The most beautiful line you can draw with Kalman filter

by david ivanishvili

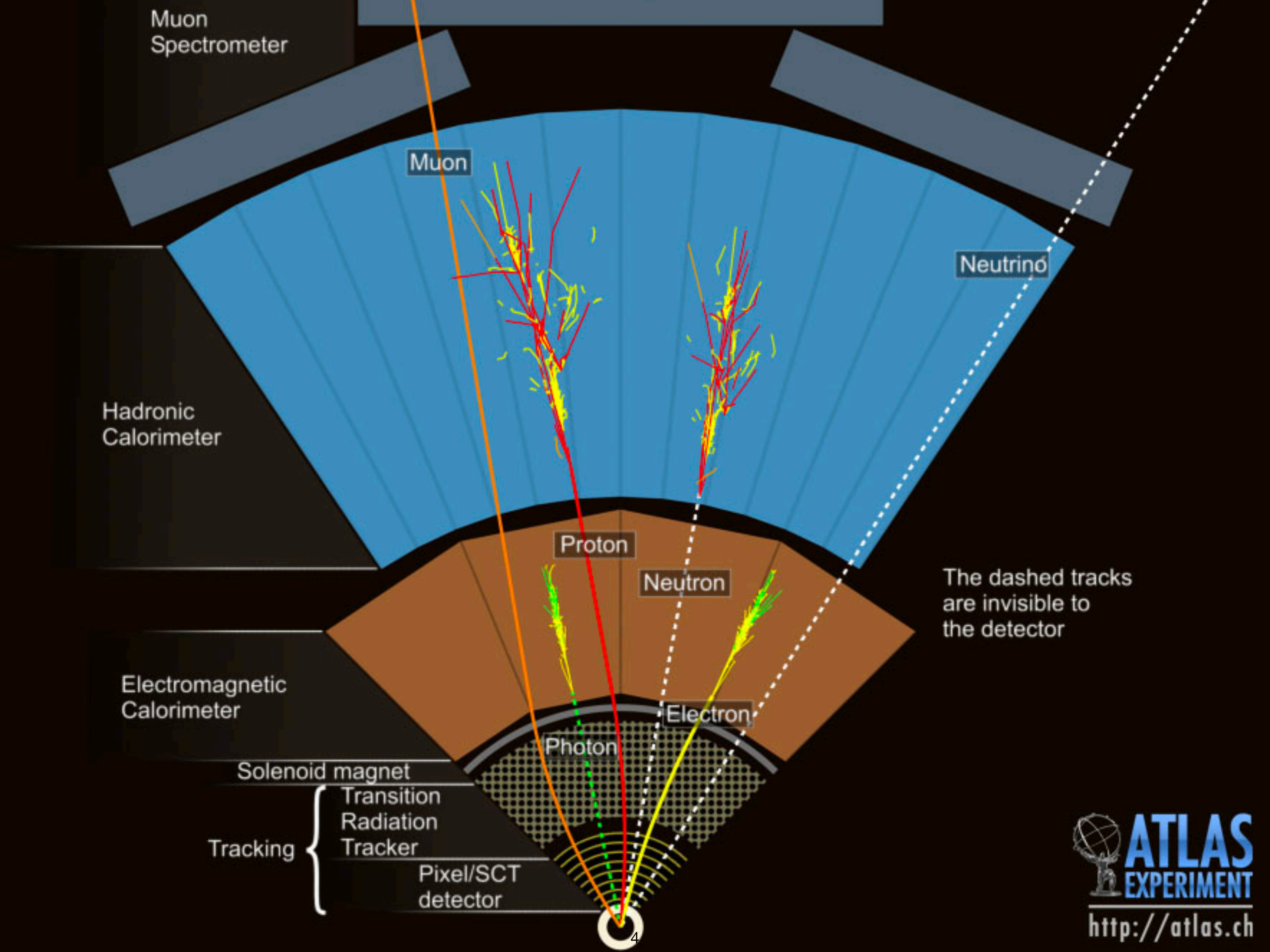
Valeriia (Lera) Lukashenko

iCSC 2023

Based on/your read list

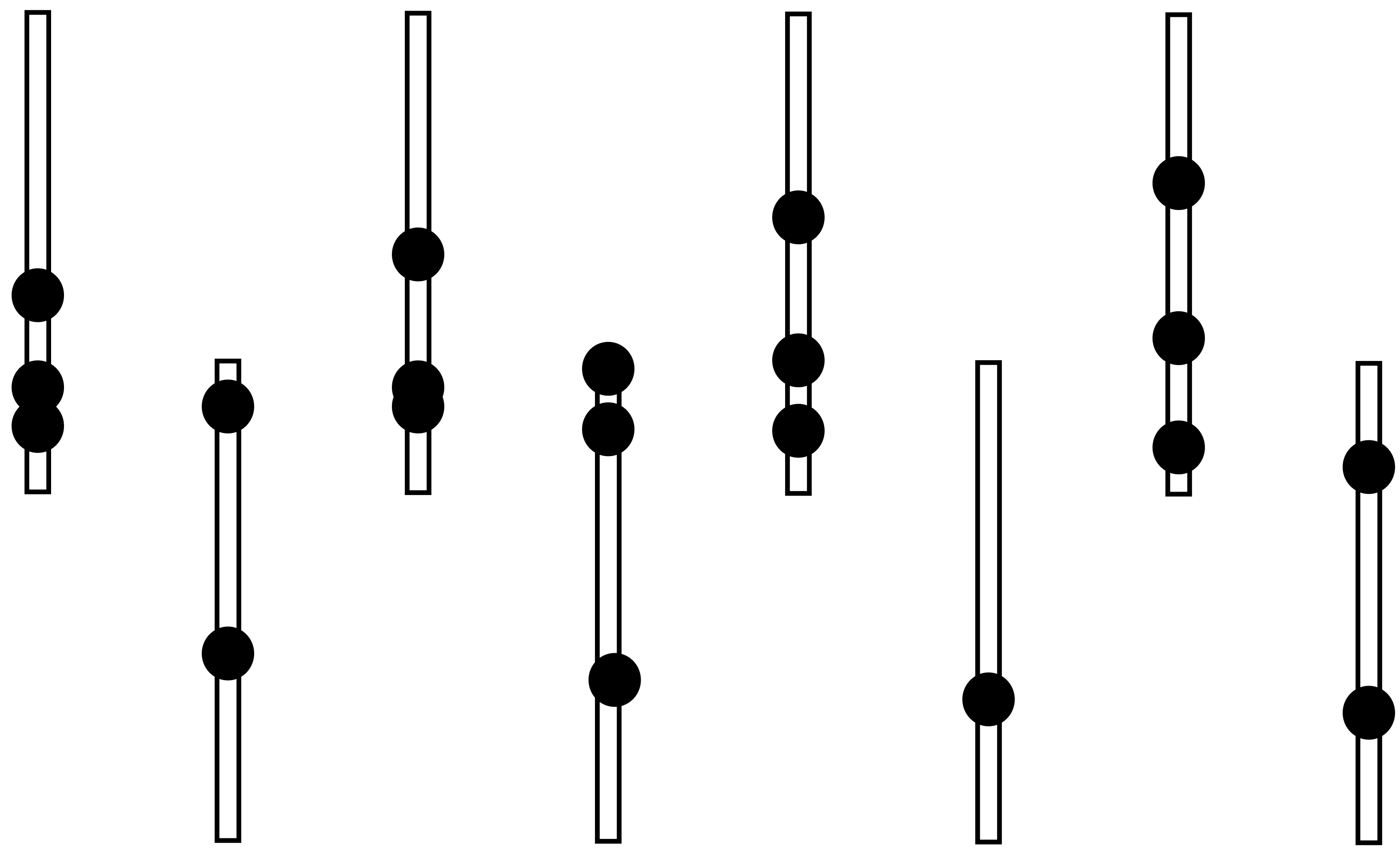
- *Wouter Hulsbergen* "Charged particle reconstruction and alignment" - all the math can be found here
- *Jeroen Van Tilburg* "Track Simulation and reconstruction in LHCb" - the to-go tracking thesis in LHCb, useful Kalman filter reference
- *R.Frühwirth, A.Strandlie* "Pattern Recognition, Tracking and Vertex Reconstruction" - the most full book on tracking I know of
- *R.Frühwirth* Application of Kalman Filtering to Tracking and Vertex Fitting - eternal classics
- *X. Ai, G. Mania, H.Gray, M.Kuhn, N. Styles* A GPU-based Kalman Filter for Track Fitting

Introduction

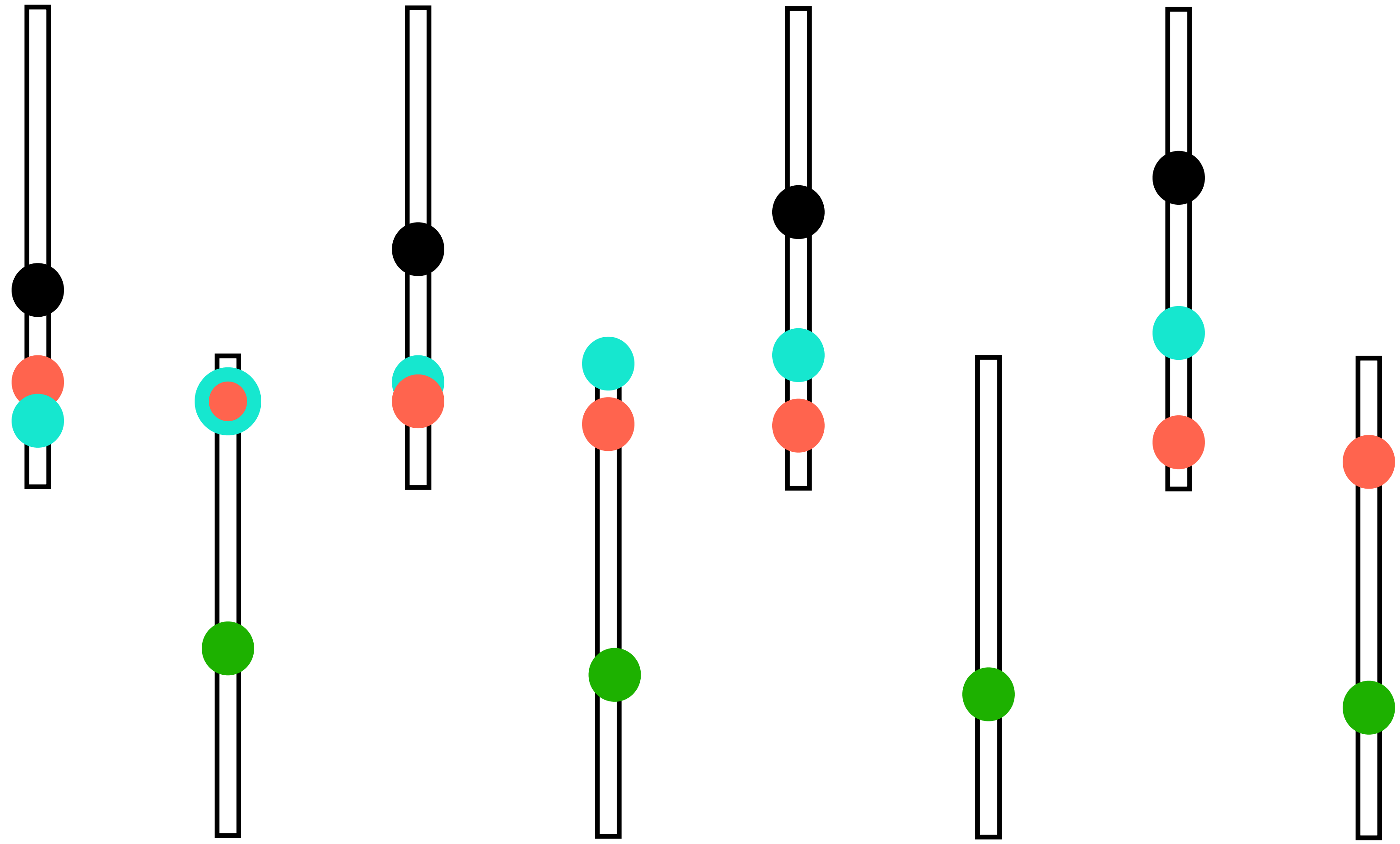


Muon Spectrometer
Muon
Neutrino
Hadronic Calorimeter
Proton
Neutron
The dashed tracks are invisible to the detector
Electromagnetic Calorimeter
Electron
Photon
Solenoid magnet
Tracking
Transition Radiation Tracker
Pixel/SCT detector
ATLAS EXPERIMENT
http://atlas.ch

