Modeling of coasting beams in Xsuite

G. Iadarola

Acknowledgements:
H. Bartosik, R. De Maria, S.Lopaciuk, K. Paraschou

Outline

- **Introduction**
  - Particle slippage
  - Implications for collective effects
- **Simulation method description**
  - Reference speed β_{sim}
  - From turns to frames
  - Algorithm step bu step
- **Mathematical description**
  - Propositions on time time of arrivals
  - Justification of the method
  - Generalization of ζ coordinate
  - ζ update at start turn
  - ζ update on frame jump
- **Numerical tests**

In a ring the particles **revolution frequency depends on the on the particle energy.**

To first order, this dependence is described by the **slip factor**:

$$\Delta f_{\mathrm{rev}} = -\eta f_0 \frac{\Delta P}{P_0}$$

In **bunched beams**, the momentum of the particles **oscillates around the equilibrium momentum**:

- The **average revolution period** (over a synchrotron period) is the **same for all particles**
- The **differences in time of arrival among particles never exceeds one revolution period**
- Over a given time interval, **all particles make the same number of turns**

  → The turn index can be used as independent variable in the simulation

In a ring the particles **revolution frequency depends on the on the particle energy.**

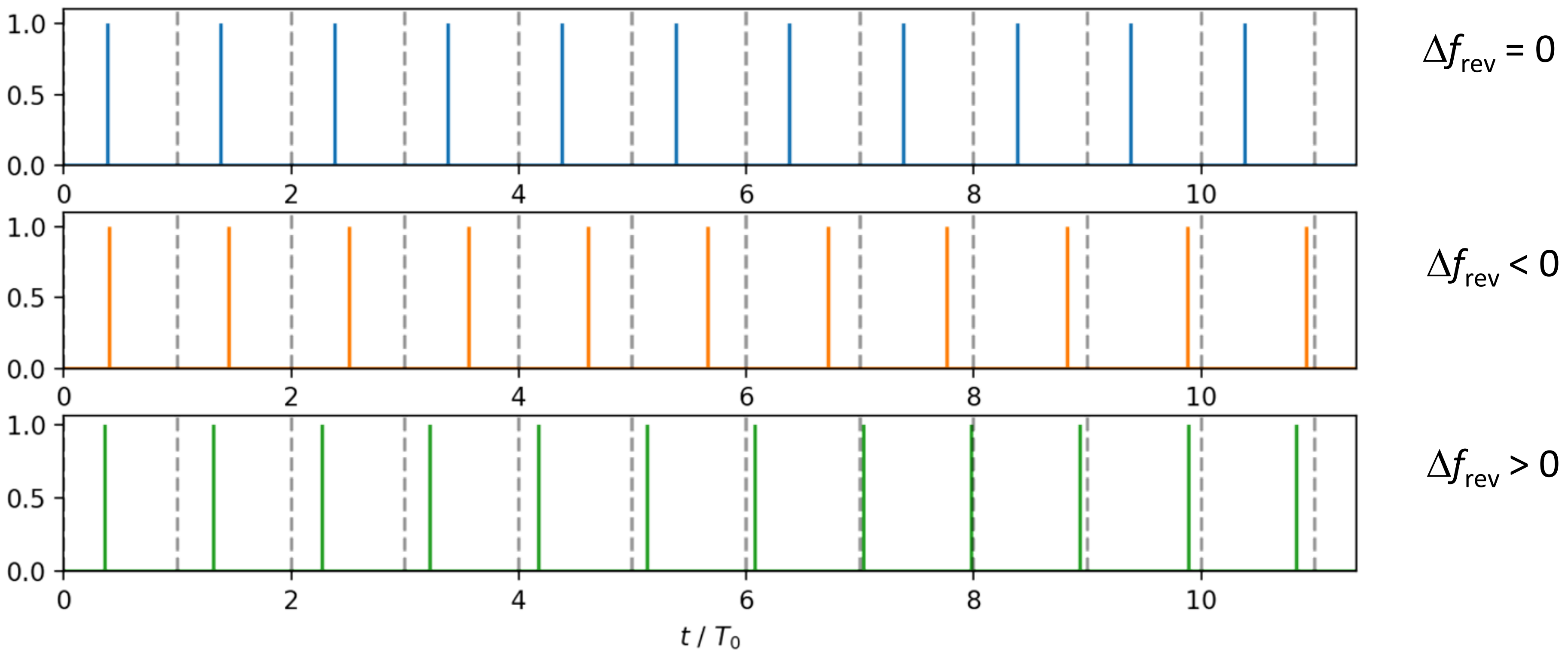
To first order, this dependence is described by the **slip factor**:

$$\Delta f_{\mathrm{rev}} = -\eta f_0 \frac{\Delta P}{P_0}$$

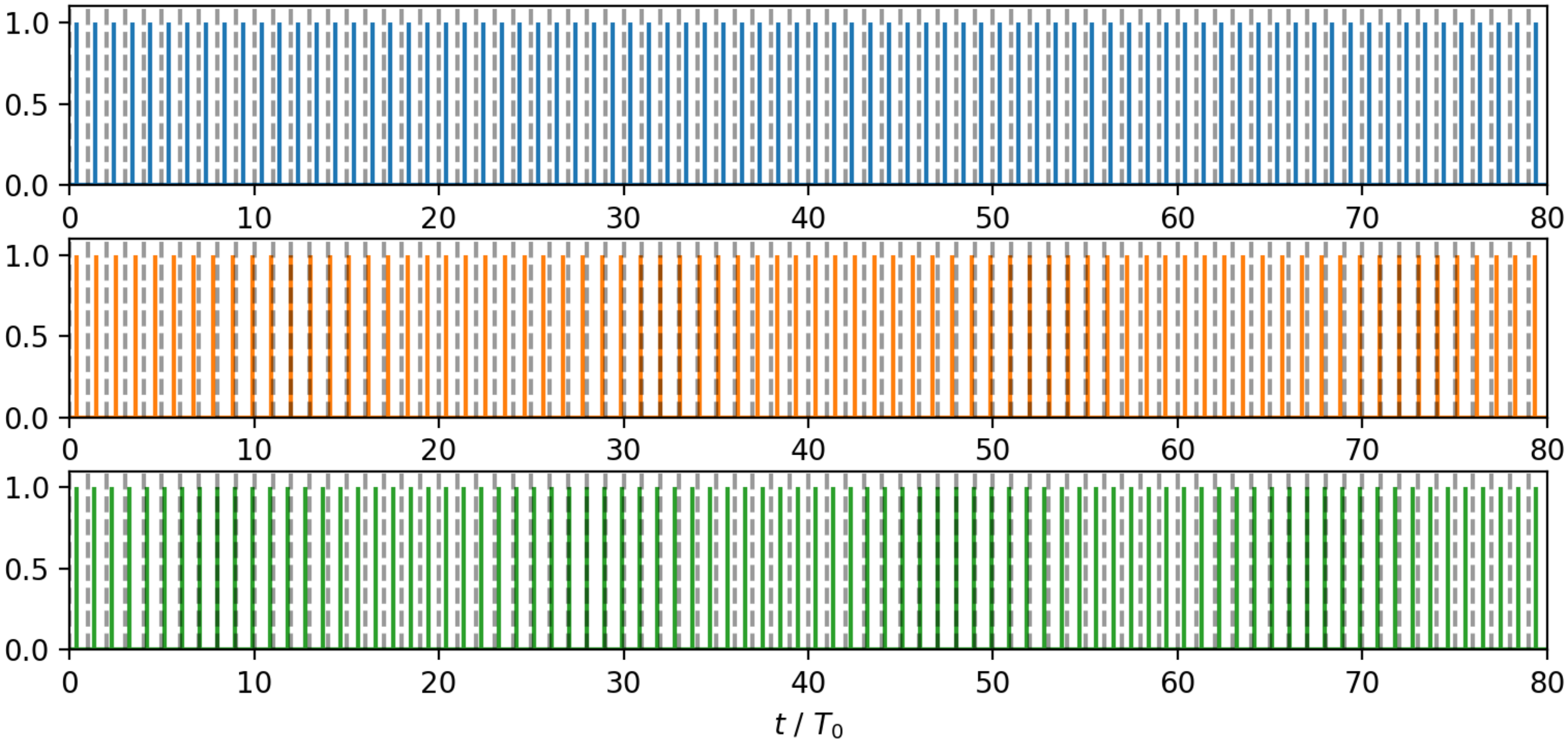
In **coasting beams**, there is no RF focusing:

- Particles **keep their momentum deviation indefinitely**
- Hence, **differences in revolution frequency are also kept indefinitely**
- Over a given time interval, **different particles perform a different number of turns**

- To visualize this effect, in the case of a coasting beam, we **compare three particles having three different revolution frequencies** by **marking the times at which they are detected at a given location of the ring**



- To visualize this effect, in the case of a coasting beam, we **compare three particles having three different revolution frequencies** by **marking the times at which they are detected at a given location of the ring**
  - Over a given time interval, the three particles **perform a different number of revolutions**



$\Delta f_{rev} = 0$

80 revolutions

$\Delta f_{rev} < 0$

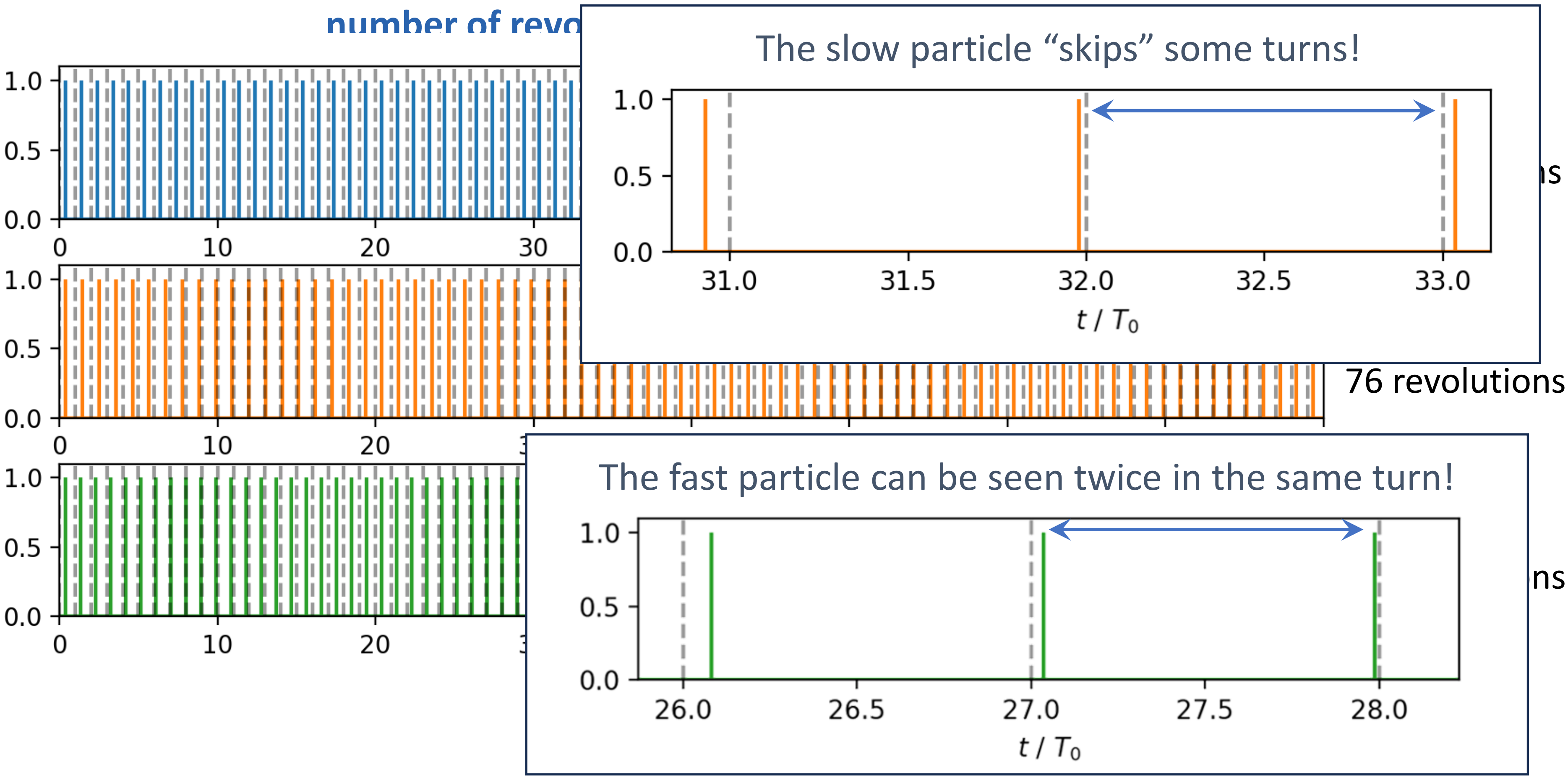
76 revolutions

$\Delta f_{rev} > 0$

84 revolutions

Particle slippage

- To visualize this effect, in the case of a coasting beam, we **compare three particles having three different revolution frequencies** by **marking the times at which they are detected at a given location of the ring**
    - Over a given time interval, the three particles **perform a different number of revo**



