
10^7
10^6
10^5
10^4
Veto-Max
Max-Veto
300
350
400
450
500
Multiplicity

Generate-Select



Timings


Average Time [s]
Veto-Max
Max-Veto
Generate-Select
1.4
1.2
1
0.8
0.6
0.4
0.2
0
300
350
400
450
500
Multiplicity

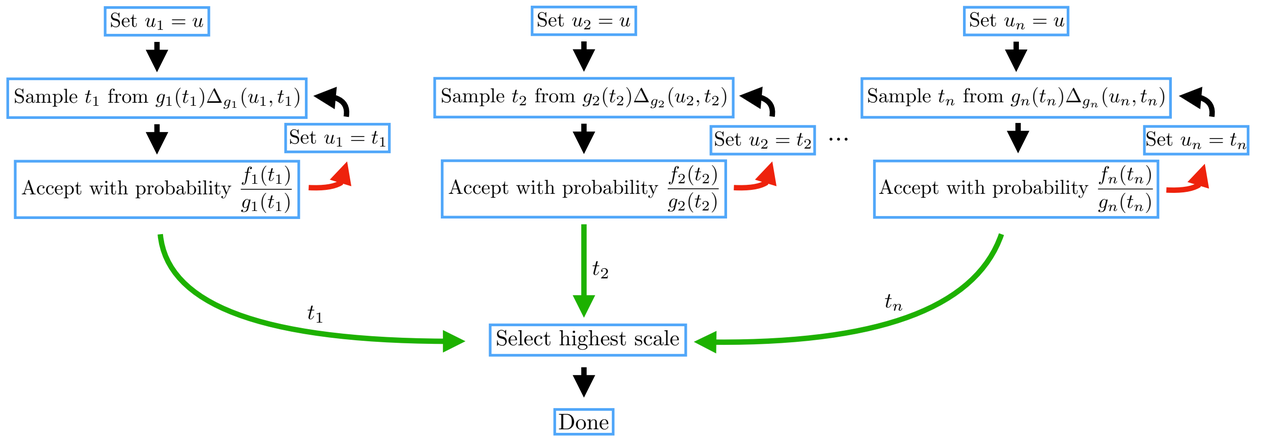

Function Calls
Veto-Max
Max-Veto
Generate-Select
10^7
10^6
10^5
10^4
300
350
400
450
500
Multiplicity

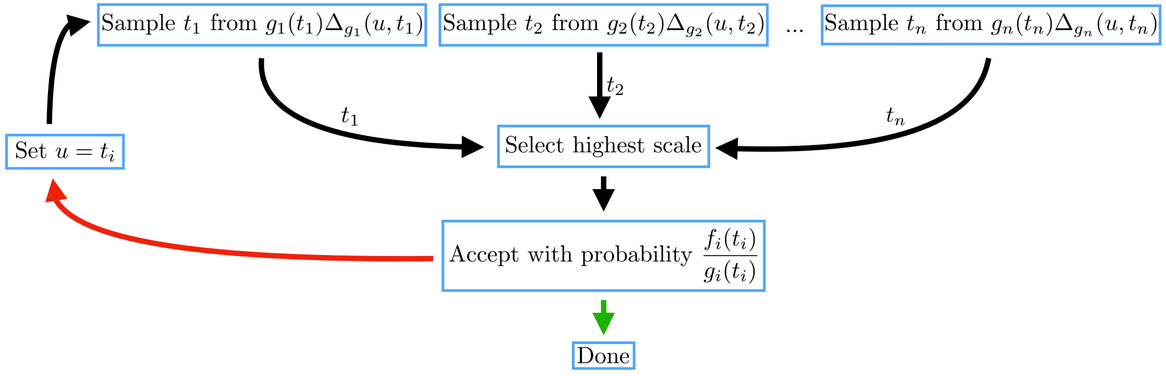
Scale Saving (Veto-Max & Max-Veto)