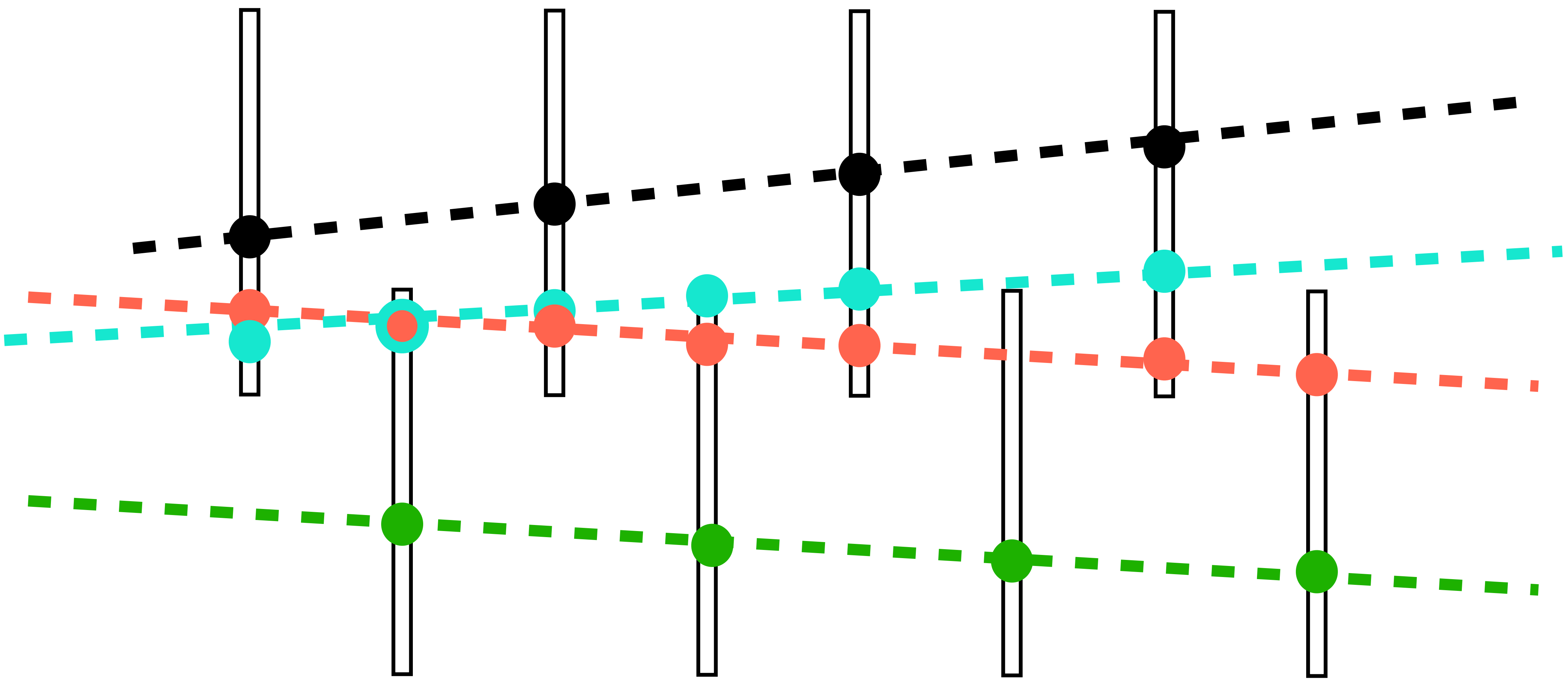
What is a track?



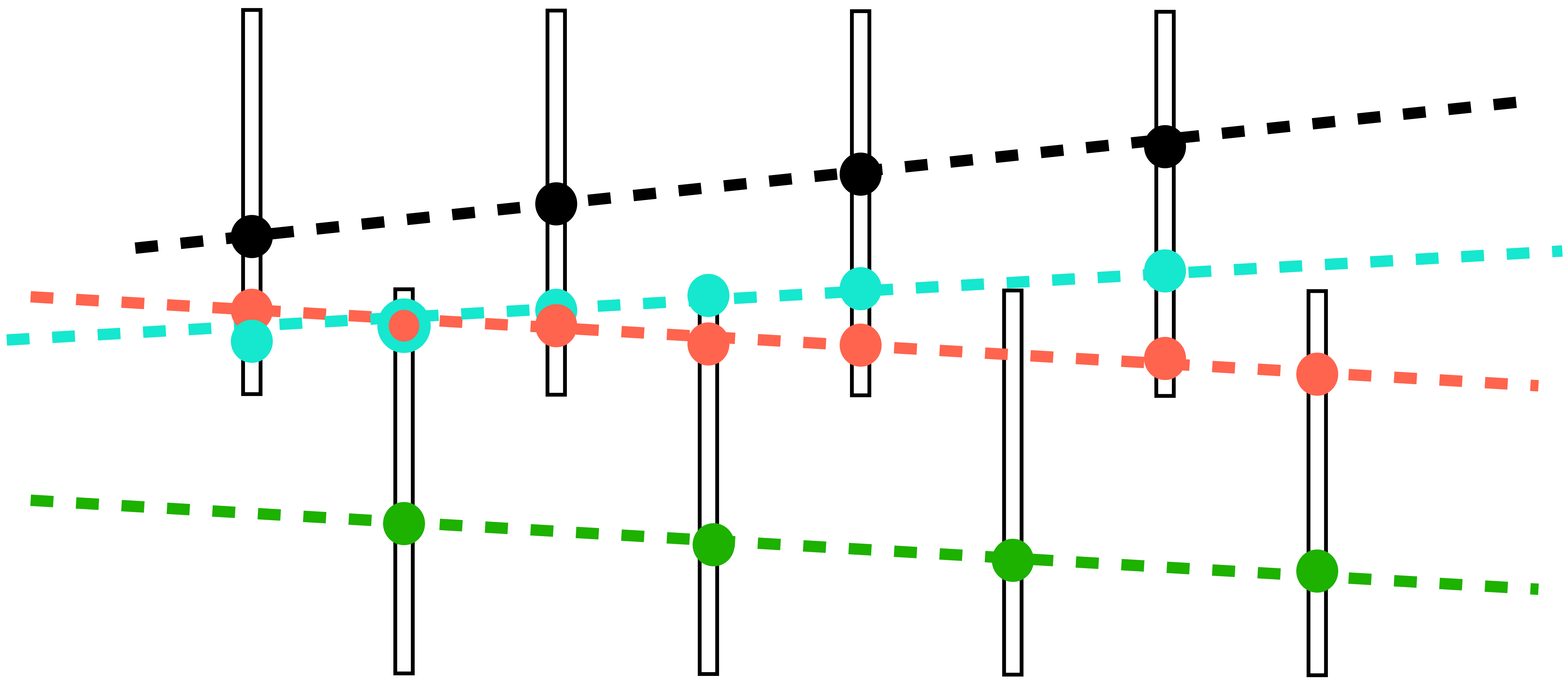
What is a track?



What is a track?

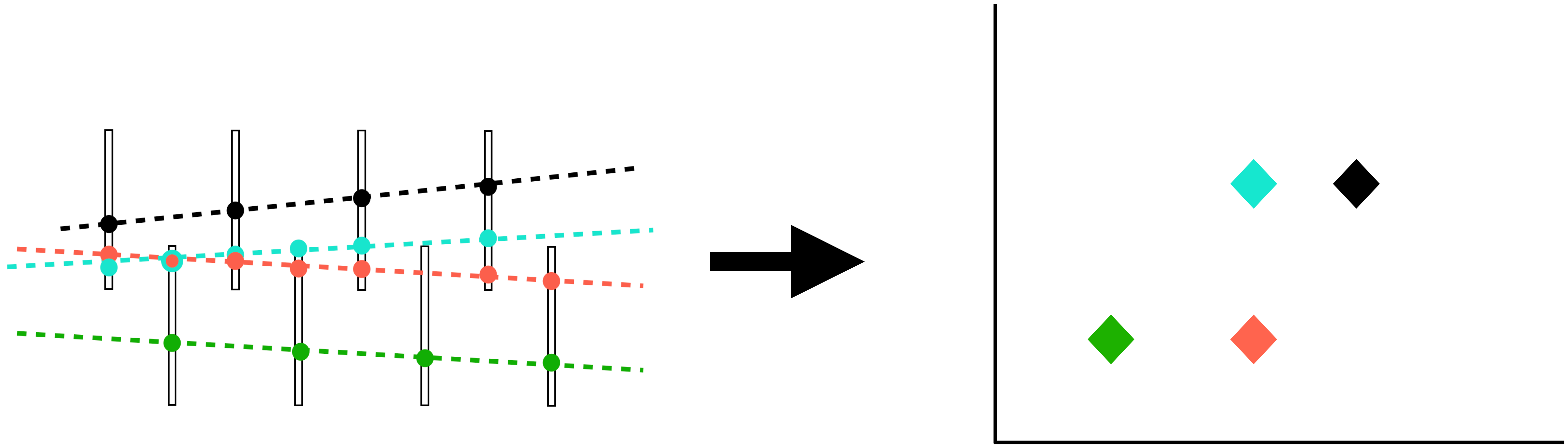


What is a track?



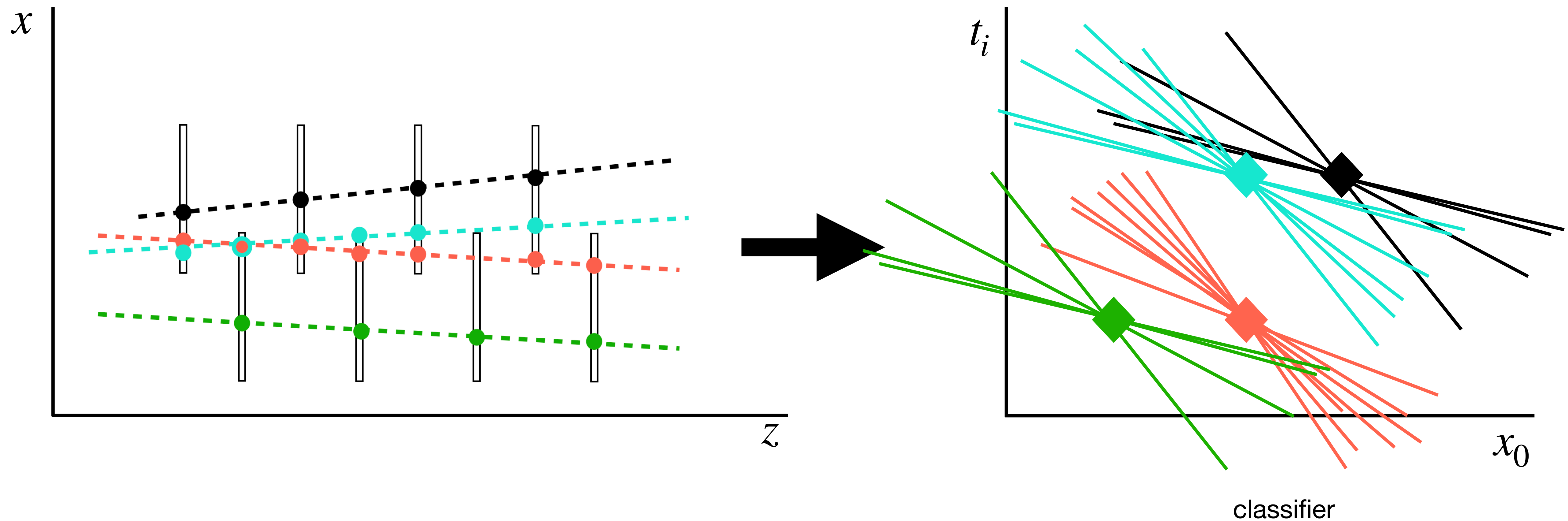
Which hit belongs to which track?