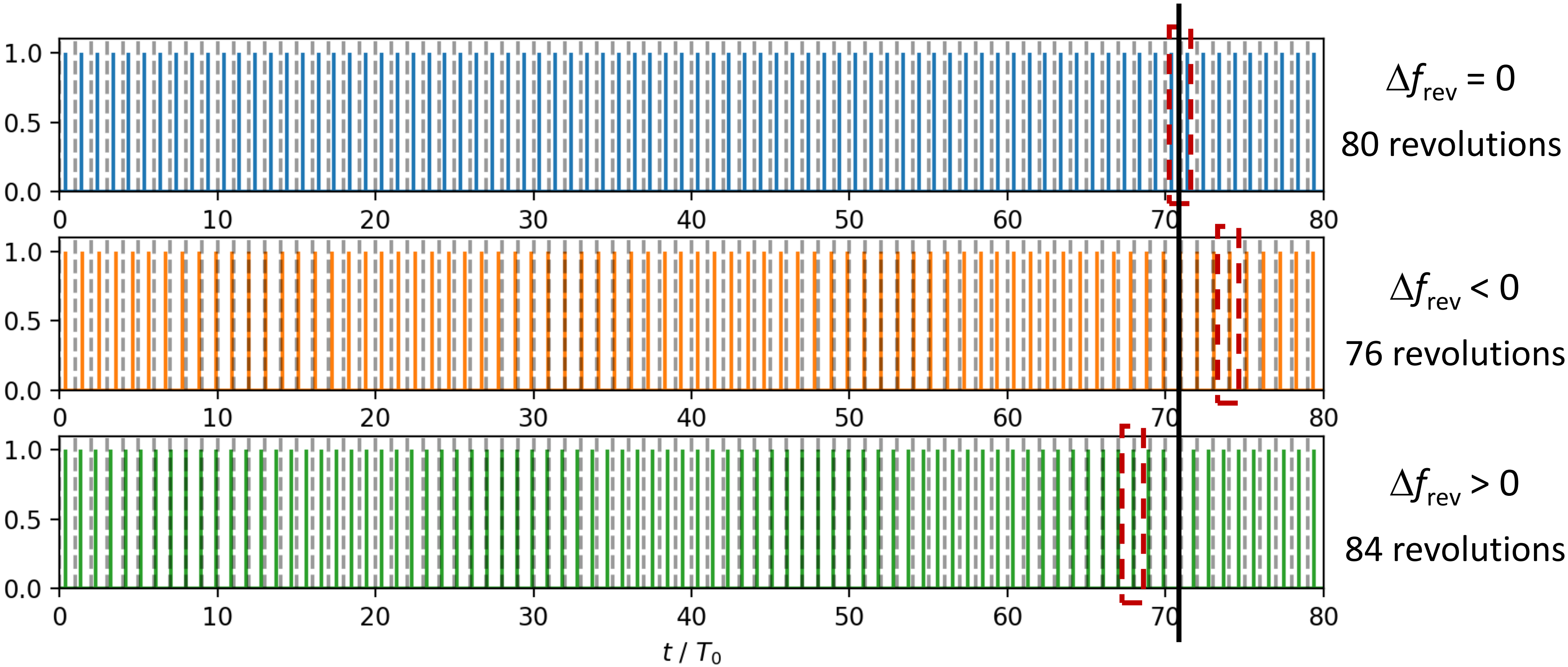
Particle slippage

In **conventional tracking simulations** we use the **longitudinal coordinate *s* and the turn number as independent variables** of our simulations

- At a given simulation stage **all particles have travelled the same distance**, but in the case of coasting beams **their arrival time can be several turns apart**

This creates an **issue for the simulation of collective effects**.

- For example, to compute **space charge** forces on a particle we **need to know the position of all the other particles at the same time**. Same for getting distribution moments in **wakefield simulations**.



Particle slippage

In **conventional tracking simulations** we use the **longitudinal coordinate *s* and the turn number as independent variables** of our simulations

- At a given simulation stage **all particles have travelled the same distance**, but in the case of coasting beams **their arrival time can be several turns apart**

This creates an **issue for the simulation of collective effects**.

- For example, to compute **space charge** forces on a particle we **need to know the position of all the other particles at the same time**. Same for getting distribution moments in **wakefield simulations**.

One **radical solution to this problem** would be to change simulation approach and **use time as independent variable**:

- While conceptually simple, it would **extremely heavy in terms of development effort and very expensive in terms of follow-up and maintenance** (it's basically another code)
- Instead, we have **devised a method** allowing us to **reuse without changes** our conventional simulation modules (tracking elements, space-charge, wakefields, etc.) while **achieving the required synchronization for collective effects**
  - The **implementation** turns out to be quite **simple** and **confined in a dedicated module** (great advantage to preserve Xsuite extendibility and maintainability)
  - The **derivation is a bit cumbersome**, so bear with me…

- Introduction
  - Particle slippage
  - Implications for collective effects
- **Simulation method description**
  - Reference speed β_{sim}
  - From turns to frames
  - Algorithm step bu step
- Mathematical description
  - Propositions on time time of arrivals
  - Justification of the method
  - Generalization of ζ coordinate
  - ζ update at start turn
  - ζ update on frame jump
- Numerical tests

Reference simulation speed

We define for each particles:

$$\hat{\beta}(s_1, s_2) = \frac{1}{c} \frac{s_2 - s_1}{t(s_2) - t(s_1)}$$

Speed measured on the reference trajectory

Then we choose a **"reference speed" β_{sim}** such that for all particles

$$\hat{\beta}(s_1, s_2) < \beta_{sim}$$

i. e. No particle moves faster than β_{sim}

$$\hat{\beta}(s_1, s_2) > \frac{\beta_{sim}}{2}$$

i. e. No particle moves slower β_{sim} / 2

Easy to find: $\beta_{sim} = 1.1\ \beta_0$ works practically for any storage ring

In a nutshell, what we will do in the following is to **use β_{sim} as reference velocity**

→ **Particles can be too slow** (skip a turn) **but not too fast** (arrive twice in a turn)

Note that in the following of this presentation, if we replace β_{sim} with β_0 we get back the definitions and relations used for bunched beams.

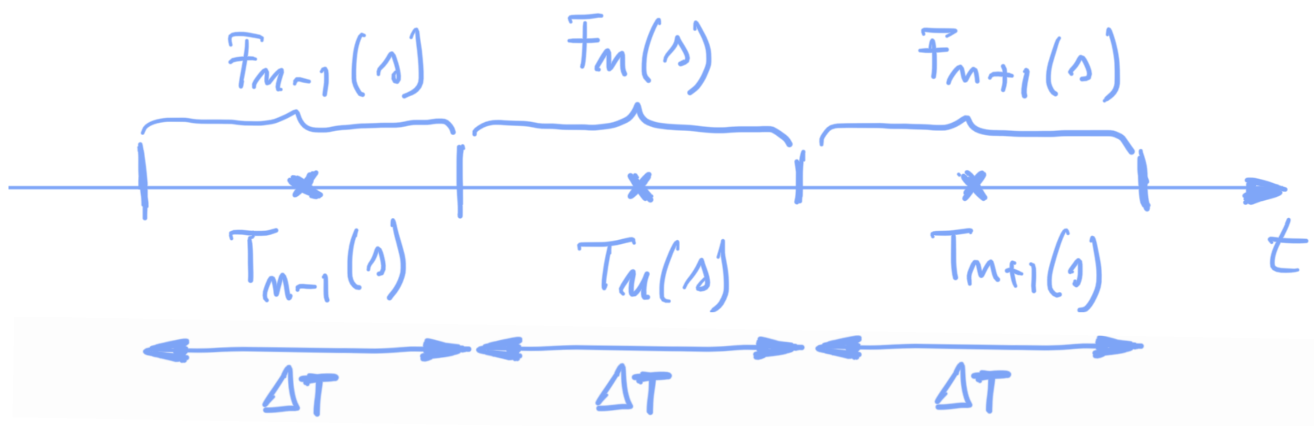
From turns to time frames

We use a “virtual particle” moving at the “reference speed” β_{sim} on the reference trajectory to **generalize the concept of turn**:

- We call **frame** a time interval of length: $$\Delta T = \frac{L}{\beta_{\mathrm{sim}} c}$$

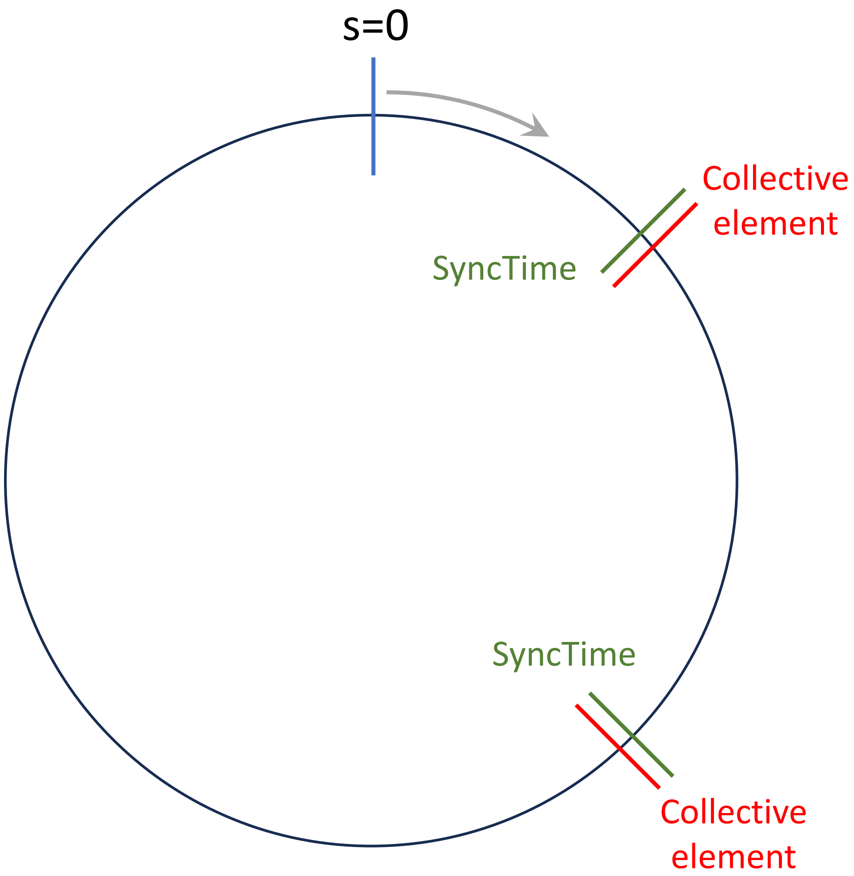
- The **mid point of each frame** depends on the *s* position and is defined by the **passage of the virtual particle at the given** ***s***:

$$T_n(s) = n\Delta T + \frac{s}{\beta_{\mathrm{sim}} c}$$



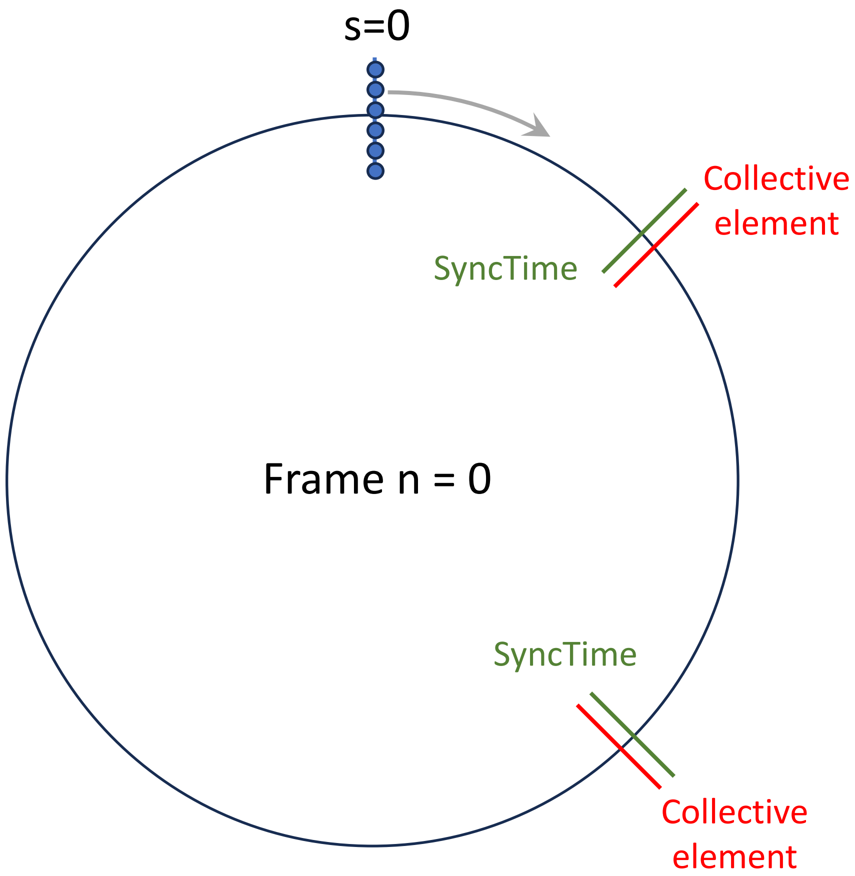
Method

- The **time frames act as the turns** of the bunched beam case:
  - For **each frame we track particles around the ring**
  - We **compute forces due to collective effects only for the present frame**
- We need to ensure that **at each collective interaction we pass to the collective elements all particles arriving during the present time frame** (independently of their number of performed revolutions)
- For this purpose, **we install in front of each collective element, a "SyncTime" element**
  - The SyncTime takes care of **disabling particles that fall out of the current time frame** (arrive too late) and re-enabling them at the following frame



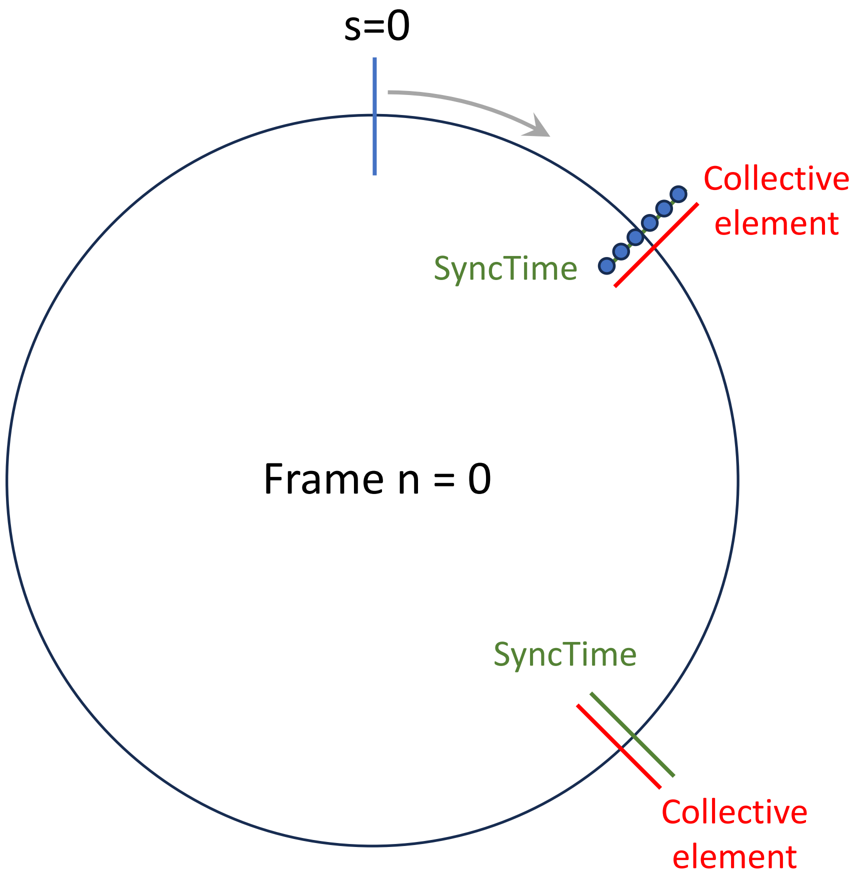
We start by simulating frame $\mathbf{F_0(s)}$:

- We track particles from the start of the ring to the first SyncTime elements



We start by simulating frame $\mathbf{F_0(s)}$:

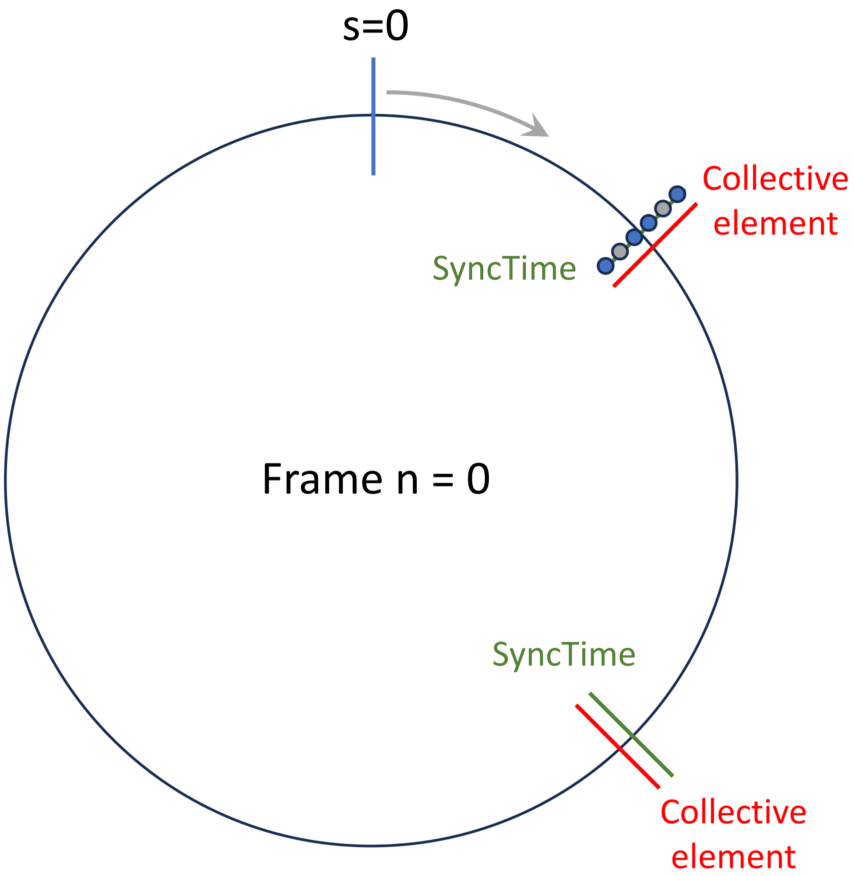
- We track particles from the start of the ring to the first SyncTime elements



We start by simulating frame $\mathbf{F_0(s)}$:

- We track particles from the start of the ring to the first SyncTime elements
- **Some particles are found to arrive too late**, outside $F_0(s)$

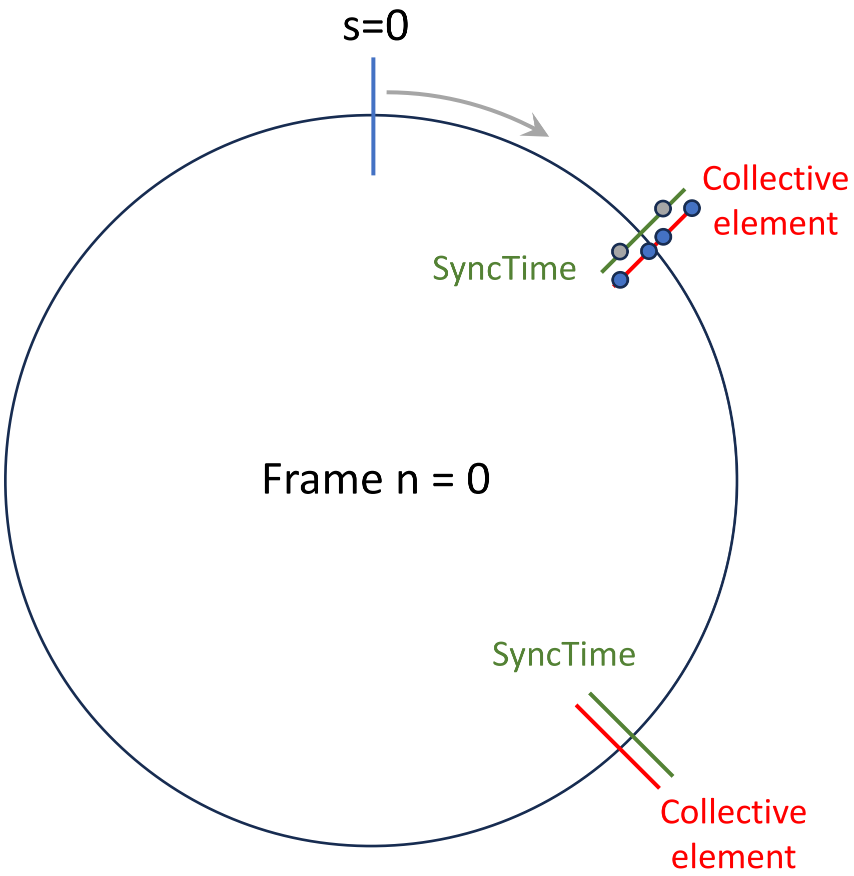
  → these particles are **deactivated**



We start by simulating frame $F_0(s)$:

- We track particles from the start of the ring to the first SyncTime elements
- **Some particles are found to arrive too late**, outside $F_0(s)$

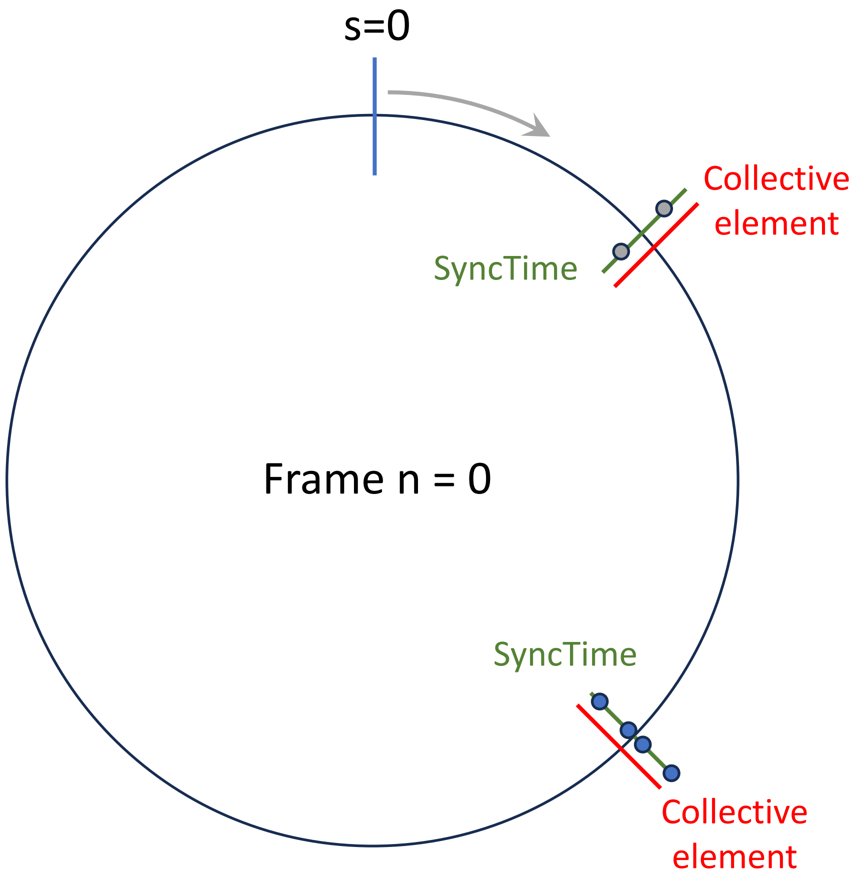
  → these particles are **deactivated**
- The **active particles** are taken into account by the **collective element**