Scales of unselected emissions can be saved

$$f(t)\Delta_f(u,t) = \Delta_f(u,\bar{t}) \times f(t)\Delta_f(\bar{t},t)$$

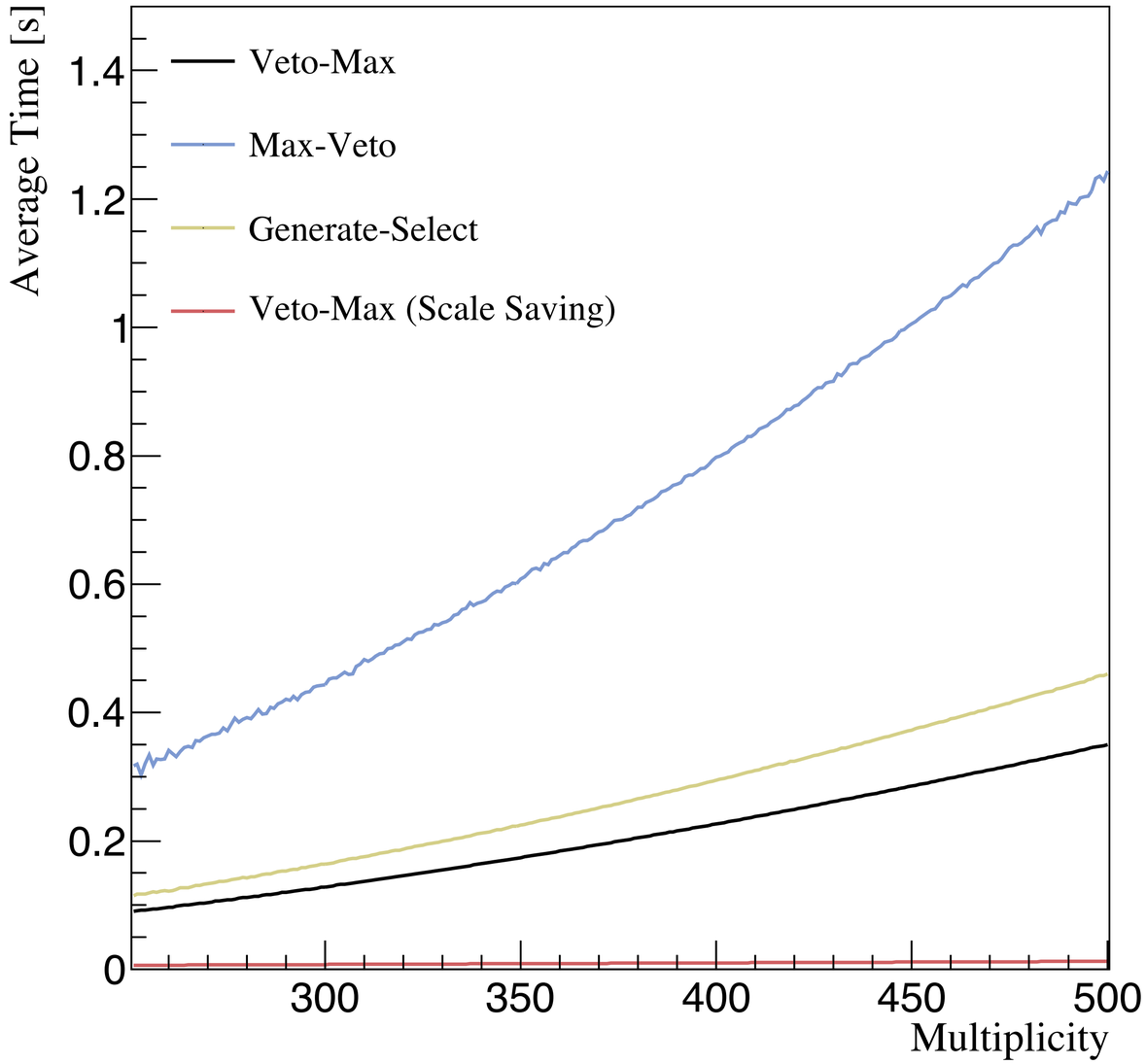
Constant