Which hit belongs to which track



classifier

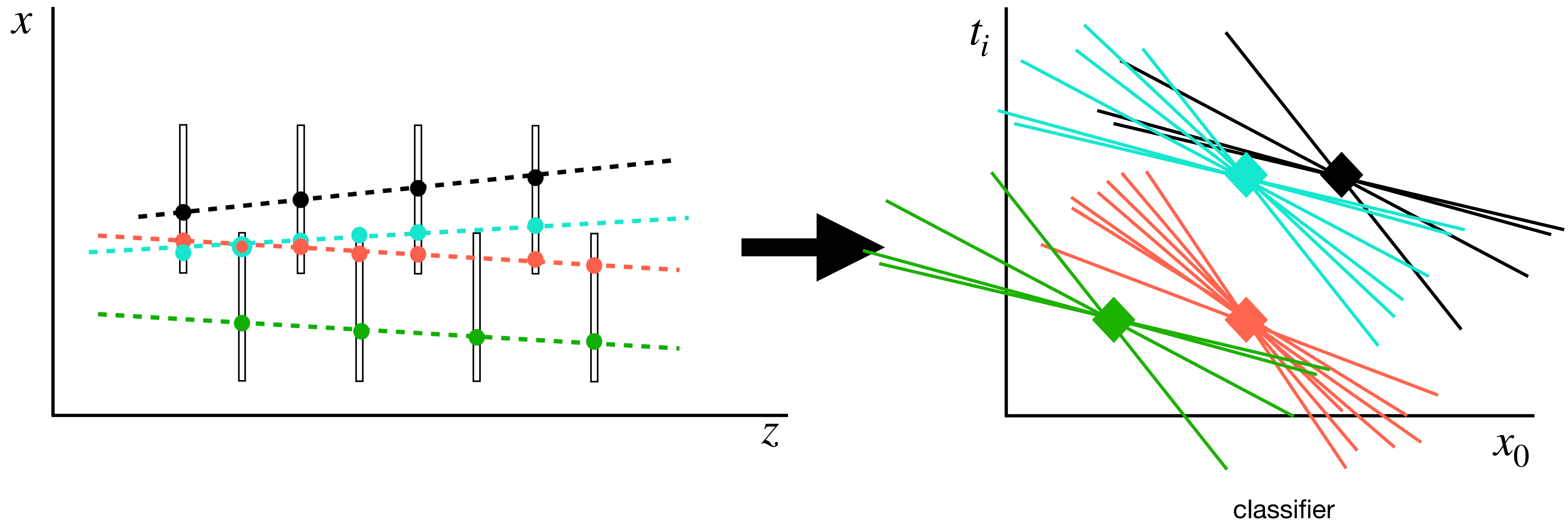
Which hit belongs to which track



Hough transform

$$m_i(x, z) \rightarrow t_i = \frac{x - x_0}{z}$$

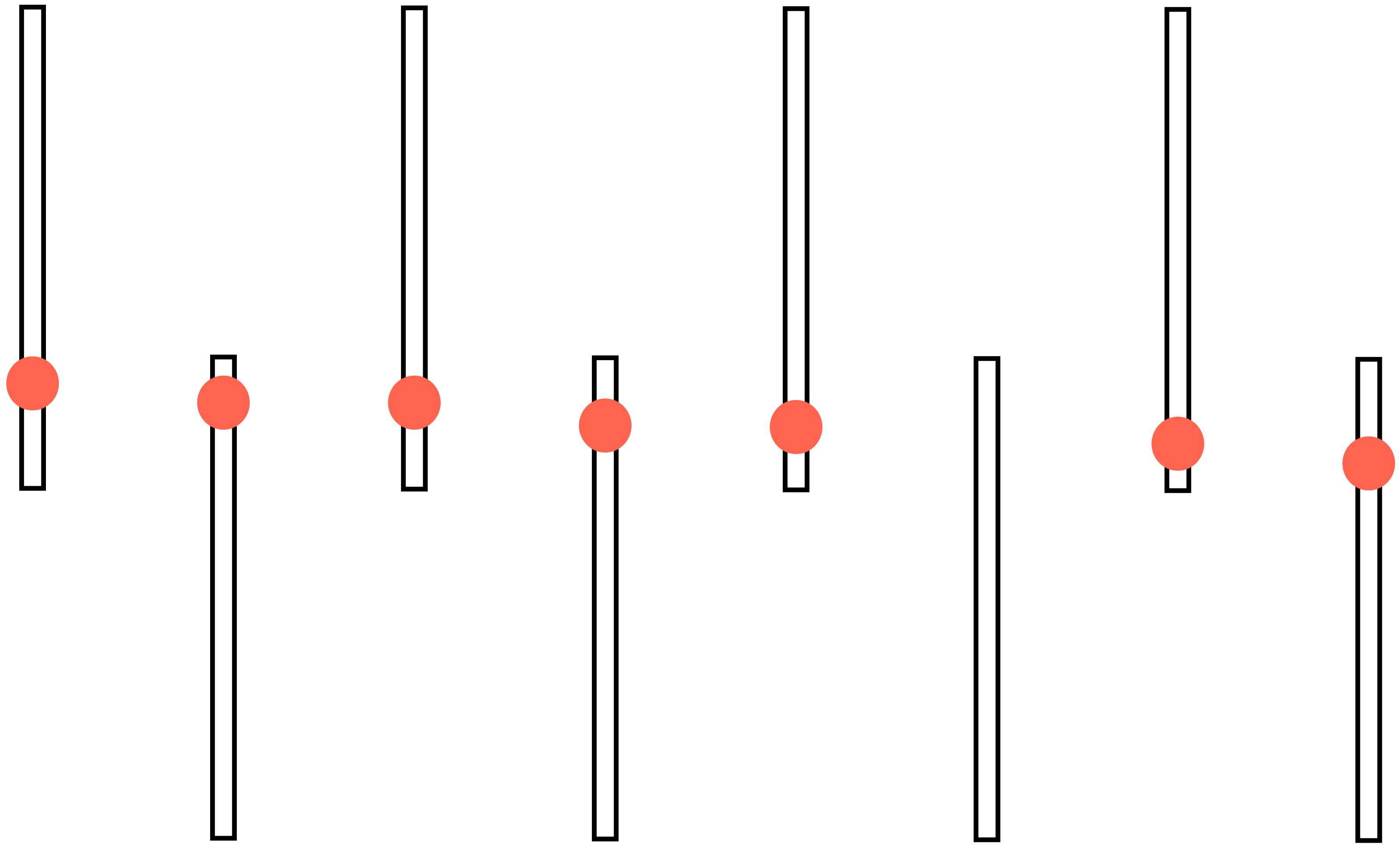
Which hit belongs to which track



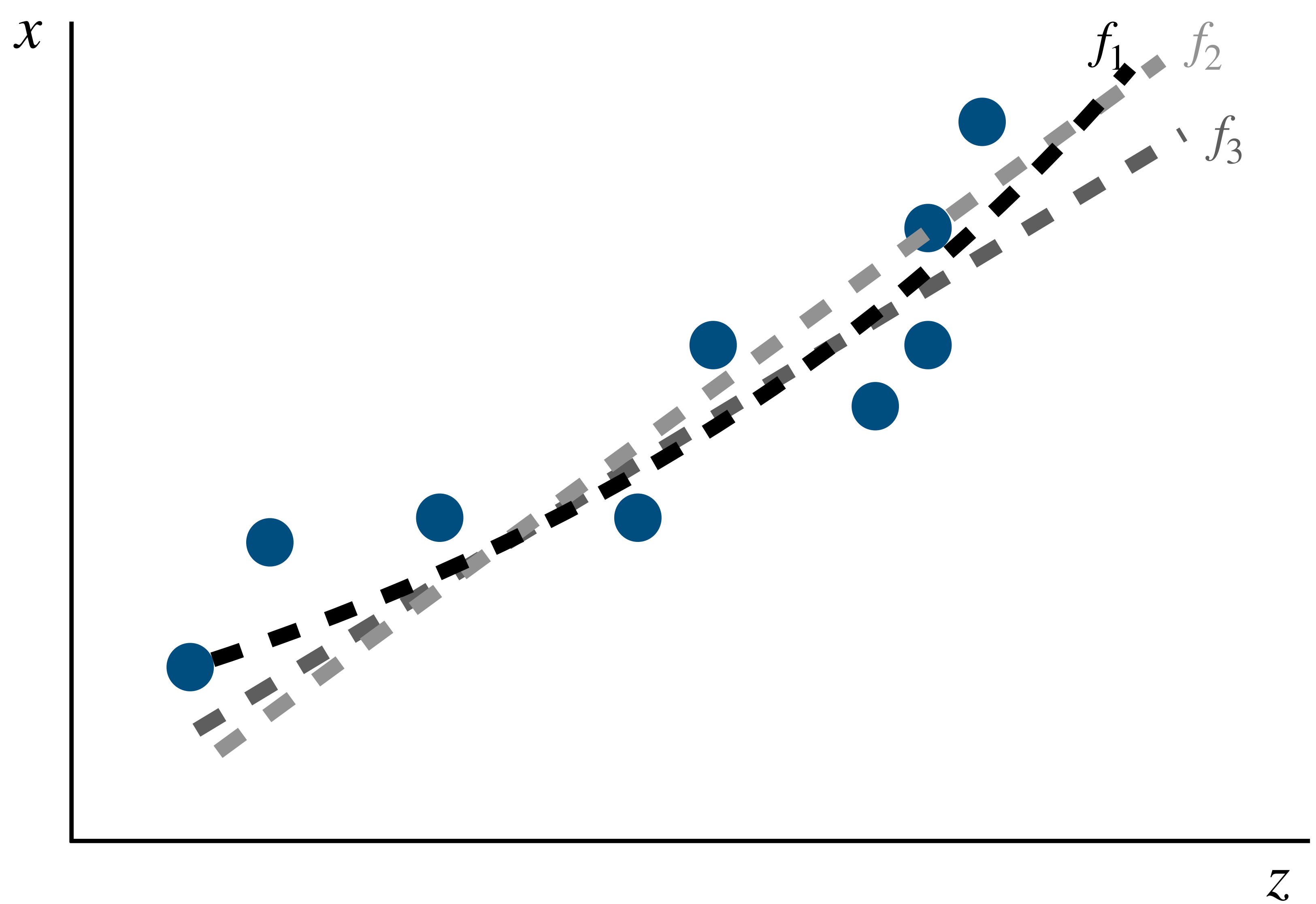
- machine learning
- “seeding” algorithm (local patter recognition)
- etc.

How do I draw a line?

aka track fitting



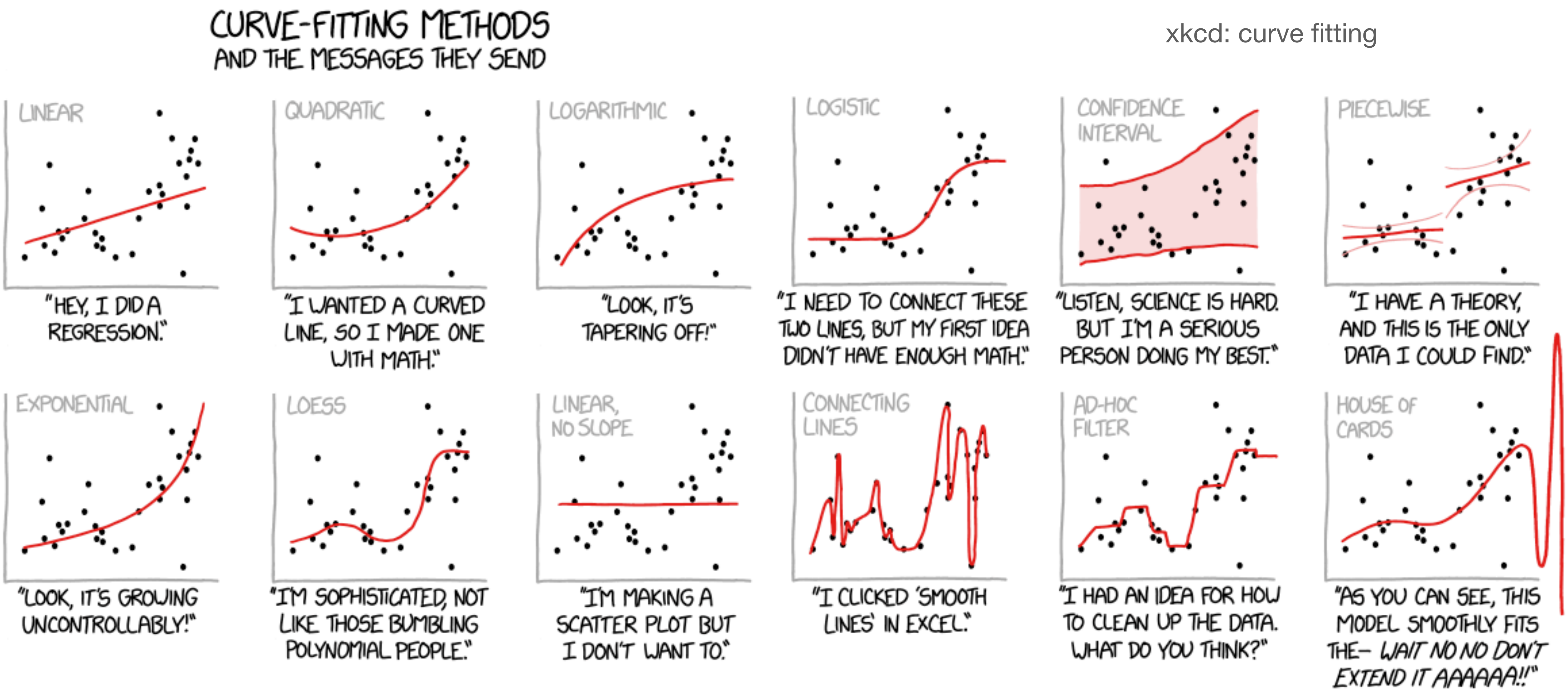
Reminder: fitting



$$f_1 = \sum_n \alpha_n z^n \qquad f_2 = \sum_n \beta_n z^n \qquad f_3 = \sum_n \gamma_n z^n$$

fitting = estimating parameters of the model

Reminder: fitting



xkcd: curve fitting

two simple facts to spell out:

- choosing the model is purely **subjective**
- if model **"fits"** it does **not** mean it is **"true"**

Reminder: fitting

	$\mathscr{P}$	$\mathscr{L}$
	probability	likelihood
know	**model**	**data**
how likely	**data**	**model**

Reminder: fitting

Maximum Likelihood Estimator

$$\mathscr{L}(\alpha; z) = \prod_i \mathscr{P}(z_i; \alpha)$$

$$max\left(\mathscr{L}\right) \rightarrow \frac{d\mathscr{L}(\alpha; z)}{d\alpha} = 0 \rightarrow \hat{\alpha}$$

Reminder: fitting

Maximum Likelihood Estimator

$$ln\mathscr{L}(\alpha; z) = \sum_i ln\mathscr{P}(z_i; \alpha)$$

$$min\Big(-\mathscr{L}\Big) \rightarrow \frac{d(-ln\mathscr{L})(\alpha; z)}{d\alpha} = 0 \rightarrow \hat{\alpha}$$

- $\sum$ is easier than $\prod$
- minimisation and maximisation are the same, but the convention is to represent maximum likelihood estimation via minimisation

Reminder: fitting

Maximum Likelihood Estimator

$$ln\mathscr{L}(\alpha; z) = \sum_i ln\mathscr{P}(z_i; \alpha)$$

$$min\left(-\mathscr{L}\right) \rightarrow \frac{d(-ln\mathscr{L})(\alpha; z)}{d\alpha} = 0 \rightarrow \hat{\alpha}$$

$$V(\hat{\alpha}) = \sigma^2 = \left[E\left(-\frac{\partial^2 ln\mathscr{L}}{\partial\alpha^2}\right)\right]^{-1}$$

V is variance aka spread aka uncertainty

Reminder: fitting, example

Least Square Estimator : χ^2

If $\mathscr{P}(z_i;\alpha) = \mathscr{N}(h_i(\alpha), \sigma_i)$, then

What I see

What I expect

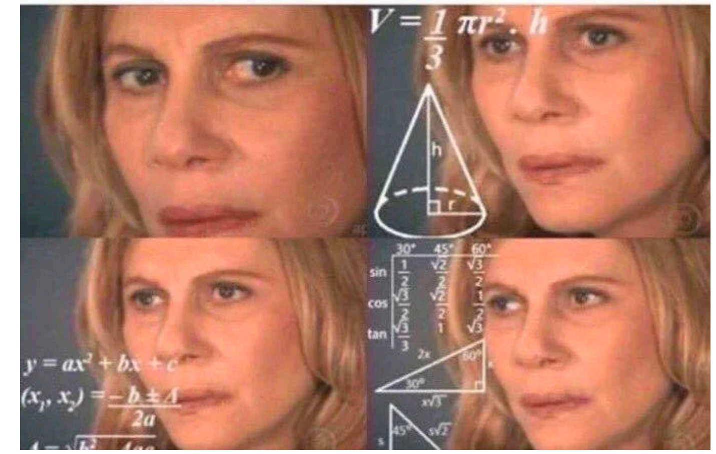
$$\sum_i ln\mathscr{P} \rightarrow \chi^2 = \sum_i \left(\frac{z_i - h_i(\alpha)}{\sigma_i}\right)^2$$