We start by simulating frame $\mathbf{F_0(s)}$:

- We track particles from the start of the ring to the first SyncTime elements
- **Some particles are found to arrive too late**, outside $F_0(s)$

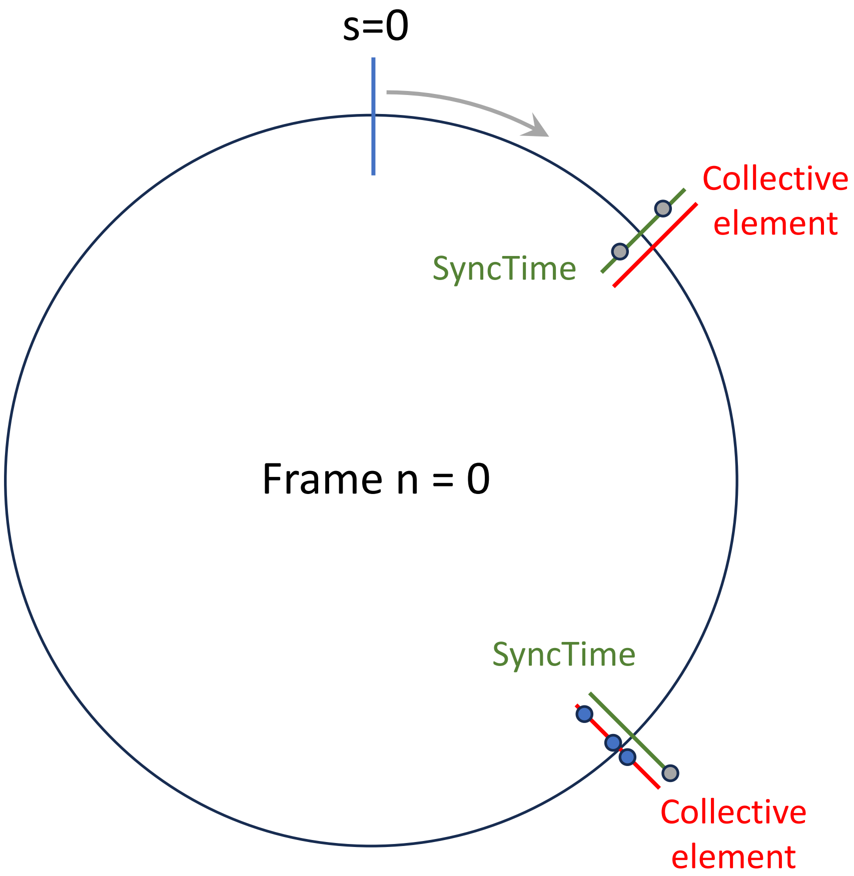
  → these particles are **deactivated**
- The **active particles** are taken into account by the **collective element**
- The active particles are tracked to the **next collective location** where the same procedure takes place



We start by simulating frame $\mathbf{F_0(s)}$:

- We track particles from the start of the ring to the first SyncTime elements
- **Some particles are found to arrive too late**, outside $F_0(s)$

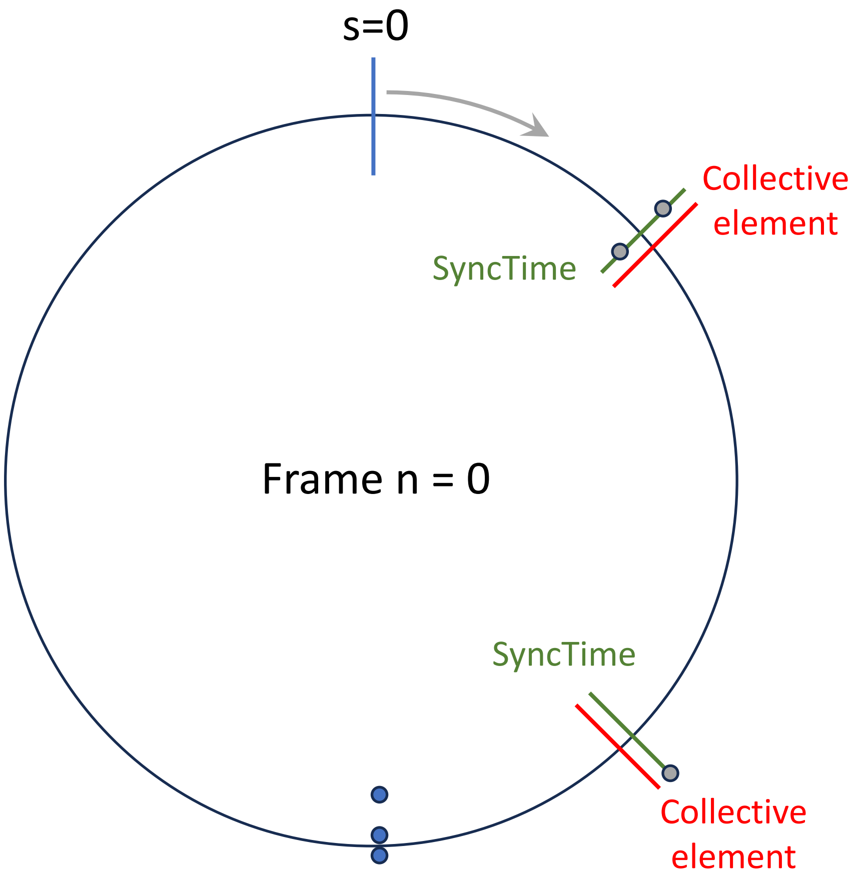
  → these particles are **deactivated**
- The **active particles** are taken into account by the **collective element**
- The active particles are tracked to the **next collective location** where the same procedure takes place



We start by simulating frame $F_0(s)$:

- We track particles from the start of the ring to the first SyncTime elements
- **Some particles are found to arrive too late**, outside $F_0(s)$

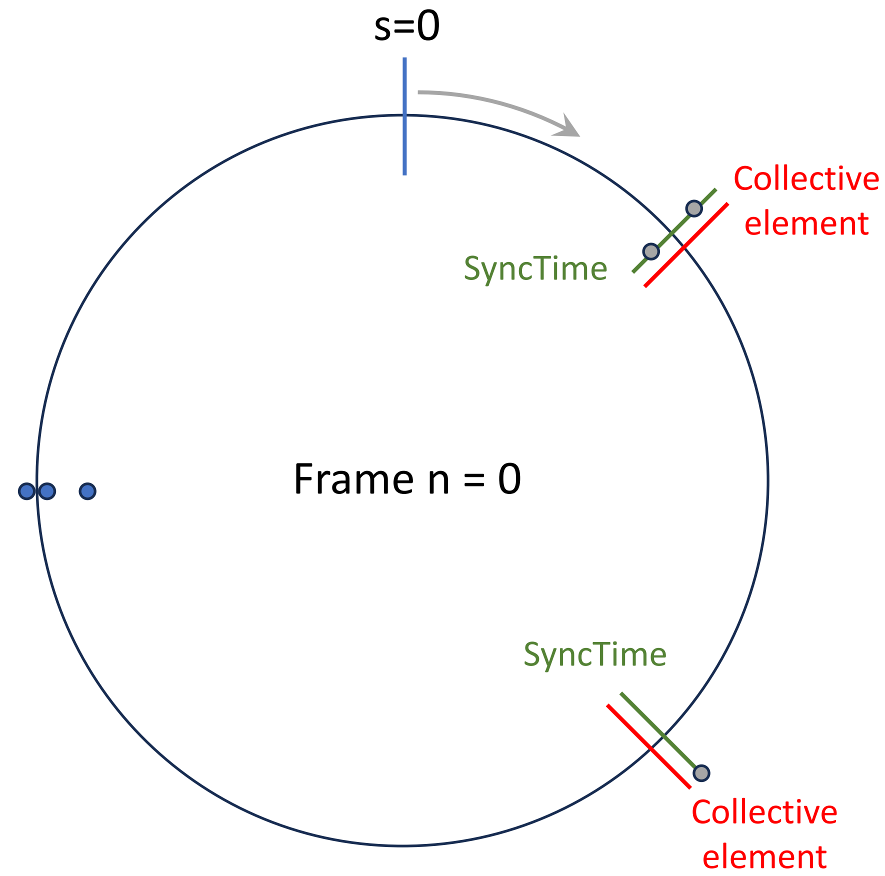
  → these particles are **deactivated**
- The **active particles** are taken into account by the **collective element**
- The active particles are tracked to the **next collective location** where the same procedure takes place
- Tracking continues to the end of the ring



We start by simulating frame $\mathbf{F_0(s)}$:

- We track particles from the start of the ring to the first SyncTime elements
- **Some particles are found to arrive too late**, outside $F_0(s)$

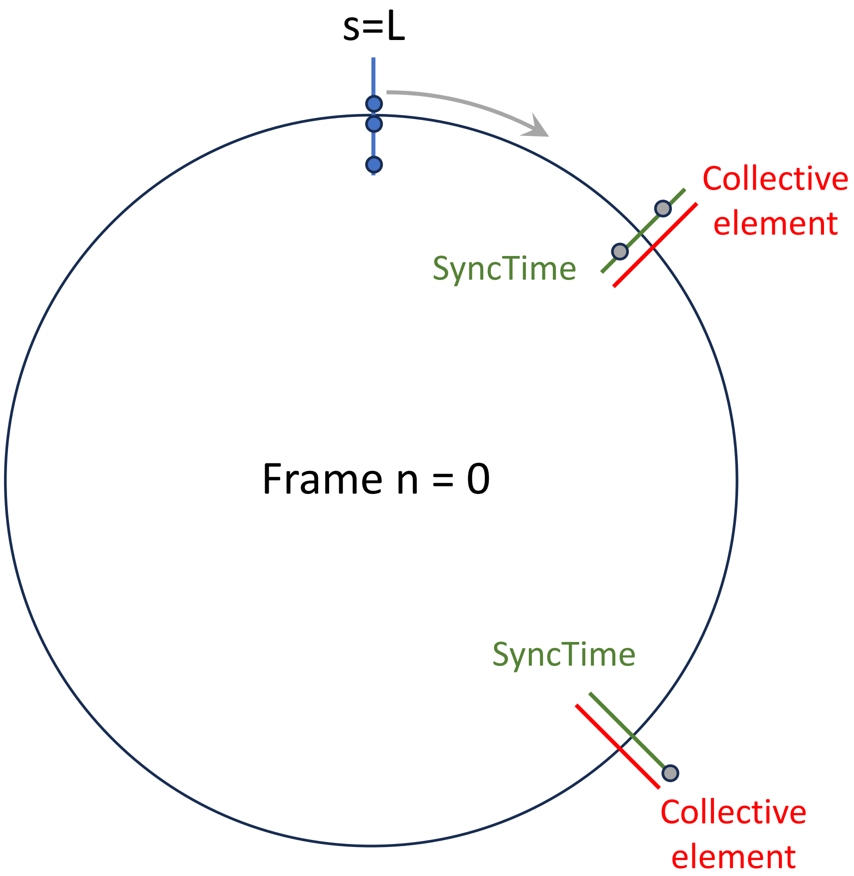
  → these particles are **deactivated**
- The **active particles** are taken into account by the **collective element**
- The active particles are tracked to the **next collective location** where the same procedure takes place
- Tracking continues to the end of the ring



We start by simulating frame **$F_0(s)$**:

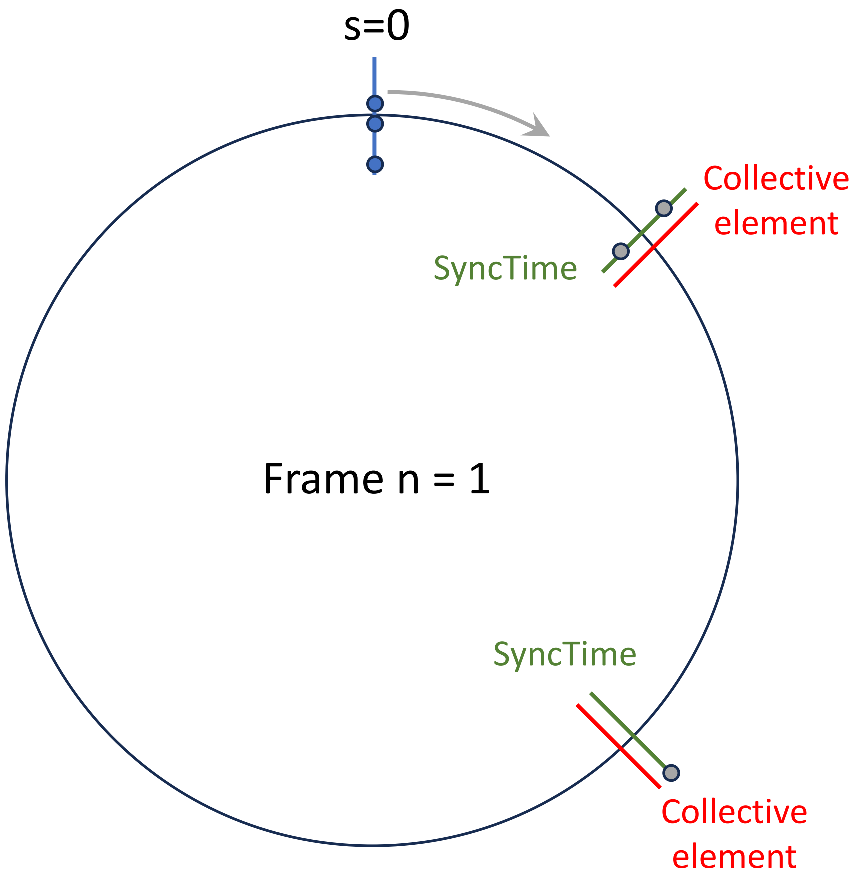
- We track particles from the start of the ring to the first SyncTime elements
- **Some particles are found to arrive too late**, outside $F_0(s)$

  → these particles are **deactivated**
- The **active particles** are taken into account by the **collective element**
- The active particles are tracked to the **next collective location** where the same procedure takes place
- Tracking continues to the end of the ring



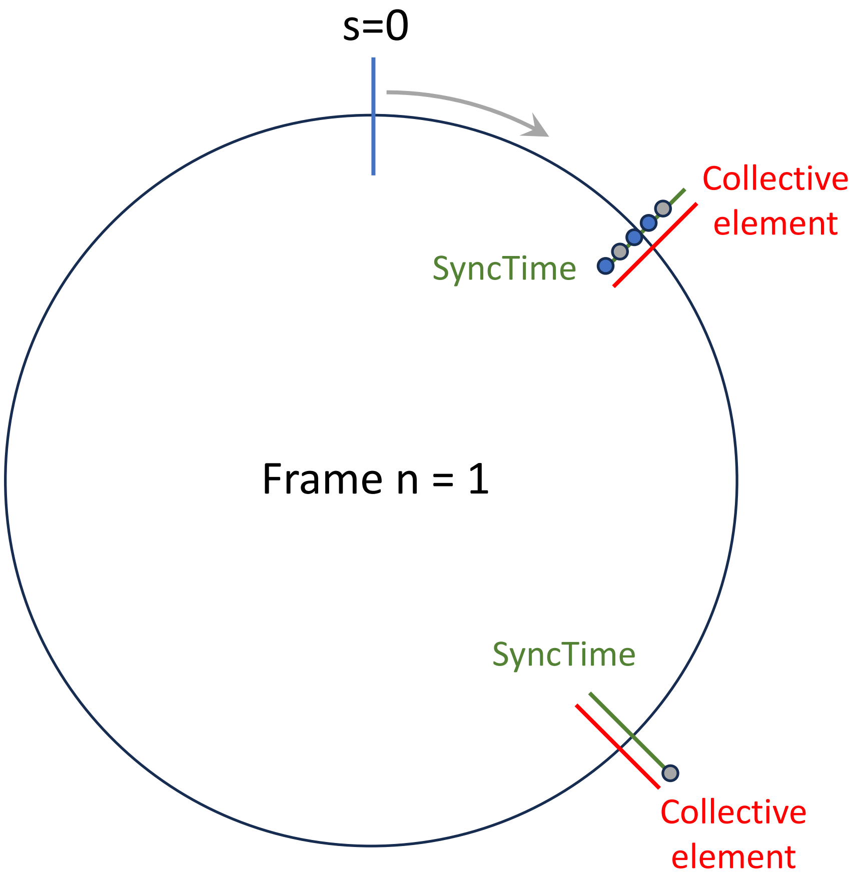
Simuation of frame **F_1(s)**:

- At the start of each turn the **ζ coordinate needs to be updated** (see later)
- We track active partilces to the **first SyncTime element**



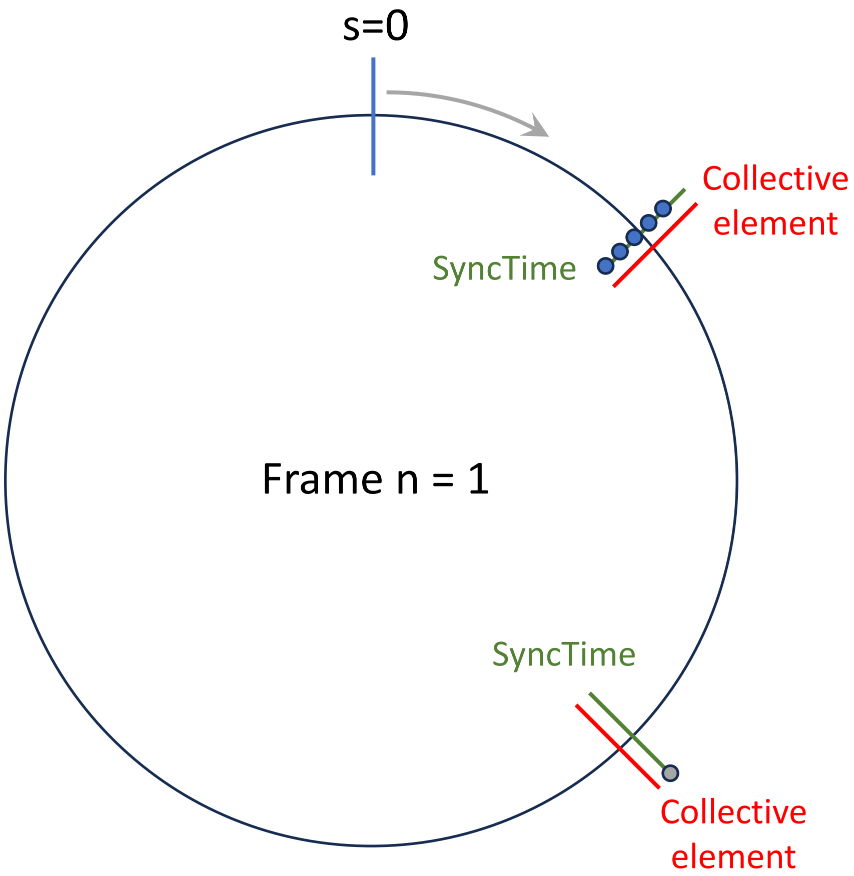
Simuation of frame $\mathbf{F_1(s)}$:

- At the start of each turn the ζ **coordinate needs to be updated** (see later)
- We track active partilces to the **first SyncTime element**



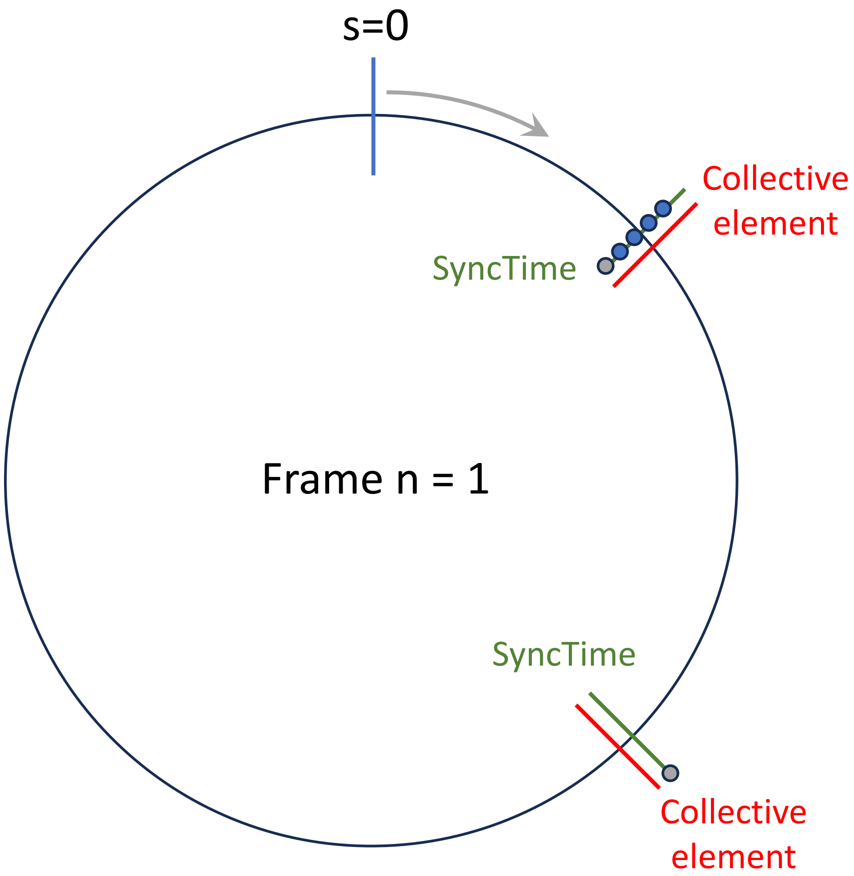
Simuation of frame $\mathbf{F_1(s)}$:

- At the start of each turn the ζ **coordinate needs to be updated** (see later)
- We track active partilces to the **first SyncTime element**
    - Particles that arrived too late in $F_0(s)$ are now **reactivated for $F_1(s)$** → ζ needs to be updated (see later)



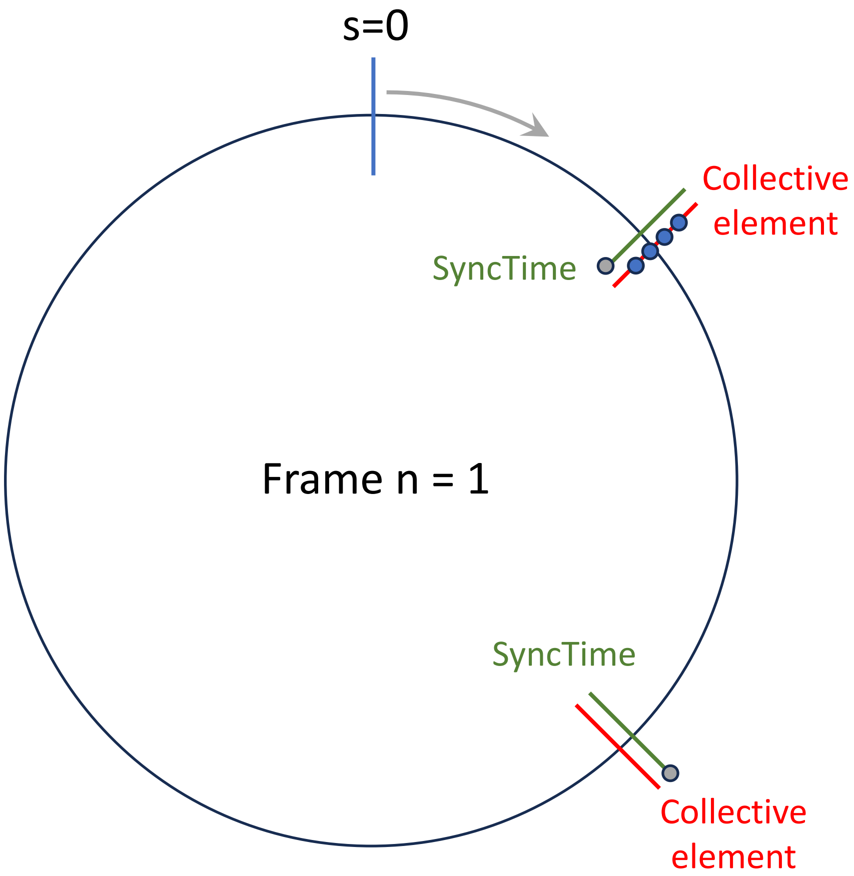
Simuation of frame $\mathbf{F_1(s)}$:

- At the start of each turn the **ζ coordinate needs to be updated** (see later)
- We track active partilces to the **first SyncTime element**
    - Particles that arrived too late in $F_0(s)$ are now **reactivated for $F_1(s)$** → ζ needs to be updated (see later)
    - Particles arriving too late for $F_1(s)$ are deactivated



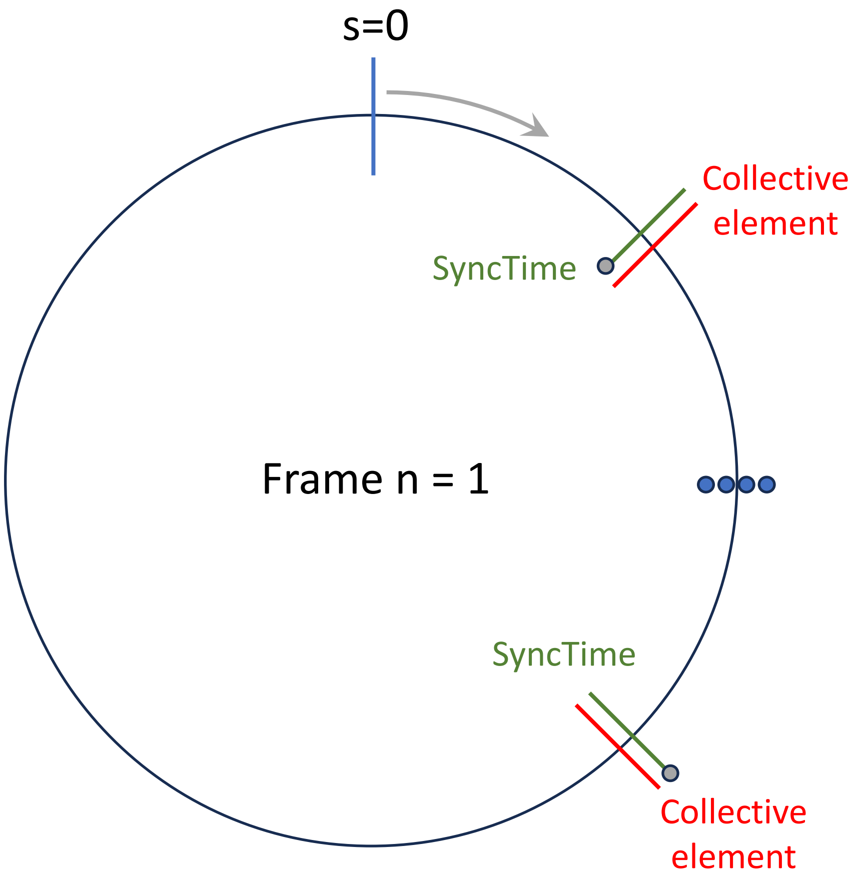
Simuation of frame $\mathbf{F_1(s)}$:

- At the start of each turn the ζ **coordinate needs to be updated** (see later)
- We track active partilces to the **first SyncTime element**
    - Particles that arrived too late in $F_0(s)$ are now **reactivated for $F_1(s)$** → ζ needs to be updated (see later)
    - Particles arriving too late for $F_1(s)$ are deactivated
- And so on…



Simuation of frame $\mathbf{F_1(s)}$:

- At the start of each turn the **ζ coordinate needs to be updated** (see later)
- We track active partilces to the **first SyncTime element**
    - Particles that arrived too late in $F_0(s)$ are now **reactivated for $F_1(s)$** → ζ needs to be updated (see later)
    - Particles arriving too late for $F_1(s)$ are deactivated
- And so on…



- **Introduction**
  - Particle slippage
  - Implications for collective effects
- **Simulation method description**
  - Reference speed β_{sim}
  - From turns to frames
  - Algorithm step bu step
- **Mathematical description**
  - Propositions on time time of arrivals
  - Justification of the method
  - Generalization of ζ coordinate
  - ζ update at start turn
  - ζ update on frame jump
- **Numerical tests**

Propositions on particles arrival times

The procedure illustrated so far can be **rigorously justified mathematically**. Full derivation available in the [Xsuite Physics Guide]().

A central role is played by the following **two propositions**:

Proposition 1: If the time $t_k(s_1)$ defining the k-th arrival of a particle at location s_1 falls in the frame $F_n(s_1)$, then the particle arrives at location $s_2 > s_1$ either in the frame $F_n(s_2)$ or in the following frame $F_{n+1}(s_2)$. In symbols:

$$t_k(s_1) \in F_n(s_1) \Rightarrow t_k(s_2) \in F_n(s_2) \cup F_{n+1}(s_2) \quad \text{for any } s_1 < s_2 \tag{5.10}$$

Proposition 2: If the time $t_k(s_2)$ defining the k-th arrival of a particle at location s_2 falls in the frame $F_n(s_2)$, then the time of $(k+1)$-th arrival at any location $s_1 < s_2$ falls in the frame $F_{n+1}(s_1)$ or in the following frame $F_{n+1}(s_1)$. In symbols:

$$t_k(s_2) \in F_n(s_2) \Rightarrow t_{k+1}(s_1) \in F_{n+1}(s_1) \cup F_{n+2}(s_1) \quad \text{for any } s_1 < s_2 \tag{5.11}$$

In the following I give you a glimpse of the proof…

Proof of proposition 1

Reminders:

$$\hat{\beta}(s_1, s_2) = \frac{1}{c} \frac{s_2 - s_1}{t(s_2) - t(s_1)}$$

$$\hat{\beta}(s_1, s_2) < \beta_{\text{sim}}$$

$$\hat{\beta}(s_1, s_2) > \frac{\beta_{\text{sim}}}{2}$$

$$T_n(s) = n\Delta T + \frac{s}{\beta_{\text{sim}} c}$$

We prove an auxiliary result:

$$\hat{\beta} < \beta_{\text{sim}} \quad \Rightarrow \quad \frac{1}{\hat{\beta}} > \frac{1}{\beta_{\text{sim}}} \quad \Rightarrow \quad \left(\frac{1}{\hat{\beta}} - \frac{1}{\beta_{\text{sim}}}\right) > 0$$

$$\hat{\beta} > \frac{\beta_{\text{sim}}}{2} \quad \Rightarrow \quad \frac{1}{\hat{\beta}} < \frac{2}{\beta_{\text{sim}}} \quad \Rightarrow \quad \left(\frac{1}{\hat{\beta}} - \frac{1}{\beta_{\text{sim}}}\right) < \frac{1}{\beta_{\text{sim}}}$$

Cobining the two:

$$0 < \left(\frac{1}{\hat{\beta}} - \frac{1}{\beta_{\text{sim}}}\right) < \frac{1}{\beta_{\text{sim}}}$$

Another relation that we will use:

$$T_n(s_2) - T_n(s_1) = \frac{s_2 - s_1}{\beta_{\text{sim}} c}$$

Proof of proposition 1

Reminders:

$$\hat{\beta}(s_1, s_2) = \frac{1}{c}\frac{s_2 - s_1}{t(s_2) - t(s_1)} \qquad 0 < \left(\frac{1}{\hat{\beta}} - \frac{1}{\beta_{\text{sim}}}\right) < \frac{1}{\beta_{\text{sim}}} \qquad T_n(s_2) - T_n(s_1) = \frac{s_2 - s_1}{\beta_{\text{sim}} c}$$

We want to prove: $t_k(s_1) \in F_n(s_1) \Rightarrow t_k(s_2) \in F_n(s_2) \cup F_{n+1}(s_2)$ for any $s_1 < s_2$

or equivalently: $T_n(s_2) - \frac{\Delta T}{2} < t_k(s_2) < T_{n+1}(s_2) + \frac{\Delta T}{2}$

$$t_k(s_1) \in F_n(s_1) \Rightarrow t_k(s_1) < T_n(s_1) + \frac{\Delta T}{2}$$

$$t_k(s_2) = t_k(s_1) + \frac{s_2 - s_1}{\hat{\beta} c}$$

$$\Rightarrow t_k(s_2) < T_n(s_1) + \frac{\Delta T}{2} + \frac{s_2 - s_1}{\hat{\beta} c}$$

$$\Rightarrow t_k(s_2) < T_n(s_2) - \frac{s_2 - s_1}{\beta_{\text{sim}} c} - \frac{\Delta T}{2} + \frac{s_2 - s_1}{\hat{\beta} c} \Rightarrow t_k(s_2) < T_n(s_2) + \frac{\Delta T}{2} + \frac{s_2 - s_1}{c}\left(\frac{1}{\hat{\beta}(s)} - \frac{1}{\beta_{\text{sim}}}\right)$$

$$\Rightarrow t_k(s_2) < T_n(s_2) + \frac{\Delta T}{2} + \frac{s_2 - s_1}{\beta_{\text{sim}} c}$$

$$\frac{s_2 - s_1}{\beta_{\text{sim}} c} < \frac{L}{\beta_{\text{sim}} c} = \Delta T$$

$$\Rightarrow t_k(s_2) < T_n(s_2) + \frac{\Delta T}{2} + \Delta T \Rightarrow t_k(s_2) < T_{n+1}(s_2) + \frac{\Delta T}{2}$$

This proves the upper bound.

Proof of proposition 1

Reminders:

$$\hat{\beta}(s_1, s_2) = \frac{1}{c}\frac{s_2 - s_1}{t(s_2) - t(s_1)} \qquad 0 < \left(\frac{1}{\hat{\beta}} - \frac{1}{\beta_{\text{sim}}}\right) < \frac{1}{\beta_{\text{sim}}} \qquad T_n(s_2) - T_n(s_1) = \frac{s_2 - s_1}{\beta_{\text{sim}} c}$$

We want to prove: $t_k(s_1) \in F_n(s_1) \Rightarrow t_k(s_2) \in F_n(s_2) \cup F_{n+1}(s_2) \quad \text{for any } s_1 < s_2$

or equivalently: $T_n(s_2) + \frac{\Delta T}{2} < t_k(s_2) < T_{n+1}(s_2) + \frac{\Delta T}{2}$

$$t_k(s_1) \in F_n(s_1) \Rightarrow t_k(s_1) < T_n(s_1) + \frac{\Delta T}{2}$$

$$t_k(s_2) = t_k(s_1) + \frac{s_2 - s_1}{\hat{\beta} c}$$

$$\Rightarrow t_k(s_2) < T_n(s_1) + \frac{\Delta T}{2} + \frac{s_2 - s_1}{\hat{\beta} c}$$

$$\Rightarrow t_k(s_2) < T_n(s_2) - \frac{s_2 - s_1}{\beta_{\text{sim}} c} - \frac{\Delta T}{2} + \frac{s_2 - s_1}{\hat{\beta} c} \Rightarrow t_k(s_2) < T_n(s_2) + \frac{\Delta T}{2} + \frac{s_2 - s_1}{c}\left(\frac{1}{\hat{\beta}(s)} - \frac{1}{\beta_{\text{sim}}}\right)$$

$$\Rightarrow t_k(s_2) < T_n(s_2) + \frac{\Delta T}{2} + \frac{s_2 - s_1}{\beta_{\text{sim}} c} \Rightarrow t_k(s_2) < T_n(s_2) + \frac{\Delta T}{2} + \Delta T \Rightarrow t_k(s_2) < T_{n+1}(s_2) + \frac{\Delta T}{2}$$

This proves the upper bound.