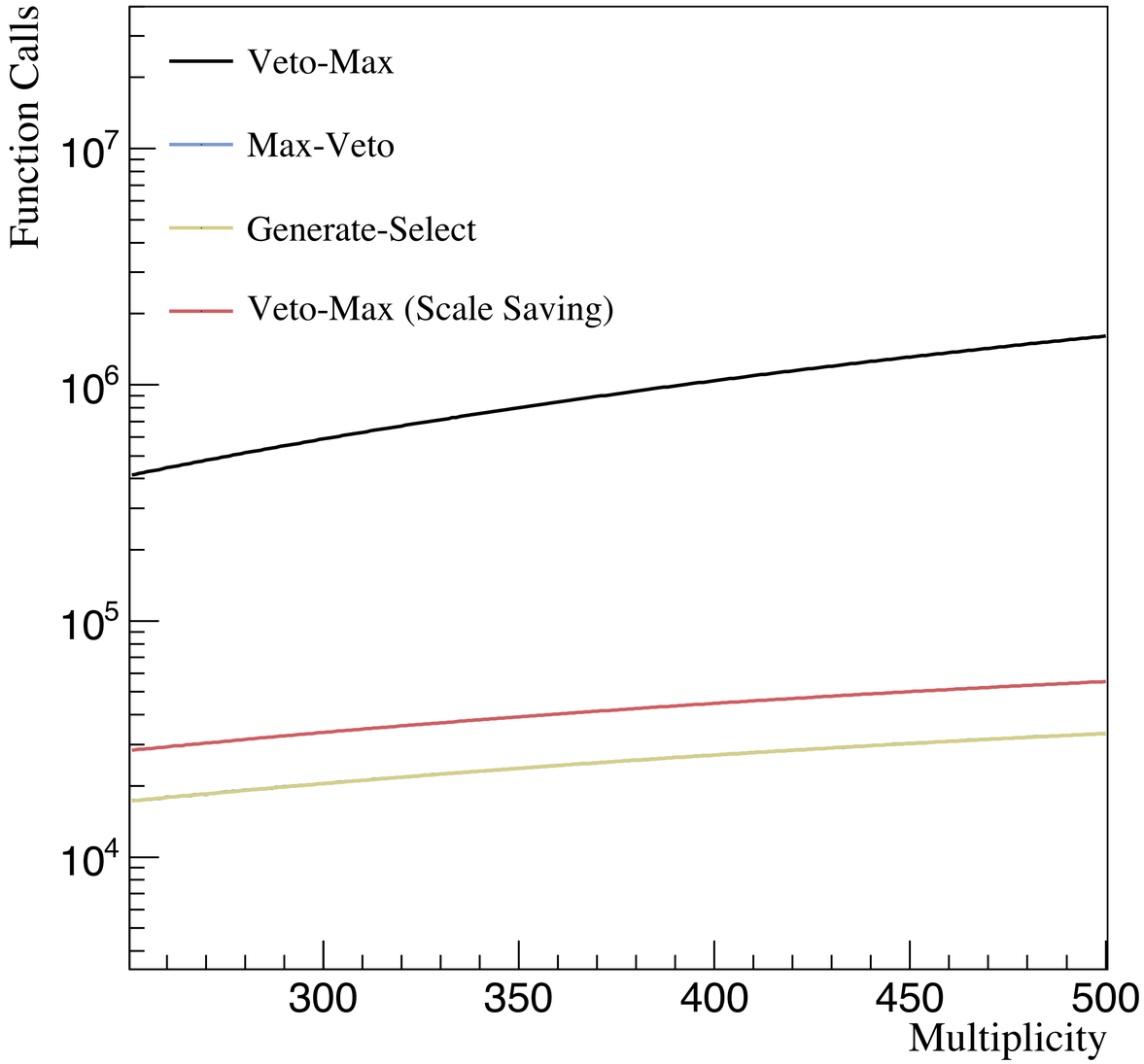




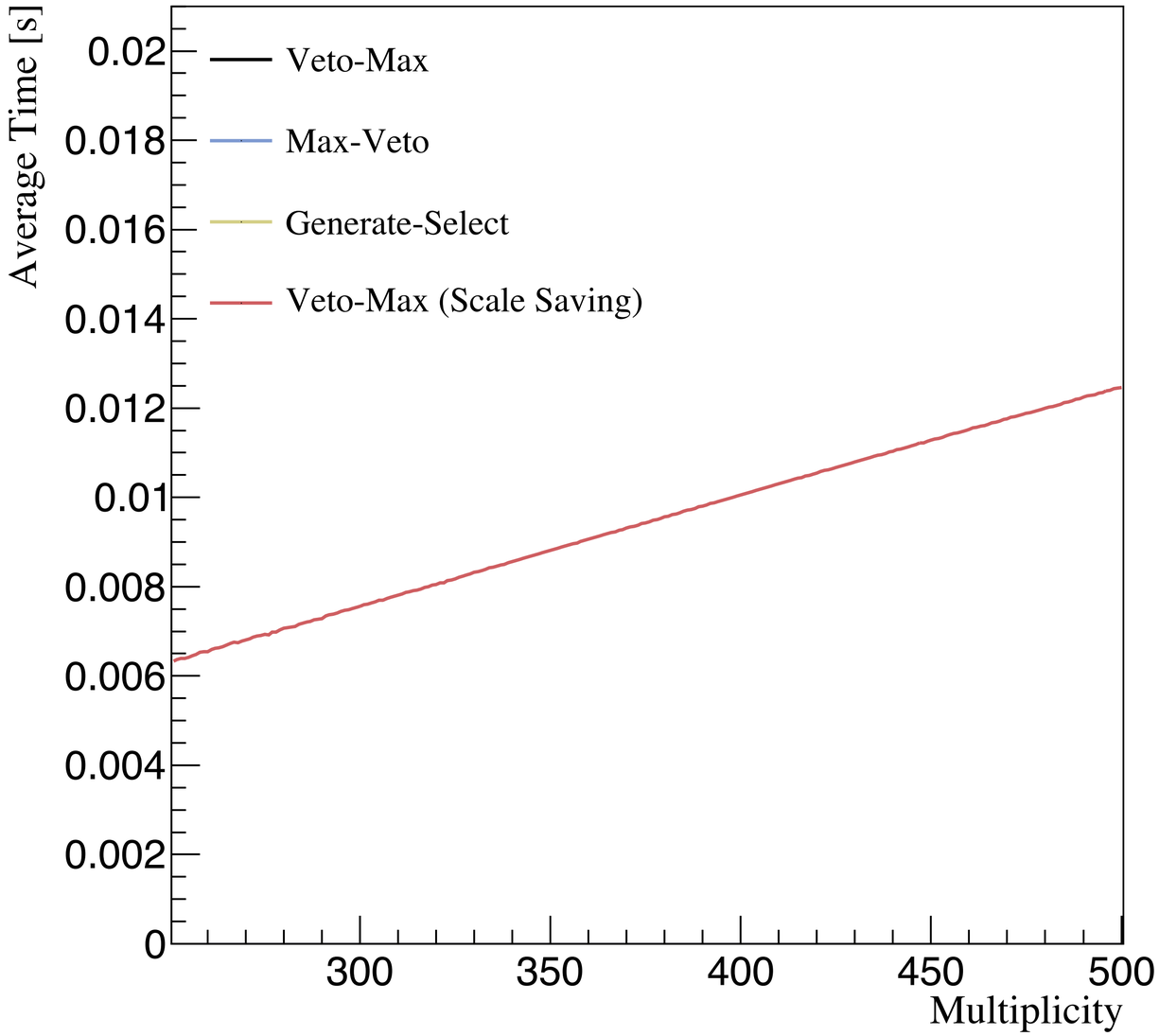
Scales can even be saved after emissions are accepted, as long as the antenna is not kinematically affected