How good my vision is

$$min\left(\sum_i ln\mathscr{P}\right) \Rightarrow \frac{d\chi^2}{d\alpha} = 0 \rightarrow \hat{\alpha}$$

χ^2 formalism

linear case



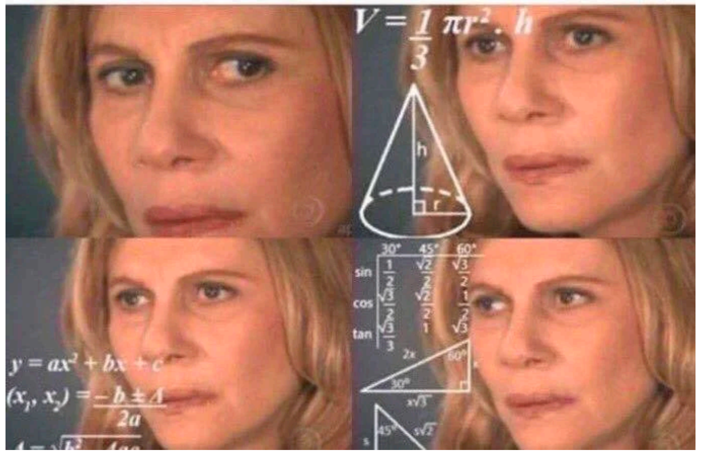
What I see What I expect

$$\chi^2 = \sum_i \left(\frac{z_i - h_i(\alpha)}{\sigma_i}\right)^2 = \big(z - h(\alpha)\big)^T(\sigma^2)^{-1}\big(z - h(\alpha)\big)$$

How good my vision is

tensor form

χ^2 formalism
linear case



What I see What I expect

$$\chi^2 = \sum_i \left(\frac{z_i - h_i(\alpha)}{\sigma_i}\right)^2 = \left(z - h(\alpha)\right)^T V^{-1} \left(z - h(\alpha)\right)$$

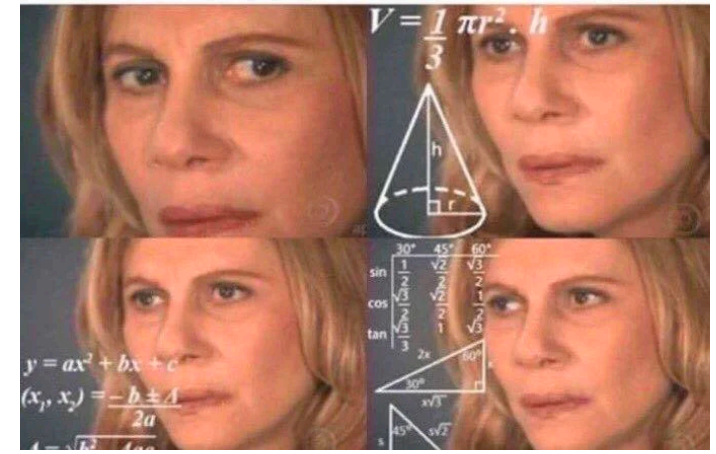
tensor form

How good my vision is

$$h(\alpha) = h_0 + H\alpha \qquad H = \frac{dh(\alpha)}{d\alpha}$$

χ^2 formalism

Solution



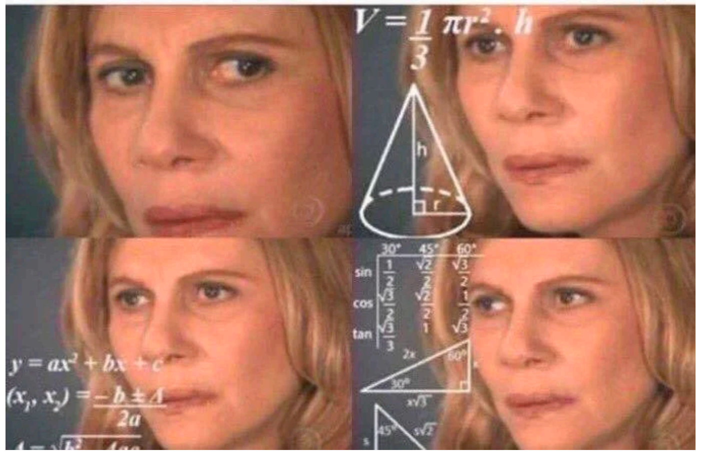
$$\hat{\alpha} = (H^T V^{-1} H)^{-1} H^T V^{-1} (z - h_0)$$

Are you worried about anything in this formula?
(note variance matrix - is a positive definite and diagonal matrix by definition)

$$h(\alpha) = h_0 + H\alpha \qquad H = \frac{dh(\alpha)}{d\alpha}$$

χ^2 formalism

Solution

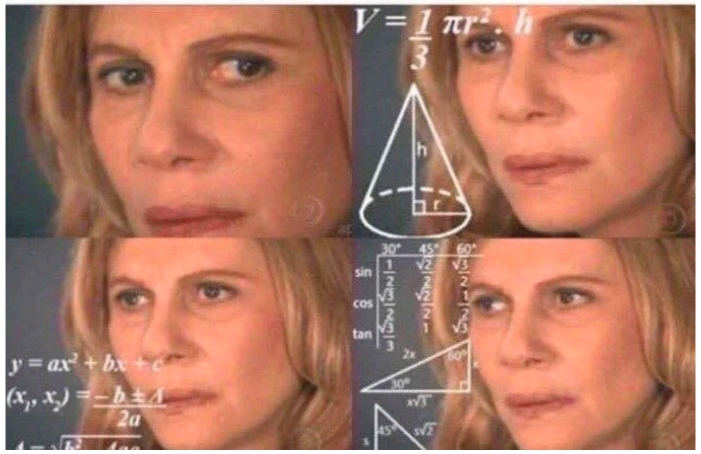


$$\hat{\alpha} = \boxed{(H^T V^{-1} H)^{-1}} H^T V^{-1} (z - h_0)$$

$$h(\alpha) = h_0 + H\alpha \qquad H = \frac{dh(\alpha)}{d\alpha}$$

χ^2 formalism

Solution



$$\hat{\alpha} = \boxed{(H^T V^{-1} H)^{-1}} H^T V^{-1} (z - h_0)$$

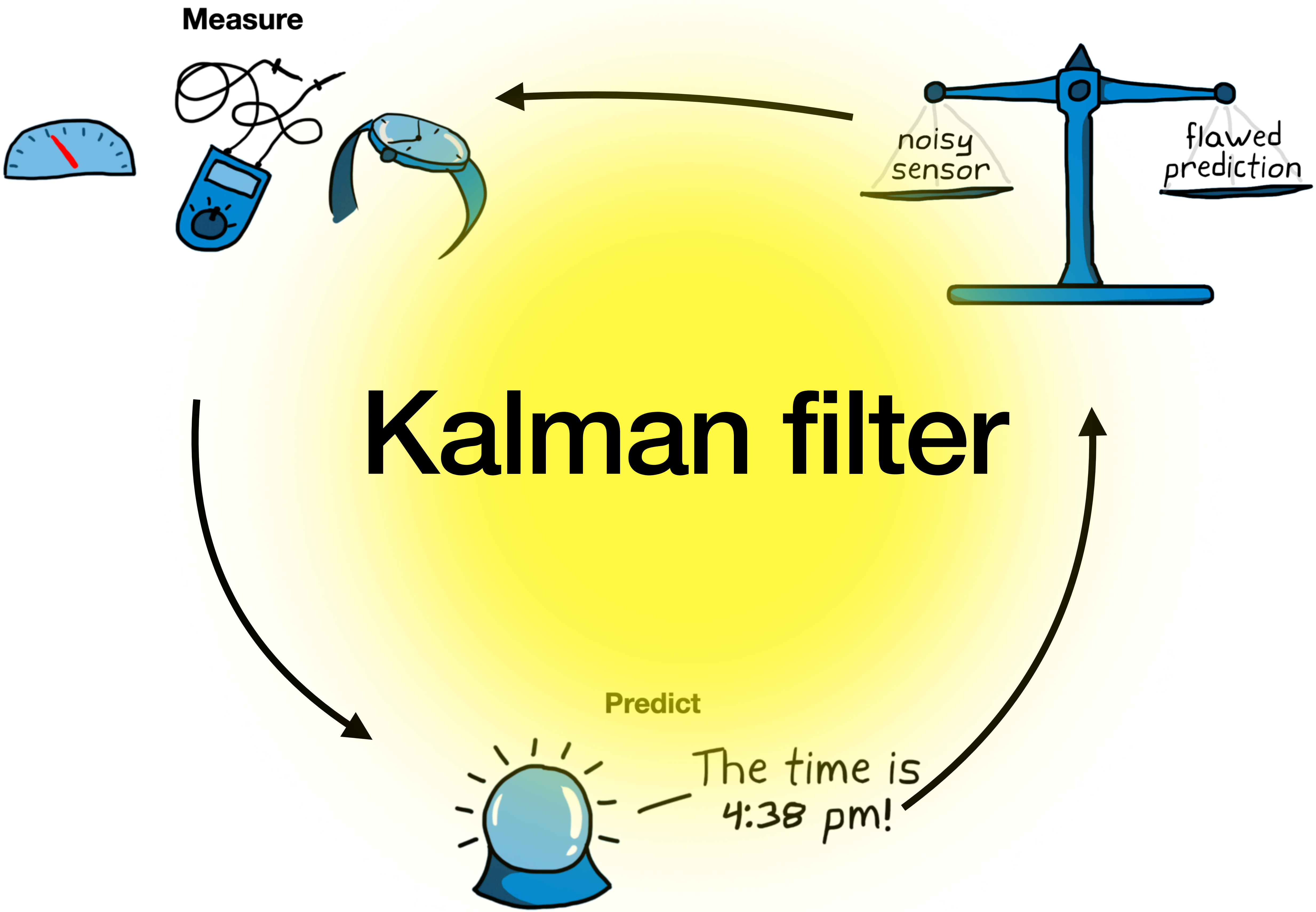
$$det(H^T V^{-1} H) \neq 0$$

$det(H^T V^{-1} H) = 0$: Underconstrained problem

$$h(\alpha) = h_0 + H\alpha \qquad H = \frac{dh(\alpha)}{d\alpha}$$

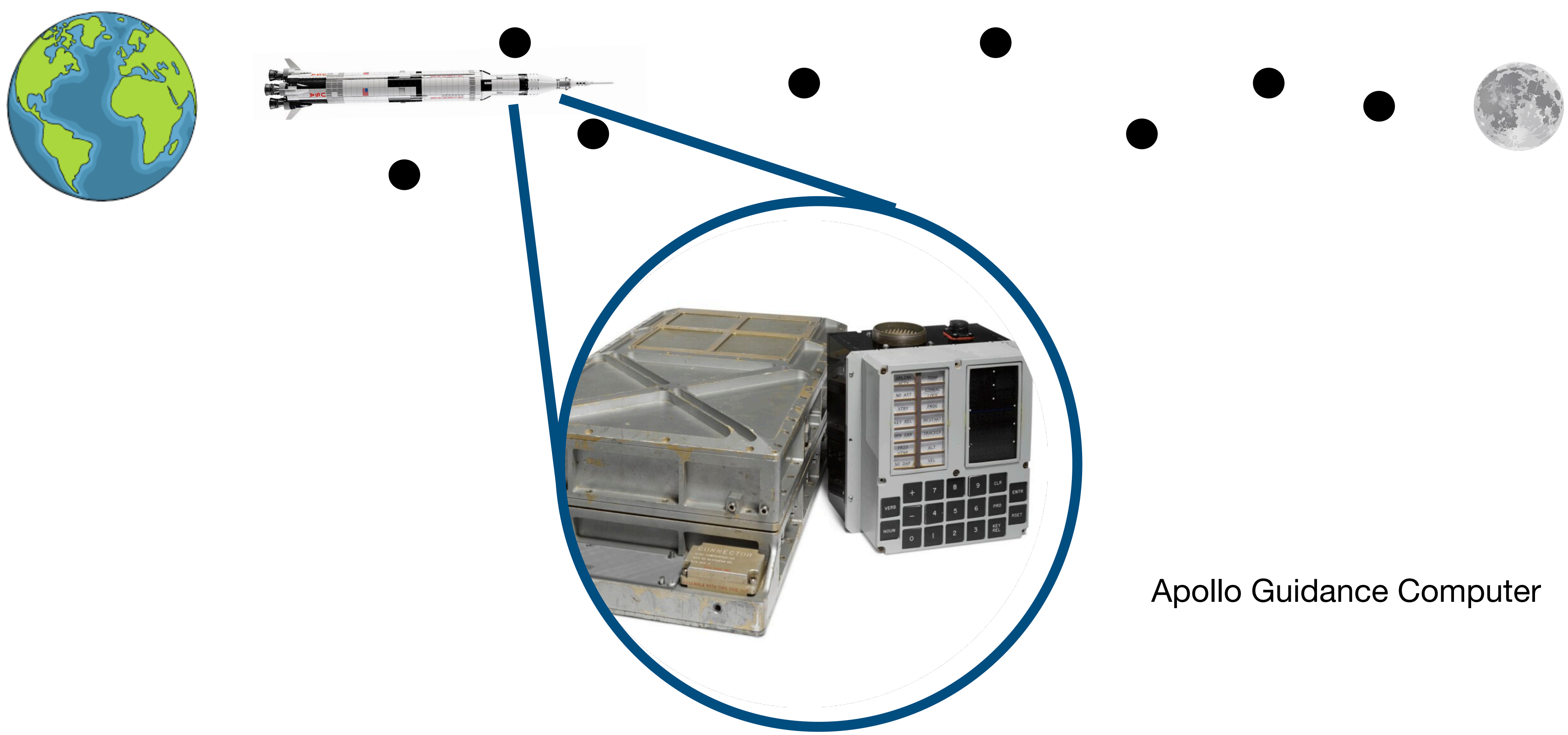
$$det(H^T V^{-1} H) = 0$$

- some linear combination of elements of α have no finite variance $\Rightarrow$ **unconstrained degrees of freedom**; can be isolated by diagonalizing the $H^T V^{-1} H$.
- the problem is always underconstrained for models with more parameters than data points