The proof for the lower bound is very similar and can be found on the [Xsuite Physics Manual]

How these proofs justify the simulation procedure

<u>Prop. 1:</u> $t_k(s_1) \in F_n(s_1) \Rightarrow t_k(s_2) \in F_n(s_2) \cup F_{n+1}(s_2)$

<u>Prop. 2:</u> $t_k(s_2) \in F_n(s_2) \Rightarrow t_{k+1}(s_1) \in F_{n+1}(s_1) \cup F_{n+2}(s_1)$

$$\text{for any } s_1 < s_2$$

- We detect that the particle is **not in the current frame $F_n(s)$ at $s=s_a$**

$$t_k(s_a) \notin F_n(s_a)$$

- We know from **Prop. 1** that the particles is **already in the next frame**

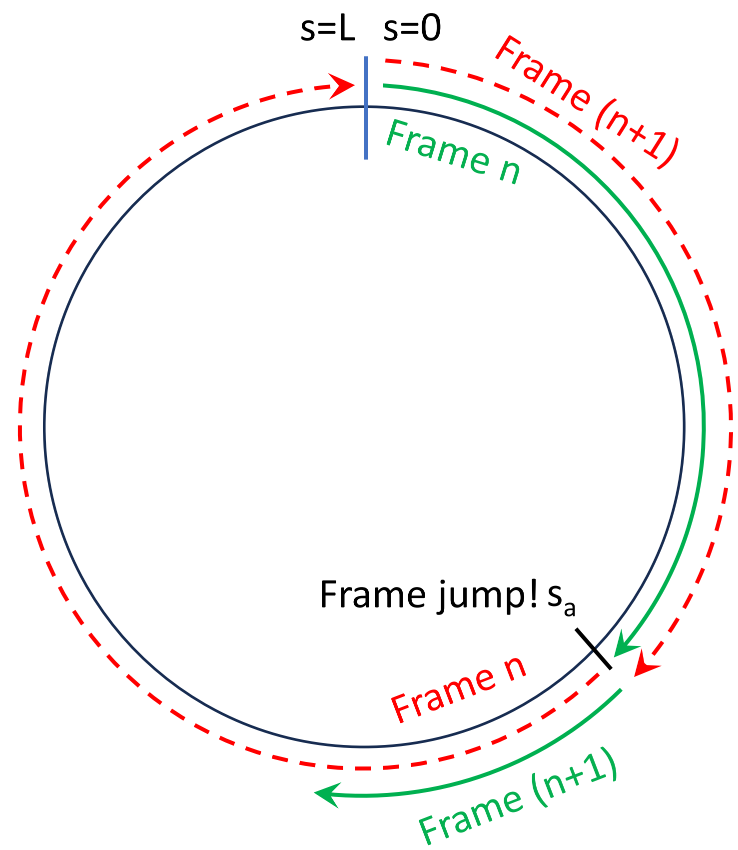
$$t_k(s_a) \in F_{n+1}(s_a)$$

- We know from **Prop. 1** that **for $s > s_a$**:

$$t_k(s_a) \in F_{n+1}(s_a) \Rightarrow t_k(s) \in F_{n+1}(s) \cup F_{n+2}(s)$$
$$\Rightarrow t_k(s) \notin F_n(s)$$

- We know from Prop. 2 for $s < s_a$:

$$t_k(s_a) \in F_{n+1}(s_a) \Rightarrow t_{k+1}(s) \in F_{n+2}(s) \cup F_{n+3}(s)$$
$$\Rightarrow t_{k+1}(s) \notin F_{n+1}(s)$$



This is exacty what we do in the simulator!

- **Introduction**
  - Particle slippage
  - Implications for collective effects
- **Simulation method description**
  - Reference speed β_{sim}
  - From turns to frames
  - Algorithm step bu step
- **Mathematical description**
  - Propositions on time time of arrivals
  - Justification of the method
  - Generalization of ζ coordinate
  - ζ update at start turn
  - ζ update on frame jump
- **Numerical tests**

Generalization of the zeta coordinate

Reminder on usual definitions (bunched beams)

In Xsuite (as in all beam dynamics codes) we don't use t to track the particle arrival time. We instead use **ζ defined as**:

$$\zeta = S - \beta_0 ct$$

where **S is the total traveled length** on the reference trajectory:

$$S = s + nL = s - n\beta_0 T_0$$

Combining the relations above we obtain:

$$\zeta = s - \beta_0 c \underbrace{(t - nT_0)}_{\text{Time within the turn}}$$

Generalization to coasting beams:

For coasting beam we define ζ as:

$$\zeta = s - \beta_0 c \underbrace{(t - n\Delta T)}_{\text{Time within the frame}}$$

Reminder

$$\Delta T = \frac{L}{\beta_{\text{sim}} c}$$

Zeta update at start of turn

Reminder: $$\zeta = s - \beta_0 c(t - n\Delta T) \qquad \Delta T = \frac{L}{\beta_{\mathrm{sim}} c}$$

One of the implications of our definition of ζ is that we **need to update ζ at the start of each turn**:

End of turn n (s=L): $$\zeta^- = L - \beta_0 c(t - n\Delta T)$$

Start of turn n+1 (s=0): $$\zeta^+ = 0 - \beta_0 c(t - (n+1)\Delta T)$$

Subtracting one from the other we obtain the **update equation**:

$$\zeta^+ = \zeta^- - L\left(1 - \frac{\beta_0}{\beta_{\mathrm{sim}}}\right)$$

Conditions on zeta for arrival time in the current frame

We can prove[1] the following condition for the **arrival time of a particle to be in the current frame $F_n(s)$**:

$$t \in F_n(s) \iff -\frac{\Delta\zeta}{2} + \zeta_{\mathrm{corr}}(s) < \zeta < \frac{\Delta\zeta}{2} + \zeta_{\mathrm{corr}}(s)$$

where: $$\Delta\zeta = \beta_0 c \Delta T = \frac{\beta_0}{\beta_{\mathrm{sim}}} L \qquad \zeta_{\mathrm{corr}}(s) = s\left(1 - \frac{\beta_0}{\beta_{\mathrm{sim}}}\right)$$

(1) <u>Proof:</u>

Reminder: $\zeta = s - \beta_0 c(t - n\Delta T)$

$$t \in F_n(s)$$

$$\Updownarrow$$

$$-\frac{\Delta T}{2} < t - T_n(s) < \frac{\Delta T}{2}$$

$$\Updownarrow$$

$$-\frac{\Delta T}{2} < t - n\Delta T - \frac{s}{\beta_{\mathrm{sim}} c} < \frac{\Delta T}{2}$$

$$\Updownarrow$$

$$-\frac{\Delta T}{2} < \left(\frac{s}{\beta_0 c} - \frac{\zeta}{\beta_0 c} + n\Delta T\right) - n\Delta T - \frac{s}{\beta_{\mathrm{sim}} c} < \frac{\Delta T}{2}$$

$$\Updownarrow$$

$$\frac{\Delta T}{2} > -\frac{s}{\beta_0 c} + \frac{\zeta}{\beta_0 c} + \frac{s}{\beta_{\mathrm{sim}} c} > -\frac{\Delta T}{2}$$

$$\Updownarrow$$

$$-\frac{\Delta T}{2} < -\frac{s}{\beta_0 c} + \frac{\zeta}{\beta_0 c} + \frac{s}{\beta_{\mathrm{sim}} c} < \frac{\Delta T}{2}$$

$$\Updownarrow$$

$$-\beta_0 c\frac{\Delta T}{2} < \zeta - s\left(1 - \frac{\beta_0}{\beta_{\mathrm{sim}}}\right) < \beta_0 c\frac{\Delta T}{2}$$

Zeta update on frame jump

Reminder:

$$t \in F_n(s) \Leftrightarrow -\frac{\Delta\zeta}{2} + \zeta_{\mathrm{corr}}(s) < \zeta < \frac{\Delta\zeta}{2} + \zeta_{\mathrm{corr}}(s)$$

where: $\Delta\zeta = \beta_0 c \Delta T = \frac{\beta_0}{\beta_{\mathrm{sim}}} L \qquad \zeta_{\mathrm{corr}}(s) = s\left(1 - \frac{\beta_0}{\beta_{\mathrm{sim}}}\right)$

Particles **arrive out of the current frame** (arrive too late) and jump to the next when:

$$t > T_n(s) + \frac{\Delta T}{2} \Leftrightarrow \zeta < -\frac{\Delta\zeta}{2} + s\left(1 - \frac{\beta_0}{\beta_{\mathrm{sim}}}\right)$$

As discussed, the particle is stopped, and its tracking is resumed when handling frame (n+1)

- When this happens the **ζ coordinates needs to be updated**:

$$\zeta_{\text{before jump}} = s - \beta_0 c\,(t - n\Delta T)$$
$$\zeta_{\text{after jump}} = s - \beta_0 c\,(t - (n+1)\Delta T)$$

Subtracting one from the other we obtain the **update equation**:

$$\boxed{\zeta_{\text{after jump}} = \zeta_{\text{before jump}} + \beta_0 c \Delta T = \zeta_{\text{before jump}} + \Delta\zeta}$$

Outline

- **Introduction**
  - Particle slippage
  - Implications for collective effects
- **Simulation method description**
  - Reference speed β_{sim}
  - From turns to frames
  - Algorithm step bu step
- **Mathematical description**
  - Propositions on time time of arrivals
  - Justification of the method
  - Generalization of ζ coordinate
  - ζ update at start turn
  - ζ update on frame jump
- **Numerical tests**

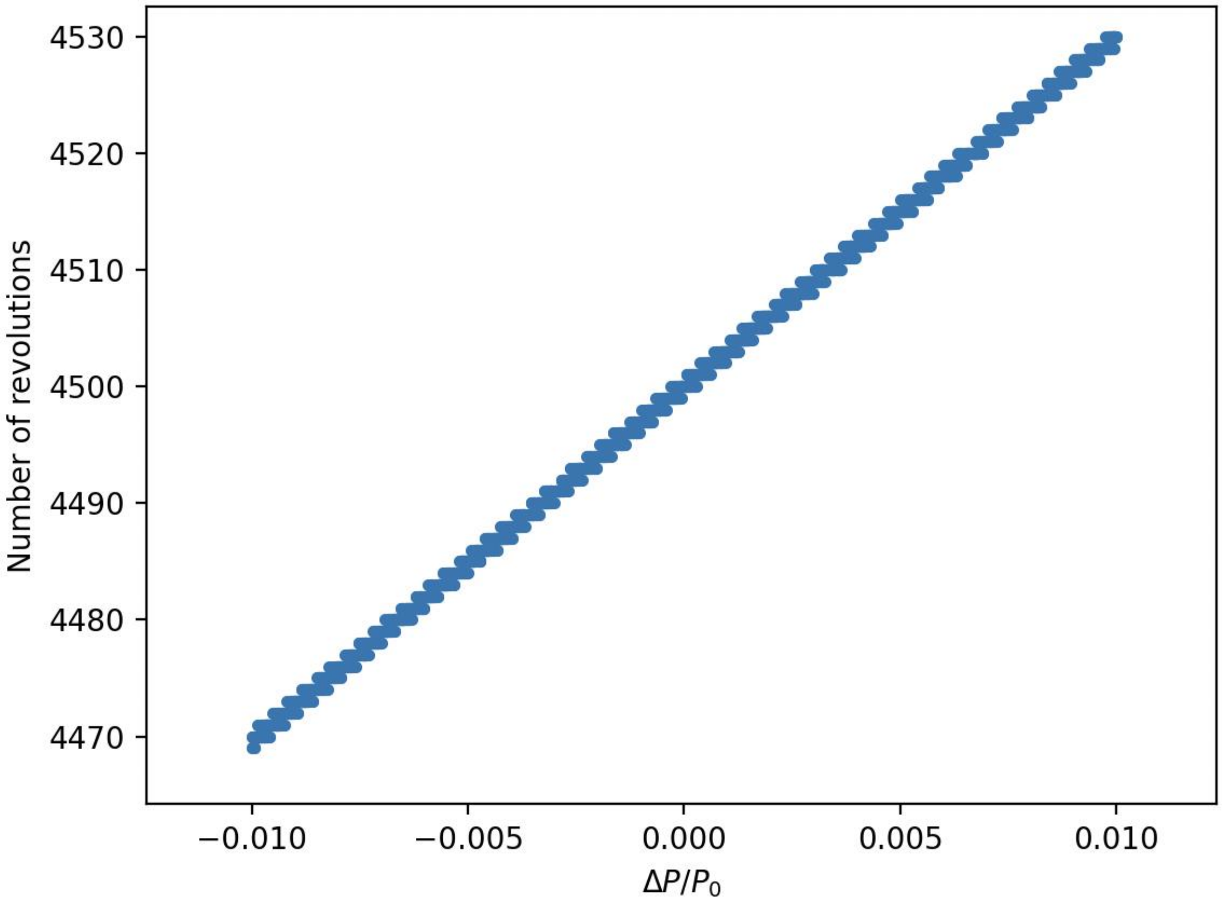
Test simulations

Test case:

CERN PS Booster — E_{kin} = 160 MeV

Full lattice — RF off

Tracking a beam with an artificially **large momentum spread** we can clearly see that different particles perform a **different number of revolution over the simulated time**.