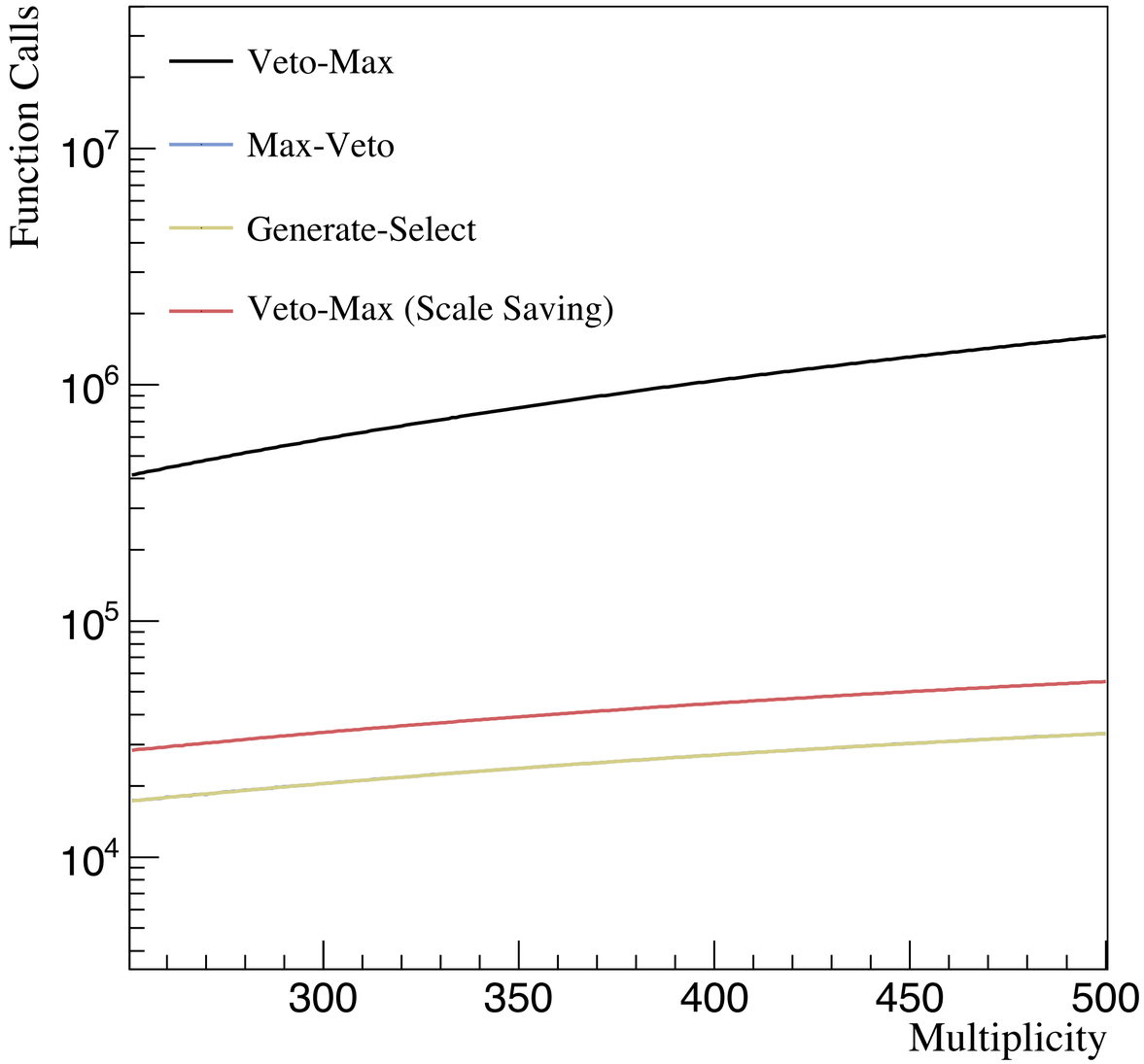
Timings


Average Time [s]
Veto-Max
Max-Veto
Generate-Select
Veto-Max (Scale Saving)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
300
350
400
450
500
Multiplicity