By Brian Douglas

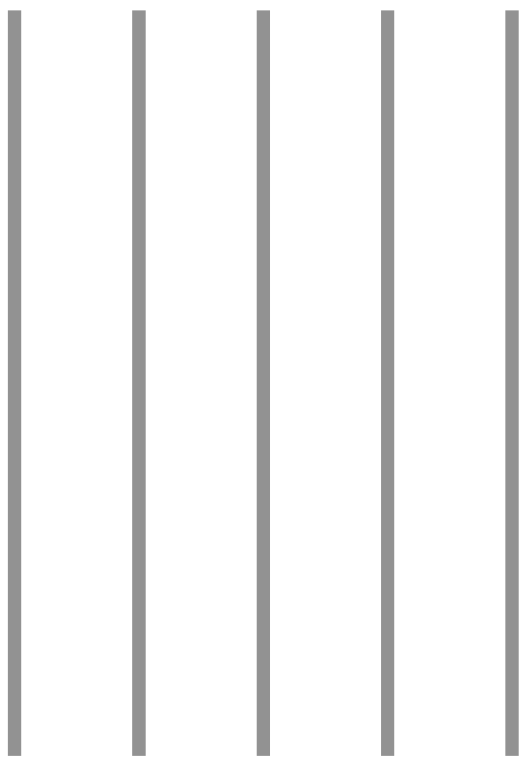
Apollo 11 mission



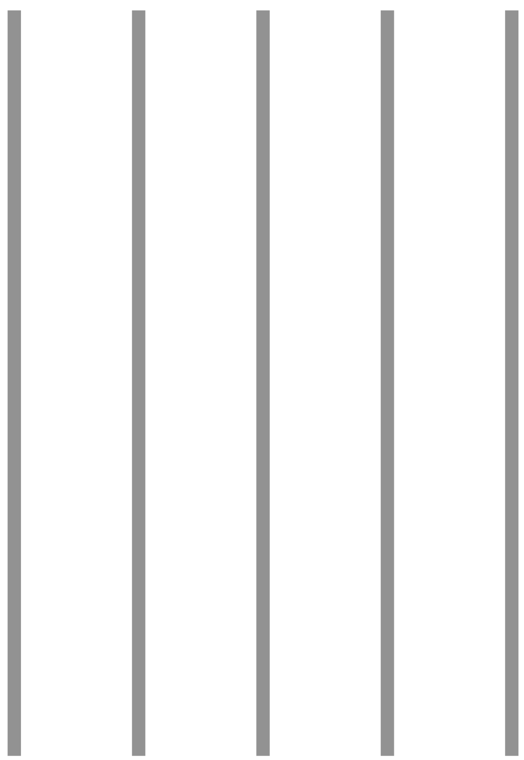
Apollo Guidance Computer

Problem: knowing trajectory of the spaceship with very limited computer resources and irregular measurements