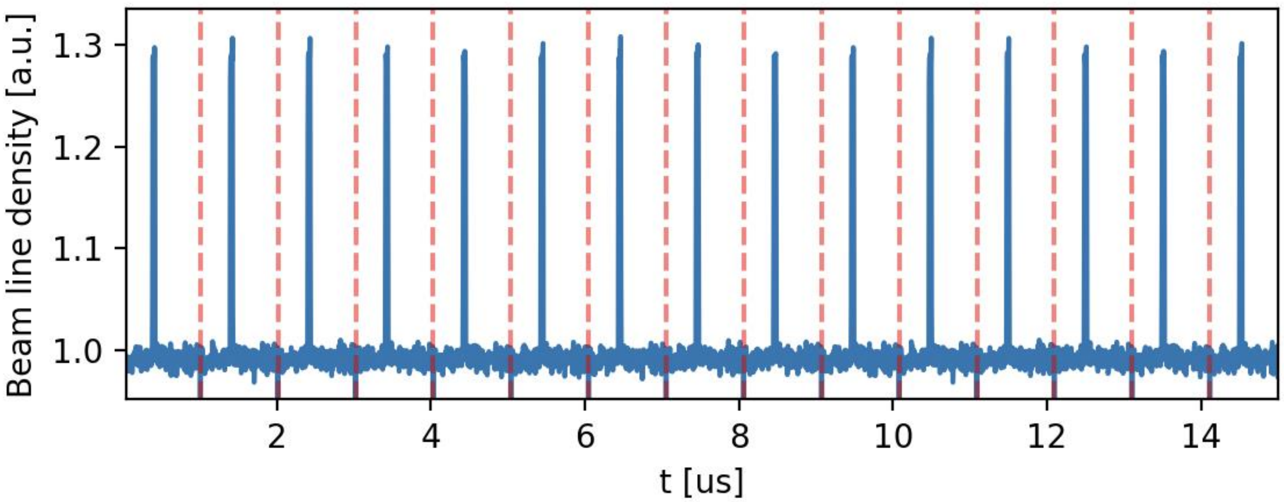


Test simulations – measuring the revolution frequency

We track a **beam with no momentum spread**, and we introduce a **perturbation on the beam line density**.

→ As there is no slippage, the perturbation is observed at each turn

→ It is possible to **measure the revolution frequency** by extracting the **main Fourier component** of the longitudinal profile (using nafflib)



The beam is generated with δ **= 0**

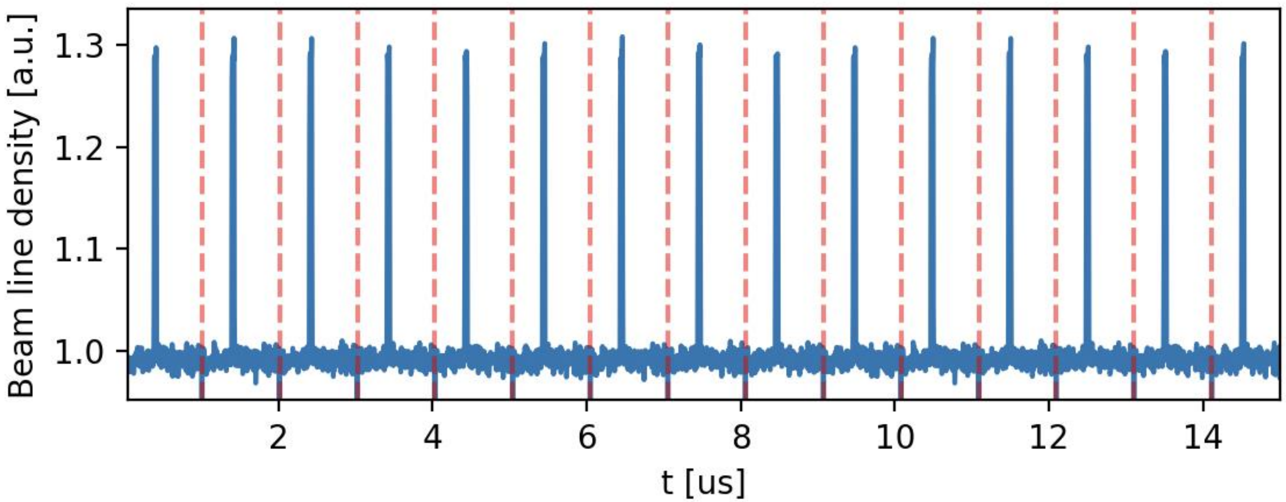
```
f_nominal (on momentum):    991.96599 kHz
```
```f_measured (off momentum): 991.96599 Hz```

Test simulations – measuring the revolution frequency

We track a **beam with no momentum spread**, and we introduce a **perturbation on the beam line density**.

→ As there is no slippage, the perturbation is observed at each turn

→ It is possible to **measure the revolution frequency** by extracting the **main Fourier component** of the longitudinal profile (using nafflib)



The beam is generated with δ = **-0.01**

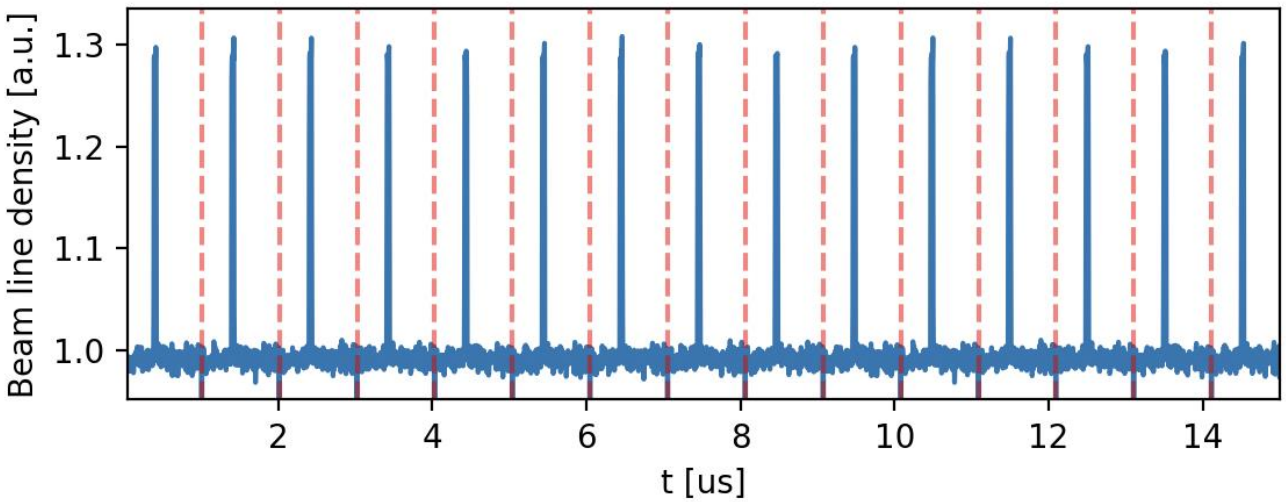
```
f_nominal (on momentum):    991.966 kHz
f_expected (off momentum):  985.271 kHz
f_measured (off momentum):  985.271 kHz
```

Test simulations – measuring the revolution frequency

We track a **beam with no momentum spread**, and we introduce a **perturbation on the beam line density**.

→ As there is no slippage, the perturbation is observed at each turn

→ It is possible to **measure the revolution frequency** by extracting the **main Fourier component** of the longitudinal profile (using nafflib)



The beam is generated with δ **= +0.01**

```
f_nominal (on momentum):    991.966 kHz
f_expected (off momentum):  998.580 kHz
f_measured (off momentum):  998.581 kHz
```

Frequency obtained from the line density agrees very well with expected one ☺

Summary and outlook

A method has been devised **to simulate coasting beams** with **tracking codes that use the reference path length *s* as independent variable**

→ The different revolution frequency among particles is accurately modeled

The method has been **implemented in Xsuite**:

- **No modifications in the conventional tracking elements**
- **Time synchronization** of particles is achieved by installing **dedicated SyncTime elements** in front of the collective elements (e.g. space charge, impedances)

Planned applications include:

- PSB with space charge (benchmark of coasting beam experiments)
- Fermilab IOTA ring with space charge
- ISIS-2 stability studies

Thanks for your attention!

```python
class SyncTime:

    def __init__(self, circumference, id, frame_relative_length=None,
                 at_start=False, at_end=False):
        if frame_relative_length is None:
            frame_relative_length = DEFAULT_FRAME_RELATIVE_LENGTH
        assert id > COAST_STATE_RANGE_START
        self.id = id
        self.frame_relative_length = frame_relative_length
        self.circumference = circumference
        self.at_start = at_start
        self.at_end = at_end

    def track(self, particles):

        assert isinstance(particles._context, xo.ContextCpu), (
                         'SyncTime only available for CPU for now')

        beta0 = particles._xobject.beta0[0]
        beta1 = beta0 / self.frame_relative_length
        beta0_beta1 = beta0 / beta1

        mask_alive = particles.state > 0

        zeta_min = -self.circumference/ 2 * beta0_beta1 + particles.s * (
                    1 - beta0_beta1)

        if (self.at_start and particles.at_turn[0] == 0
                and not (particles.state == -COAST_STATE_RANGE_START).any()): # done by the user
            mask_stop = mask_alive * (particles.zeta < zeta_min)
            particles.state[mask_stop] = -COAST_STATE_RANGE_START
            particles.zeta[mask_stop] += self.circumference * beta0 / beta1

        # Resume particles previously stopped
        particles.state[particles.state==-self.id] = 1
        particles.reorganize()

        # Identify particles that need to be stopped
        zeta_min = -self.circumference/ 2 * beta0_beta1 + particles.s * (1 - beta0_beta1)
        mask_stop = mask_alive & (particles.zeta < zeta_min)

        # Check if some particles are too fast
        mask_too_fast = mask_alive & (
            particles.zeta > zeta_min + self.circumference * beta0_beta1)
        if mask_too_fast.any():
            raise ValueError('Some particles move faster than the time window')

        # Update zeta for particles that are stopped
        particles.zeta[mask_stop] += beta0_beta1 * self.circumference

        # Stop particles
        particles.state[mask_stop] = -self.id

        if self.at_end:
            mask_alive = particles.state > 0
            particles.zeta[mask_alive] -= (
                self.circumference * (1 - beta0_beta1))

        if self.at_end and particles.at_turn[0] == 0:
            particles.state[particles.state==-COAST_STATE_RANGE_START] = 1

```

```python
def install_sync_time_at_collective_elements(line, frame_relative_length=None):

    circumference = line.get_length()

    ltab = line.get_table()
    tab_collective = ltab.rows[ltab.iscollective]
    for ii, nn in enumerate(tab_collective.name):
        cc = x=SyncTime(circumference=circumference,
                        frame_relative_length=frame_relative_length,
                        id=COAST_STATE_RANGE_START + ii + 1)
        line.insert_element(element=cc, name=f'synctime_{ii}', at=nn)

    synctime_start = SyncTime(circumference=circumference,
                        frame_relative_length=frame_relative_length,
                        id=COAST_STATE_RANGE_START + len(tab_collective)+1,
                        at_start=True)
    synctime_end = SyncTime(circumference=circumference,
                        frame_relative_length=frame_relative_length,
                        id=COAST_STATE_RANGE_START + len(tab_collective)+2,
                        at_end=True)

    line.insert_element(element=synctime_start, name='synctime_start', at_s=0)
    line.append_element(synctime_end, name='synctime_end')

def prepare_particles_for_sync_time(particles, line):
    synctime_start = line['synctime_start']
    beta0 = particles._xobject.beta0[0]
    beta1 = beta0 / synctime_start.frame_relative_length
    beta0_beta1 = beta0 / beta1
    zeta_min = -synctime_start.circumference/ 2 * beta0_beta1 + particles.s * (
                1 - beta0_beta1)
    mask_alive = particles.state > 0
    mask_stop = mask_alive * (particles.zeta < zeta_min)
    particles.state[mask_stop] = -COAST_STATE_RANGE_START
    particles.zeta[mask_stop] += synctime_start.circumference * beta0 / beta1
```