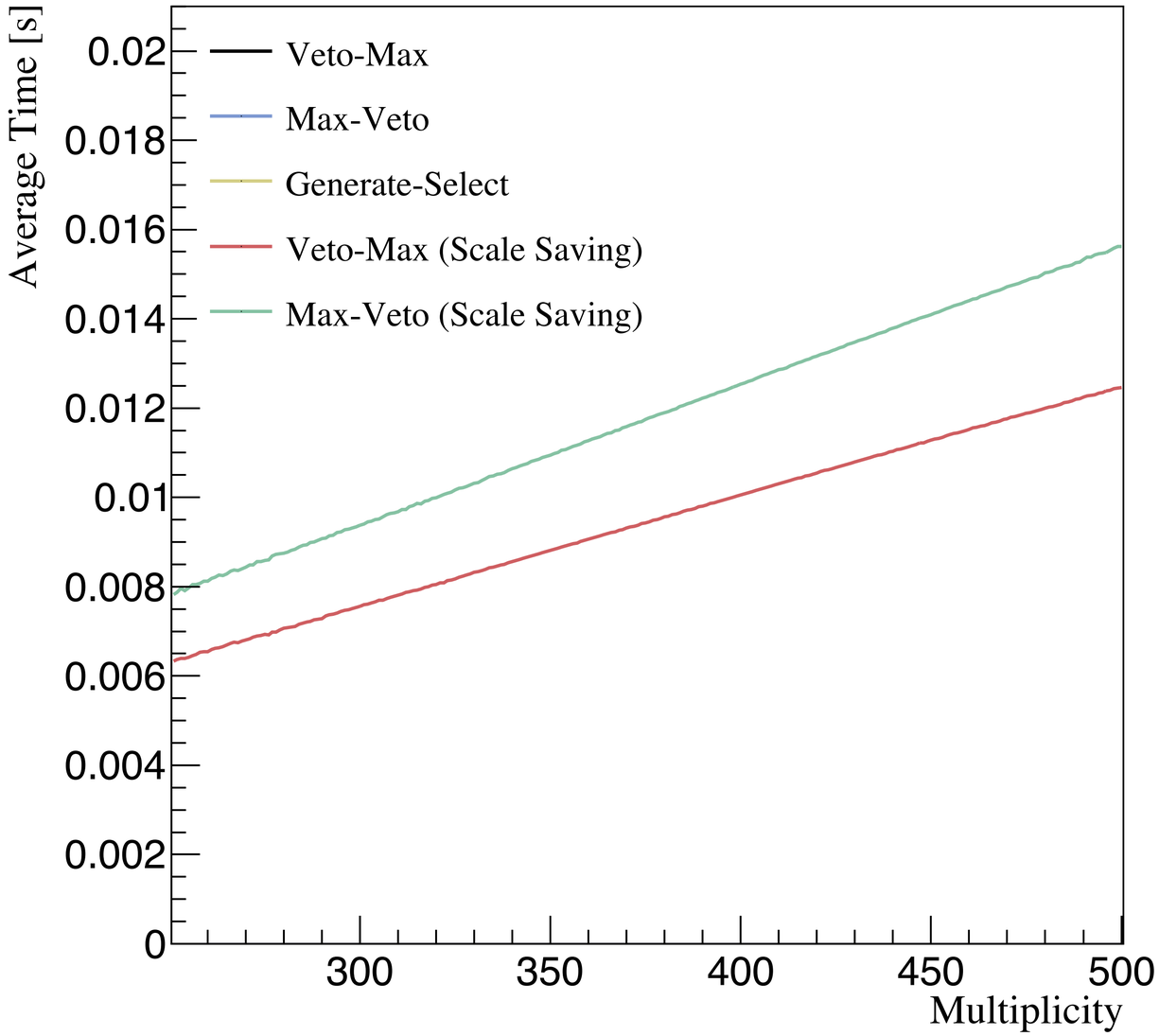

Function Calls
Veto-Max
Max-Veto
Generate-Select
Veto-Max (Scale Saving)
10^7
10^6
10^5
10^4
300
350
400
450
500
Multiplicity

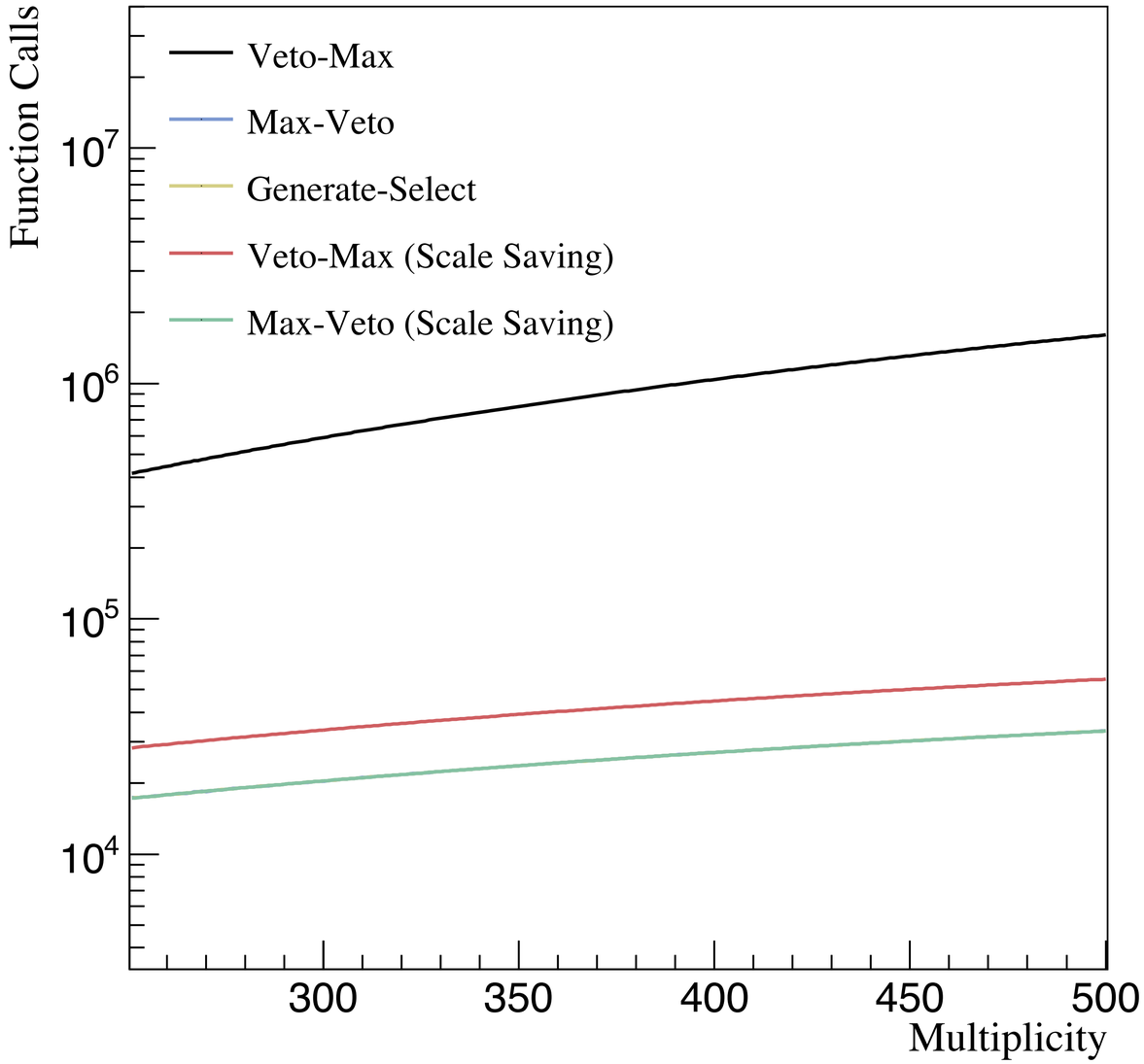
Timings


Average Time [s]
0.02
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
Veto-Max
Max-Veto
Generate-Select
Veto-Max (Scale Saving)
300
350
400
450
500
Multiplicity


Function Calls
10^7
10^6
10^5
10^4
Veto-Max
Max-Veto
Generate-Select
Veto-Max (Scale Saving)
300
350
400
450
500
Multiplicity