Kalman formalism

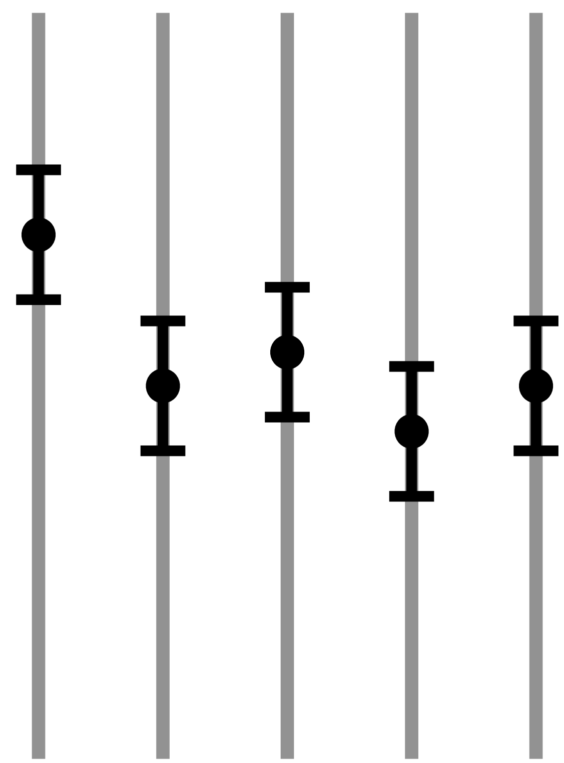


Global fit

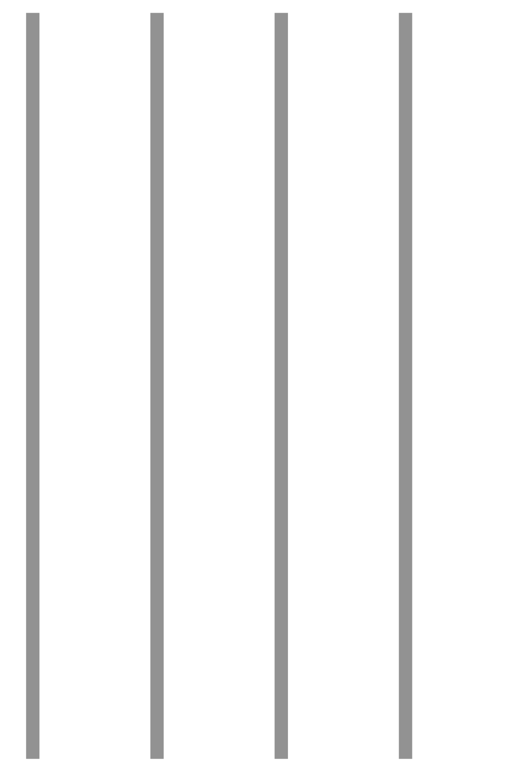


Kalman filter

Kalman formalism

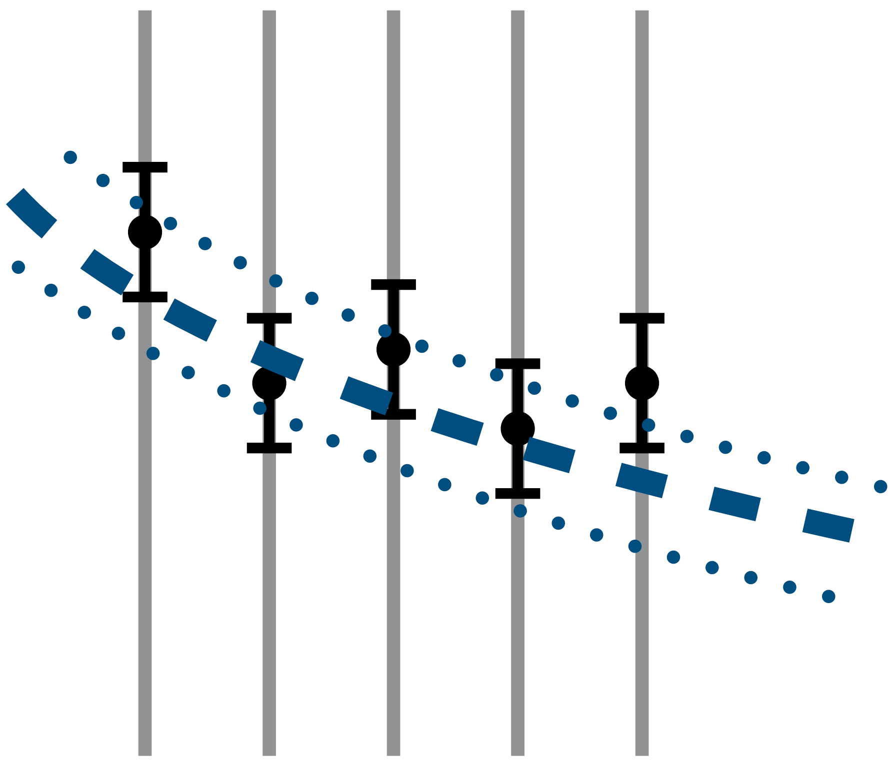


Global fit

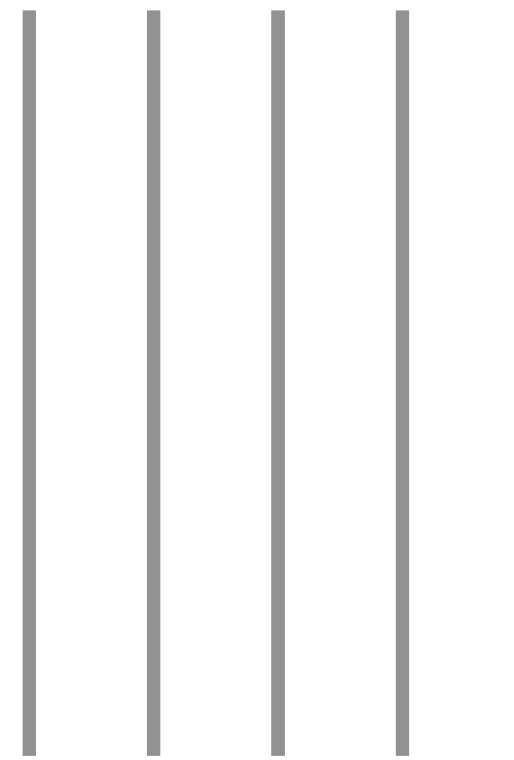


Kalman filter

Kalman formalism

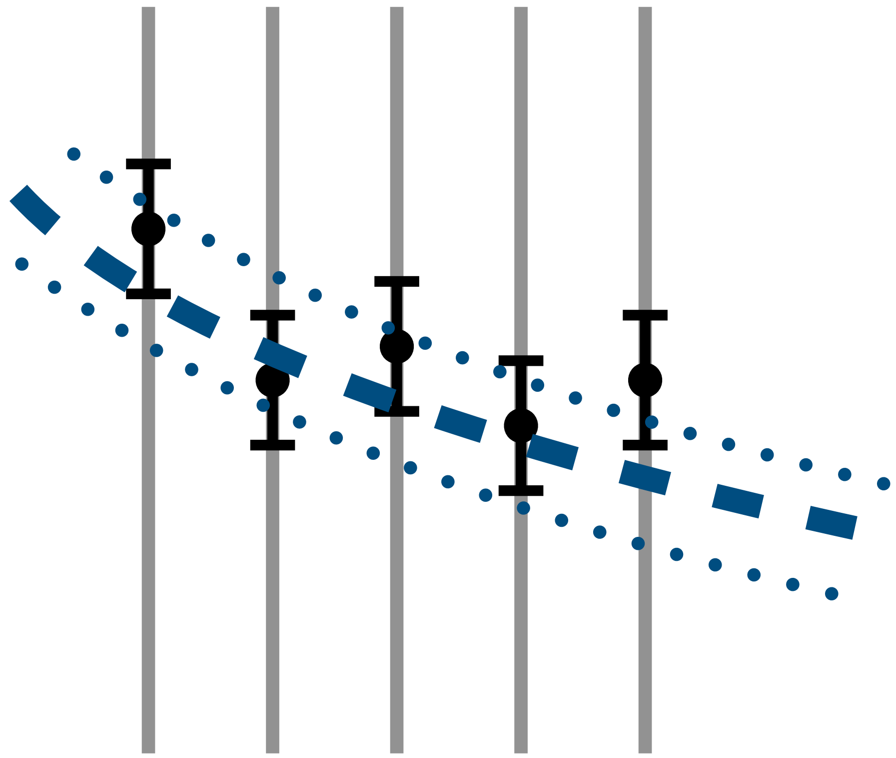


Global fit

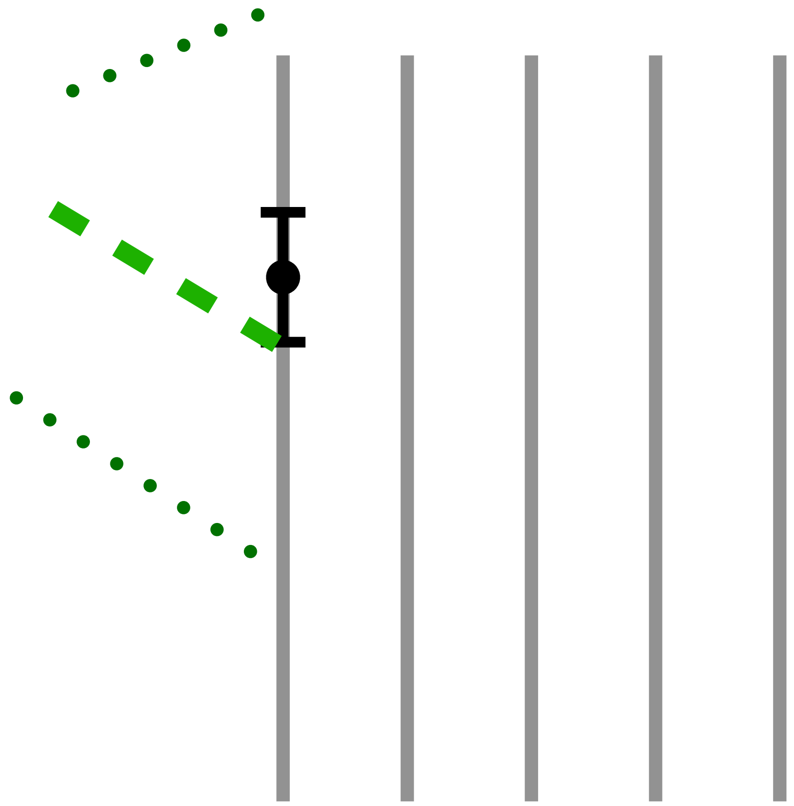


Kalman filter

Kalman formalism

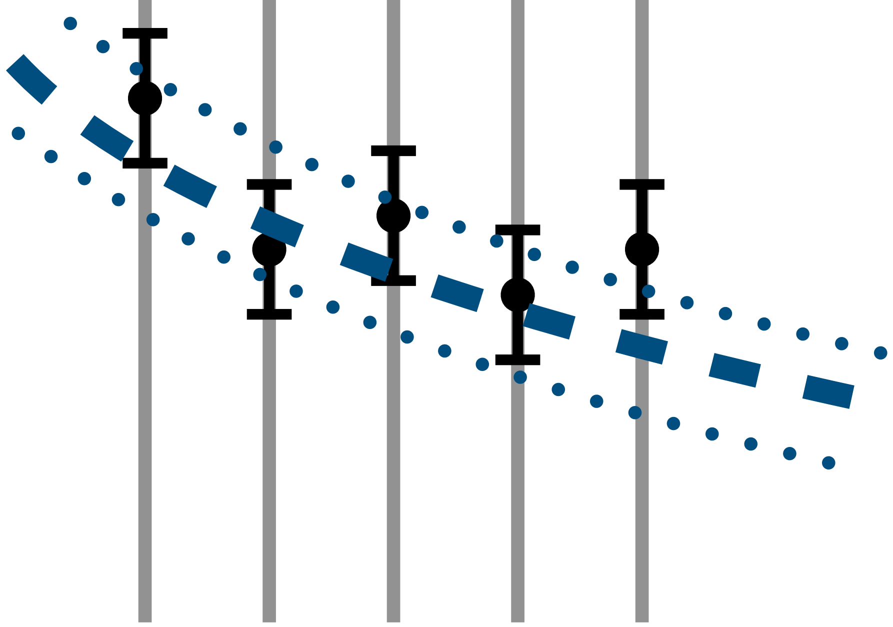


Global fit

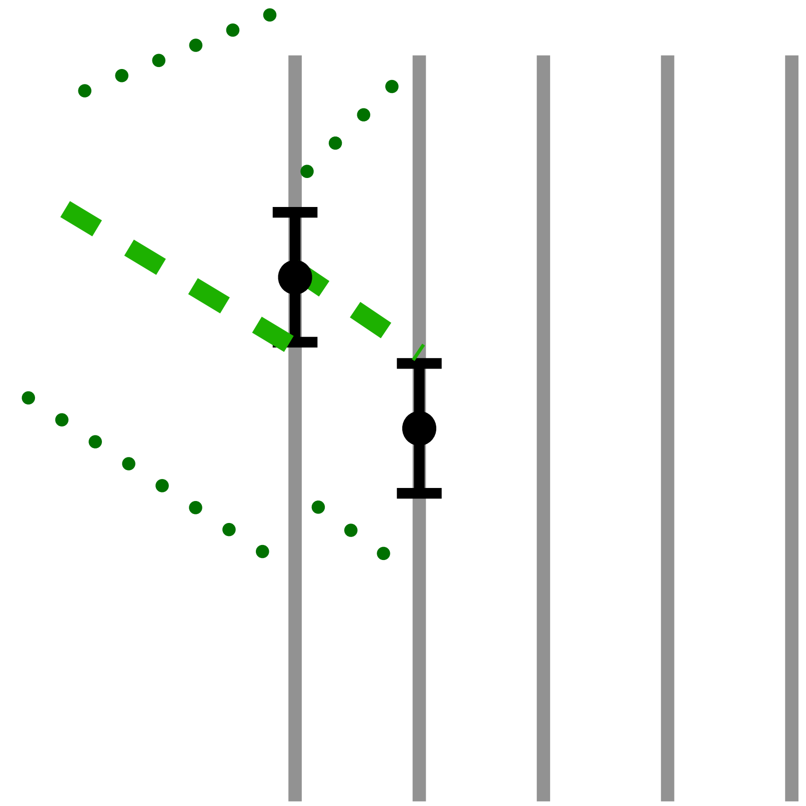


Kalman filter

Kalman formalism

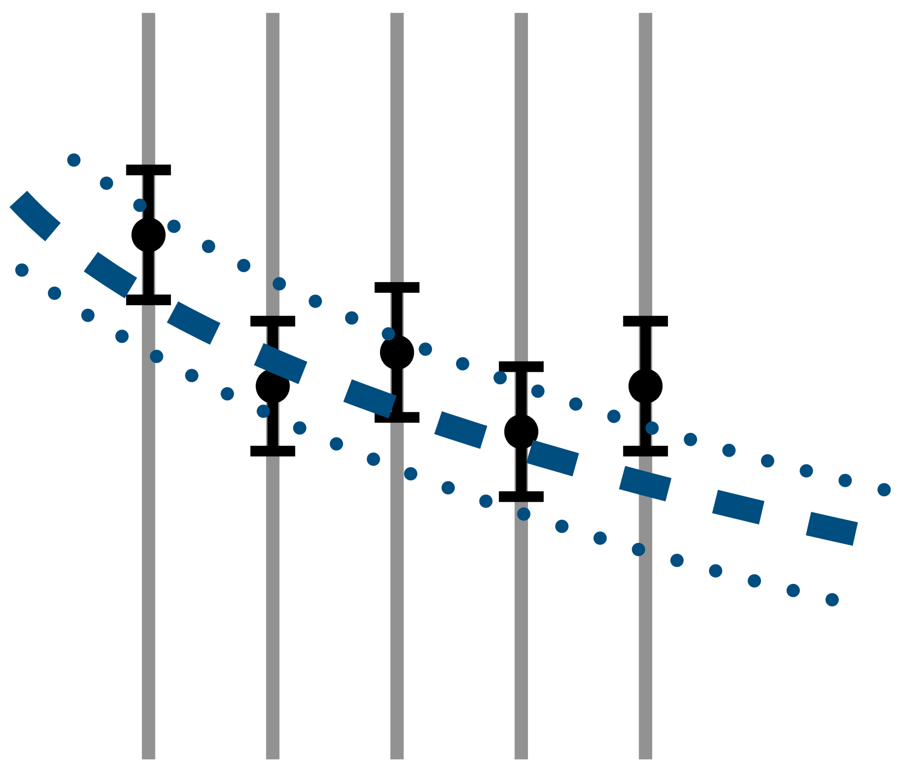


Global fit

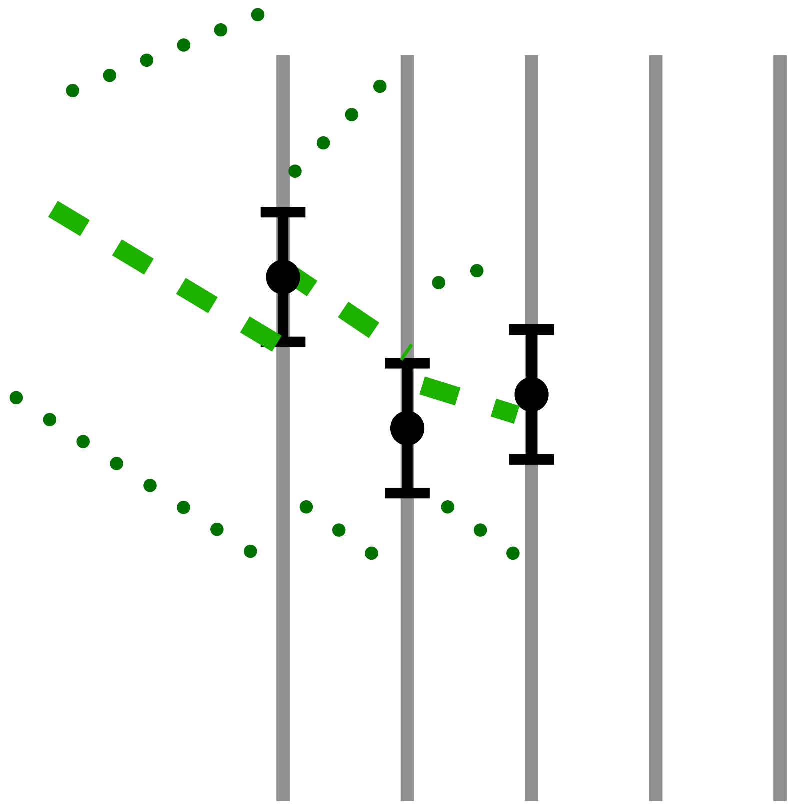


Kalman filter

Kalman formalism

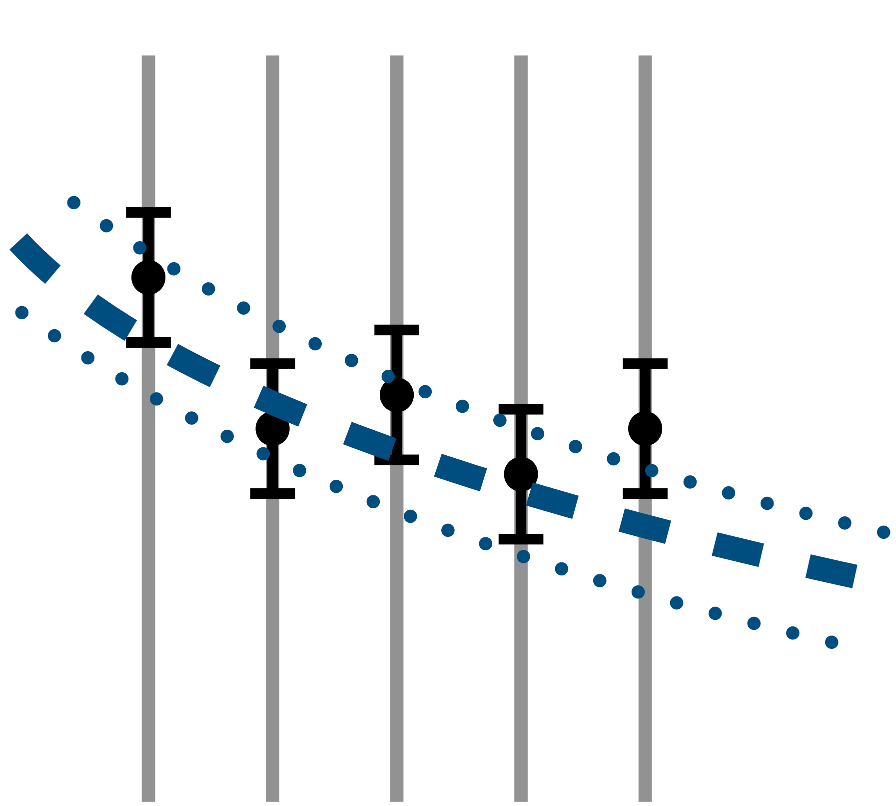


Global fit