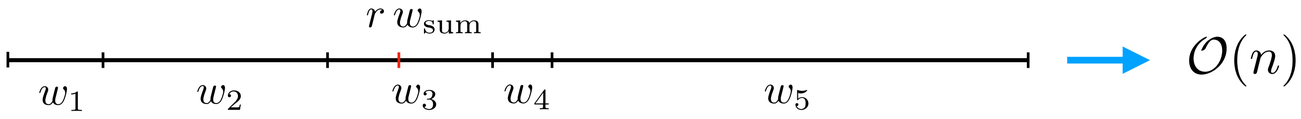
Timings


Average Time [s]
0.02
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
300
350
400
450
500
Multiplicity
Veto-Max
Max-Veto
Generate-Select
Veto-Max (Scale Saving)
Max-Veto (Scale Saving)

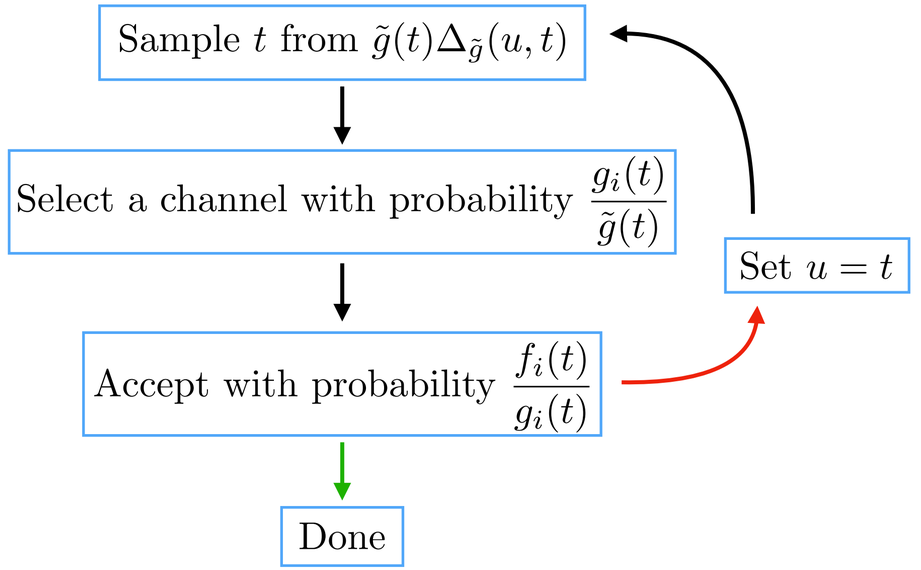

Function Calls
10^7
10^6
10^5
10^4
300
350
400
450
500
Multiplicity
Veto-Max
Max-Veto
Generate-Select
Veto-Max (Scale Saving)
Max-Veto (Scale Saving)

Roulette Wheel Selection (Generate-Select)

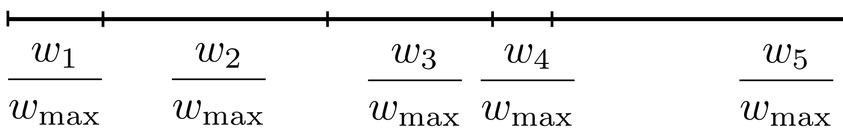
Linear search: Count up to selected element



$\rightarrow \mathcal{O}(n)$



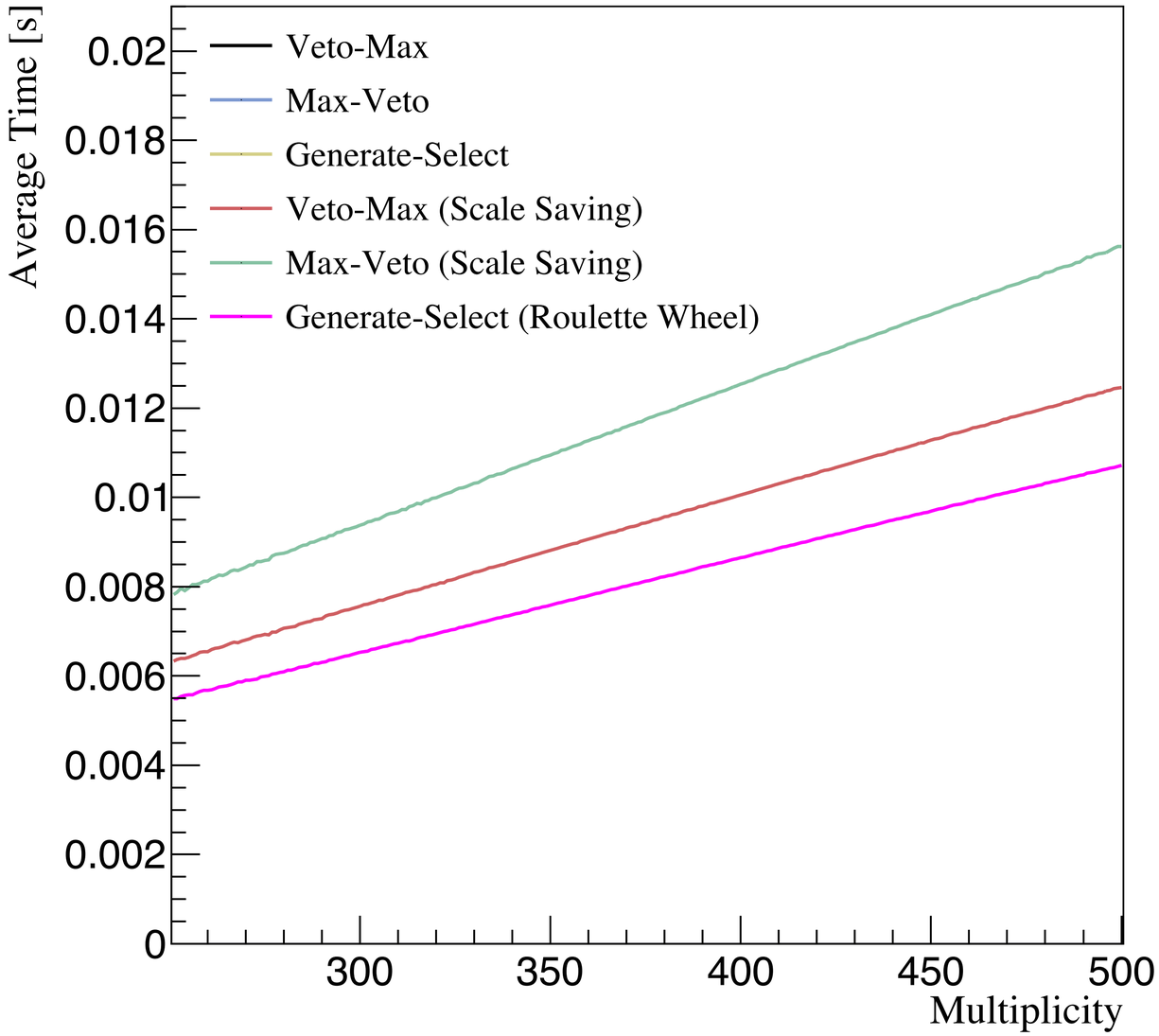
Roulette wheel selection: Select uniformly, accept with proportional probability

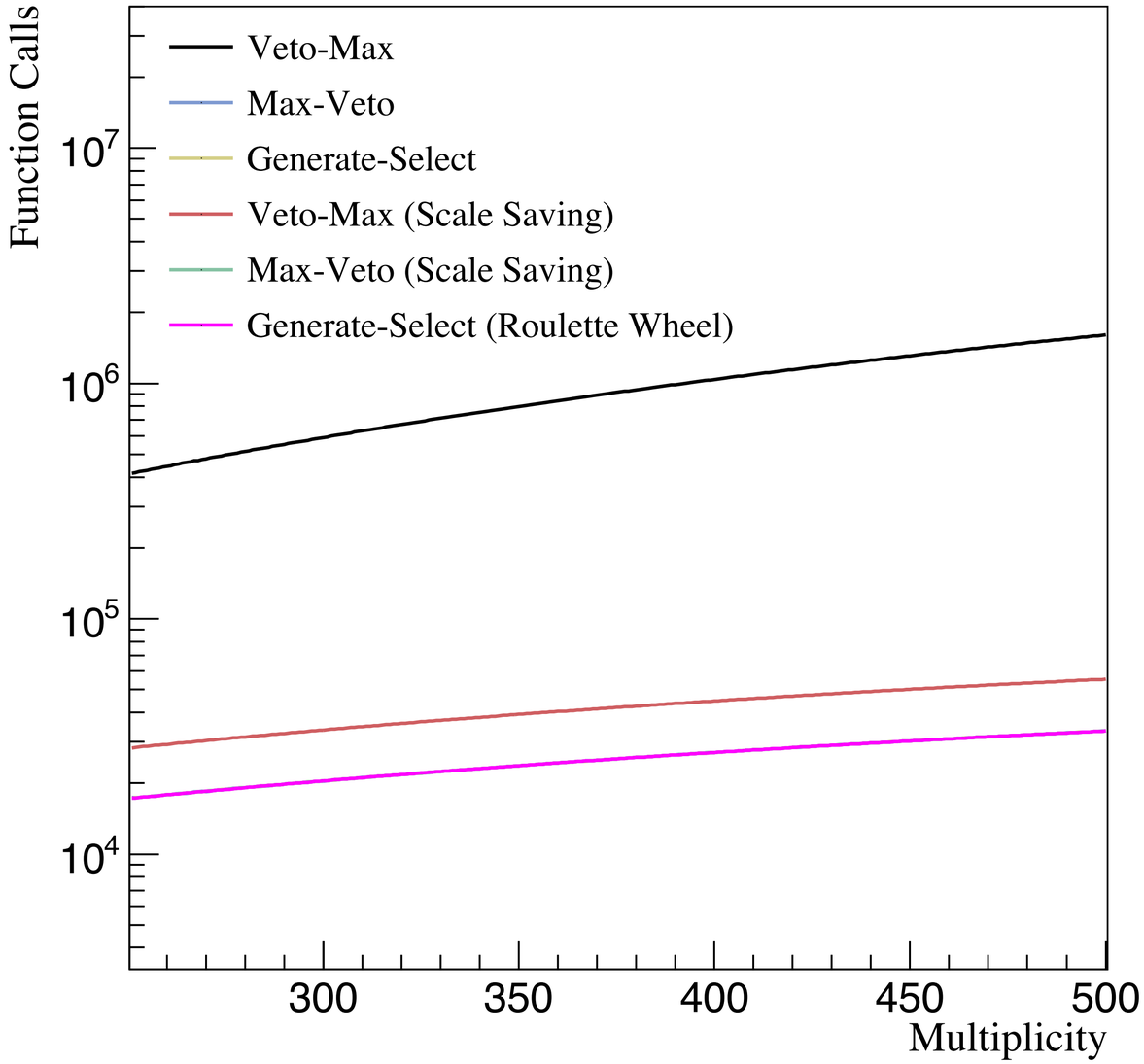


$\rightarrow \mathcal{O}(1)$

Lipowski, Lipowska arXiv:1109.3627

Timings


Average Time [s]
0.02
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
300
350
400
450
500
Multiplicity
Veto-Max
Max-Veto
Generate-Select
Veto-Max (Scale Saving)
Max-Veto (Scale Saving)
Generate-Select (Roulette Wheel)


Function Calls
10^7
10^6
10^5
10^4
300
350
400
450
500
Multiplicity
Veto-Max
Max-Veto
Generate-Select
Veto-Max (Scale Saving)
Max-Veto (Scale Saving)
Generate-Select (Roulette Wheel)