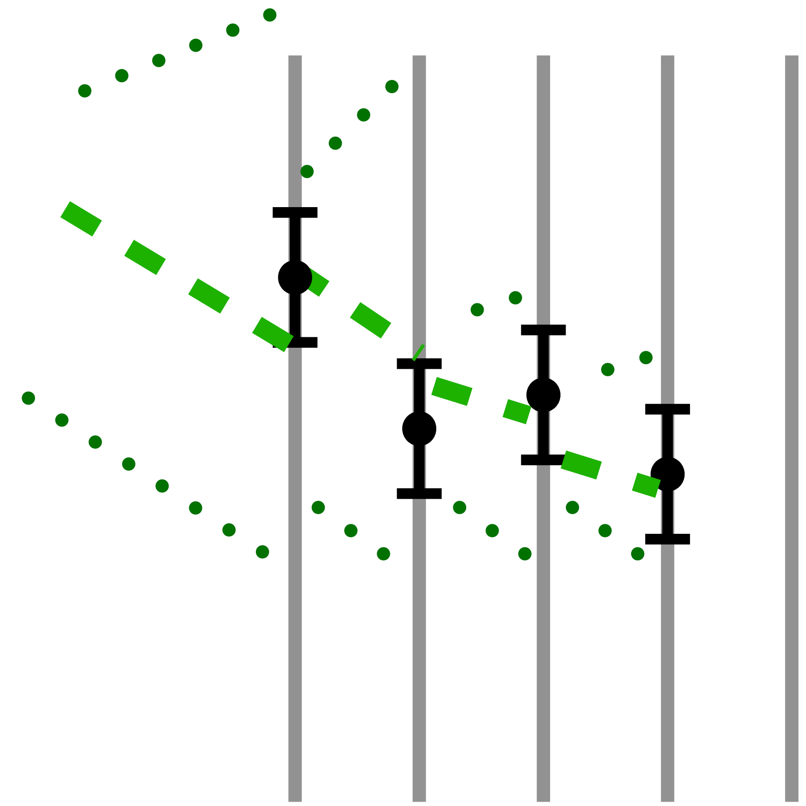


Kalman filter

Kalman formalism

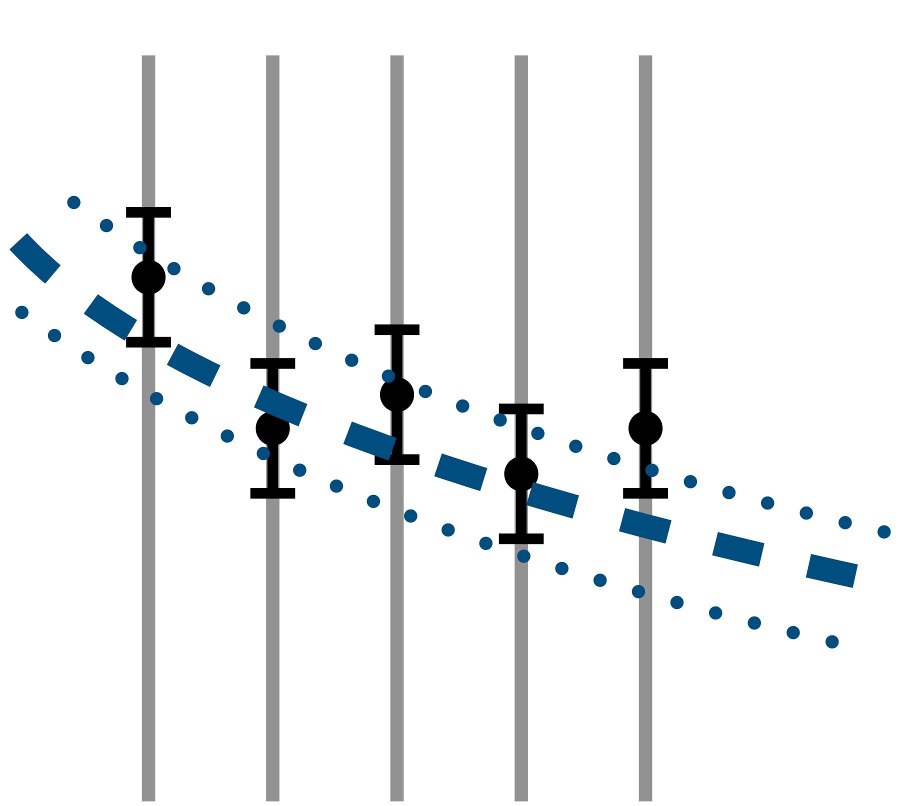


Global fit

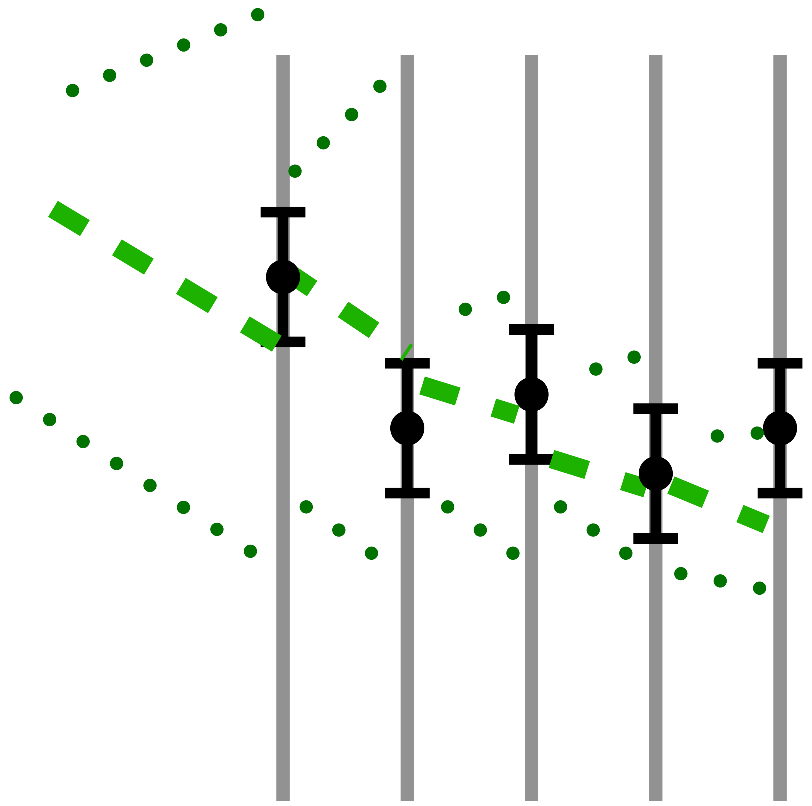


Kalman filter

Kalman formalism

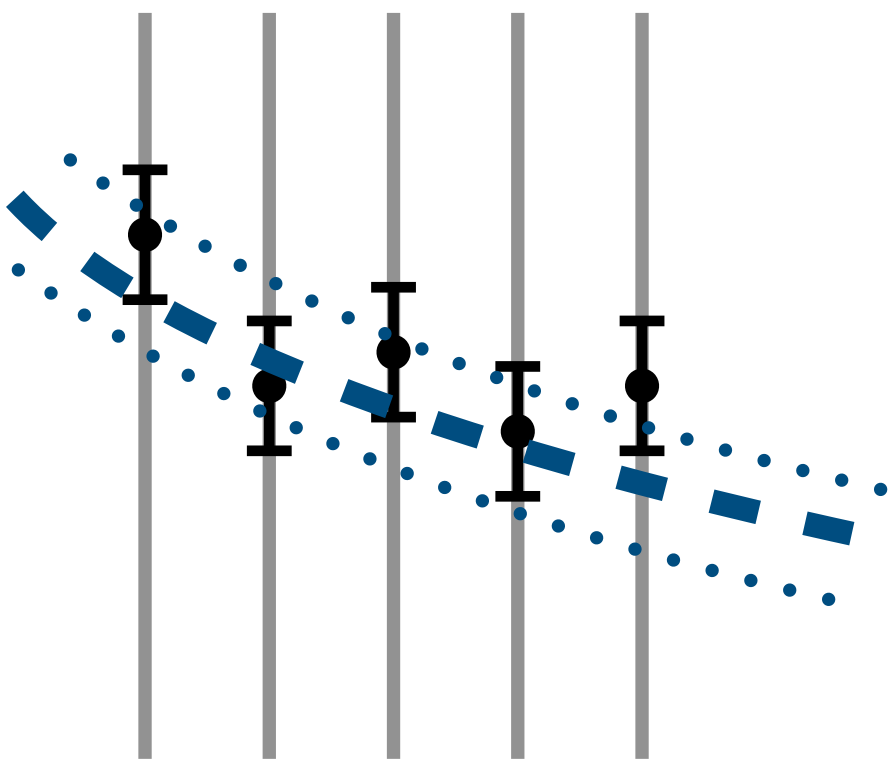


Global fit

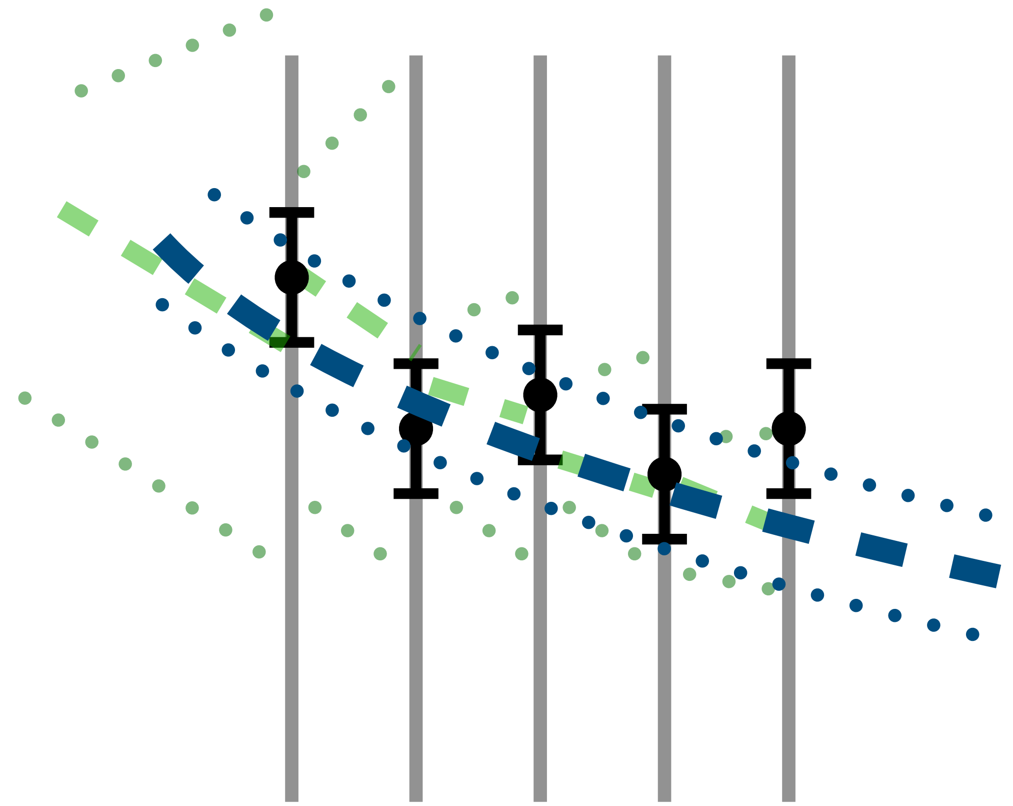


Kalman filter

Kalman formalism

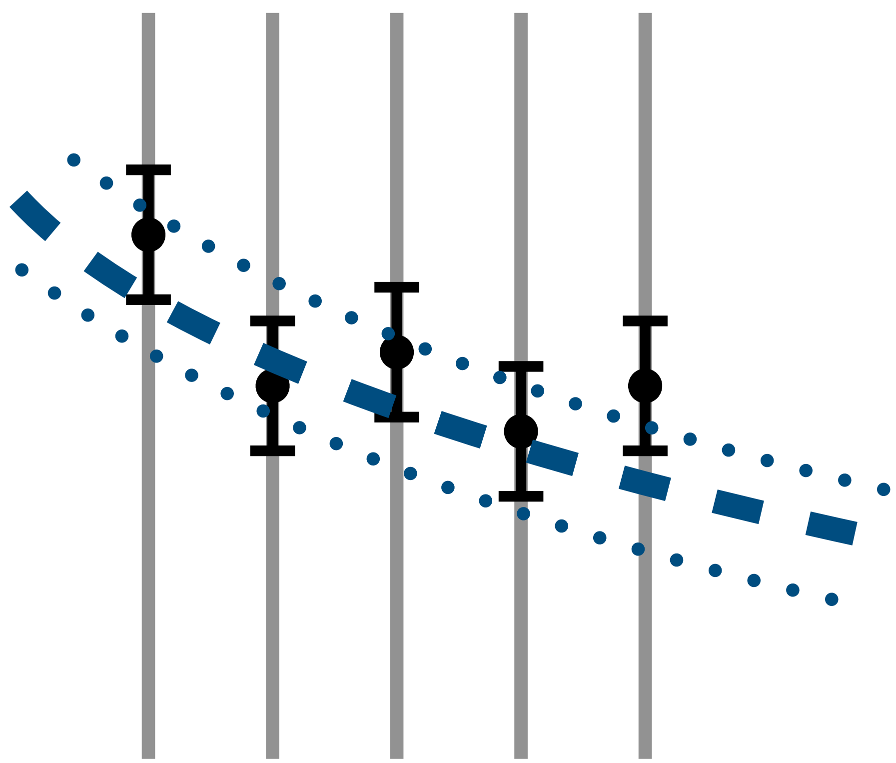


Global fit

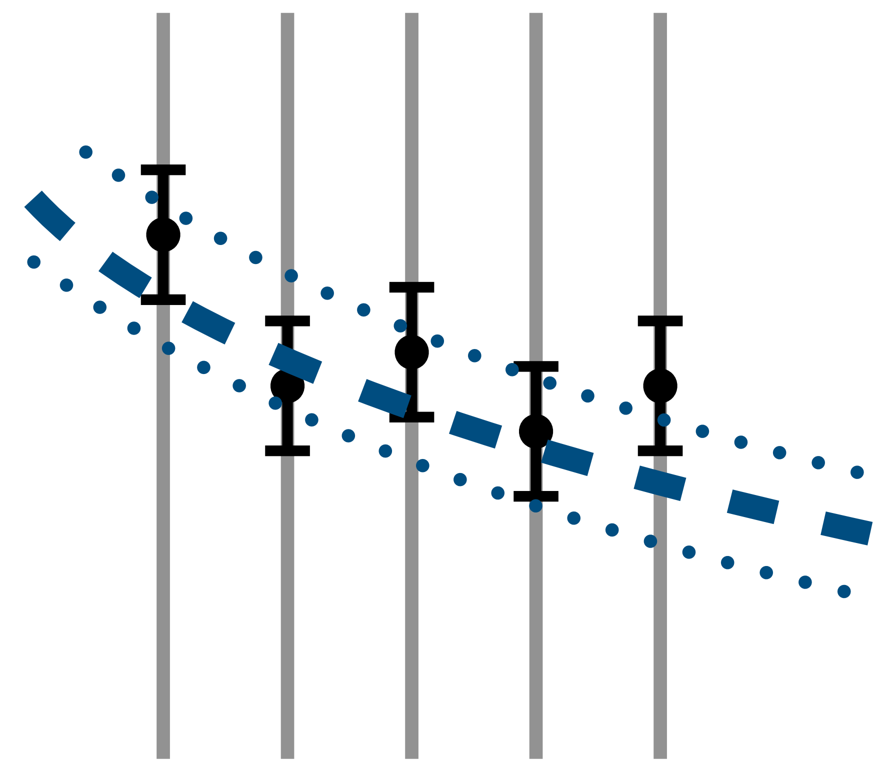


Kalman filter

Kalman formalism



Global fit



Kalman filter

1 computation of M-equation system
inversion of the $M \times M$ matrix

M computations of 1-equation system
inversion of the 1×1 matrix

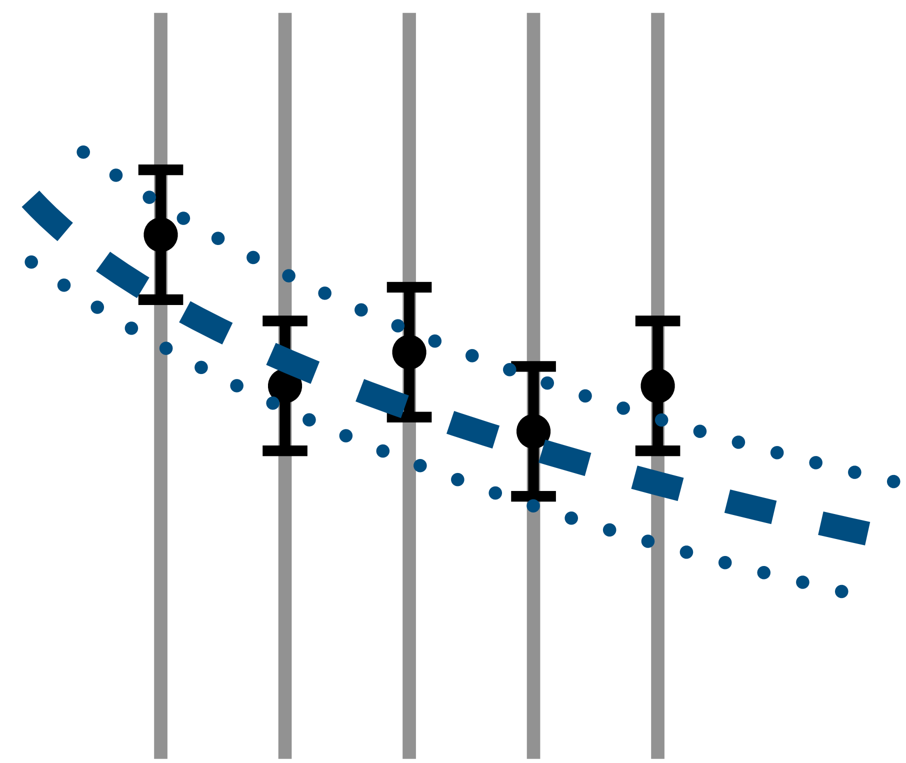
M - is number of measurement

Note: measurements might be not 1D