Examples from Vincia: Photon Emissions

No leading colour approximation in QED

→ Photon emission kernel is a sum over many terms, some negative

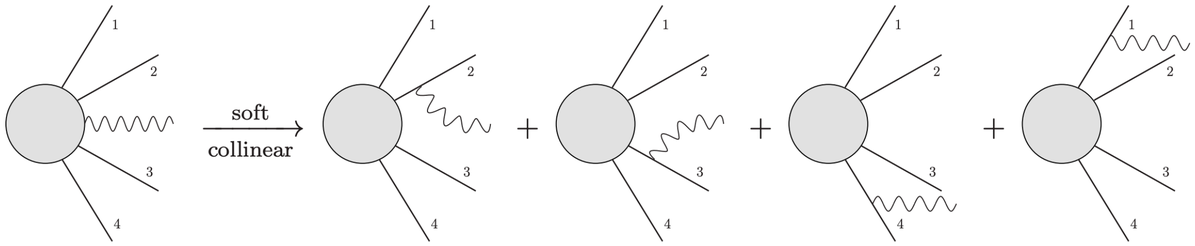
$$f(\{p\}, p_{\text{phot}}) = \sum_i \sum_{j<i} Q_i Q_j f_{ij}(p_i, p_j, p_{\text{phot}})$$

Avoid negative weights:

- Use all of $f(\{p\}, p_{\text{phot}})$ as branching kernel
- Sectorize phase space

$$\bar{f}(\{p\}, p_{\text{phot}}) = \sum_i \sum_{j<i} \Theta(p^2_{\perp,ij}) f(\{p\}, p_{\text{phot}})$$

↳ Pair ij has the smallest transverse momentum

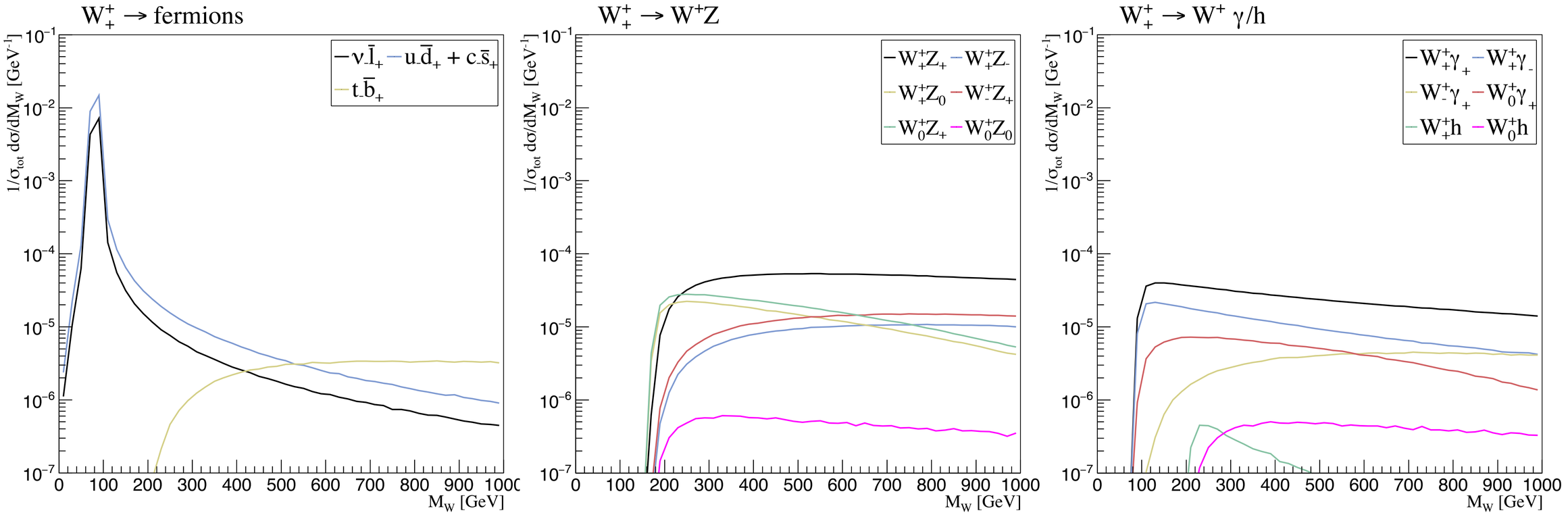


Sectors now compete: Generate and save scales in all sectors

Examples from Vincia: EW Showers

Upcoming in Vincia: Full-fledged EW shower with helicity-dependence

Problem: $\mathcal{O}(1000)$ different branchings with complicated branching kernels



Examples from Vincia: EW Showers

Solution: Write down a parameterised overestimate

$$Q^2 = s_{ij} + m_i^2 + m_j^2 - m_I^2$$

$$g(s_{ij}, s_{jk}) = \frac{1}{Q^2}\left[c_1^{\mathrm{FF}} + c_2^{\mathrm{FF}}\frac{m_{IK}^2}{s_{ij}+s_{ik}+m_i^2} + c_3^{\mathrm{FF}}\frac{m_{IK}^2}{s_{ij}+s_{jk}+m_j^2} + c_4^{\mathrm{FF}}\frac{m_I^2}{Q^2}\right]$$

Solve coefficients numerically: Linear Programming

Use mix of Max-Veto and Generate-Select for sampling:

- Sample a scale for all 4 terms, select the largest
- Select channel with relative probability of its coefficient

Conclusions

- Several ways to handle competition in the Sudakov veto algorithm
- $\mathcal{O}(n \log n)$ (or better) scaling in all cases
- Methods can be combined straightforwardly