Kalman formalism

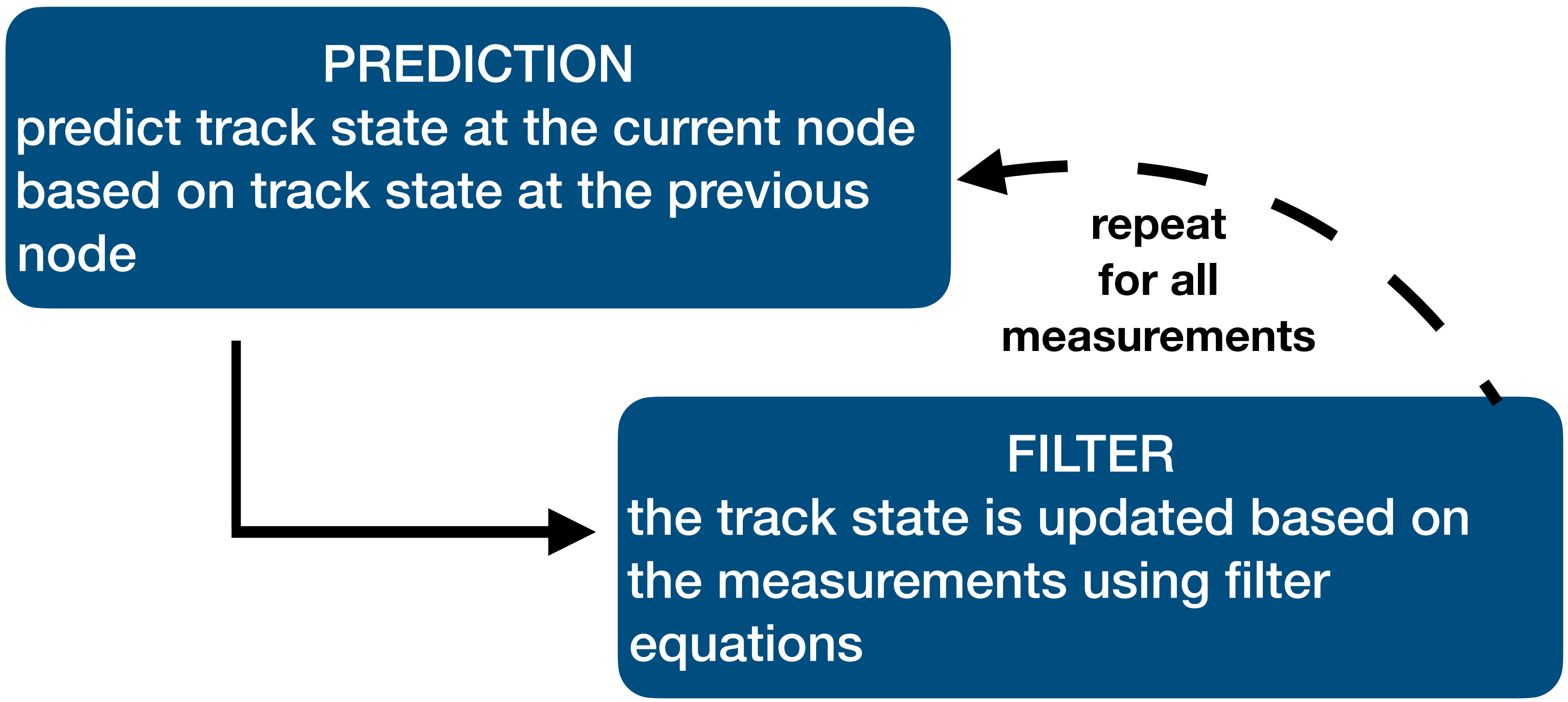
+ avoid inversion of $M \times M$ matrices
  easy to add noise and constraints

- validity of linear approximation
  only last state benefits from all information

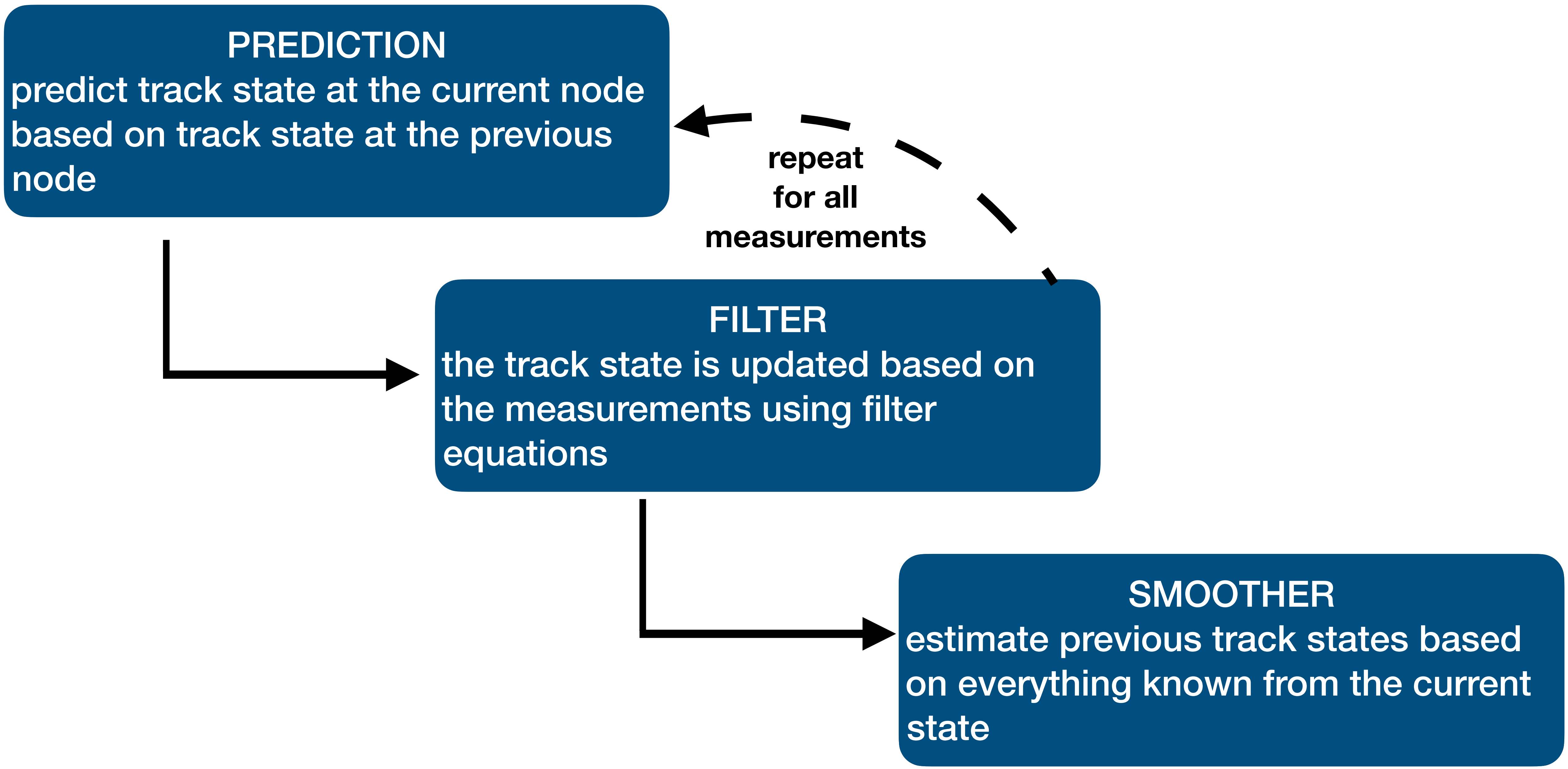


Kalman filter

Track fitter based on Kalman filter



Track fitter based on Kalman filter



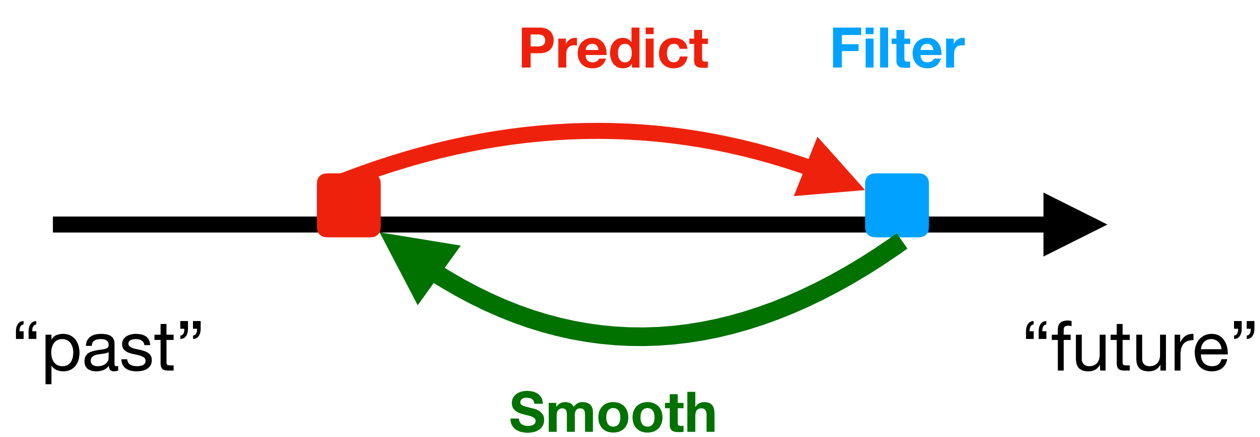
Track fitter based on Kalman filter

Time scale

Predict:
future state based on the current state

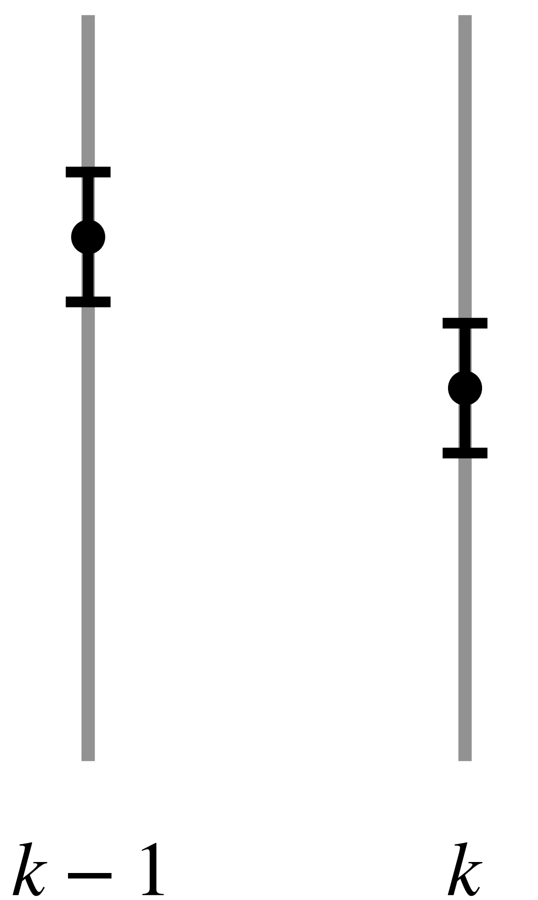
Filter:
current state based on the current and past measurements

Smoother:
past states based on all measurements up to now



Track fitter based on Kalman filter

Prediction



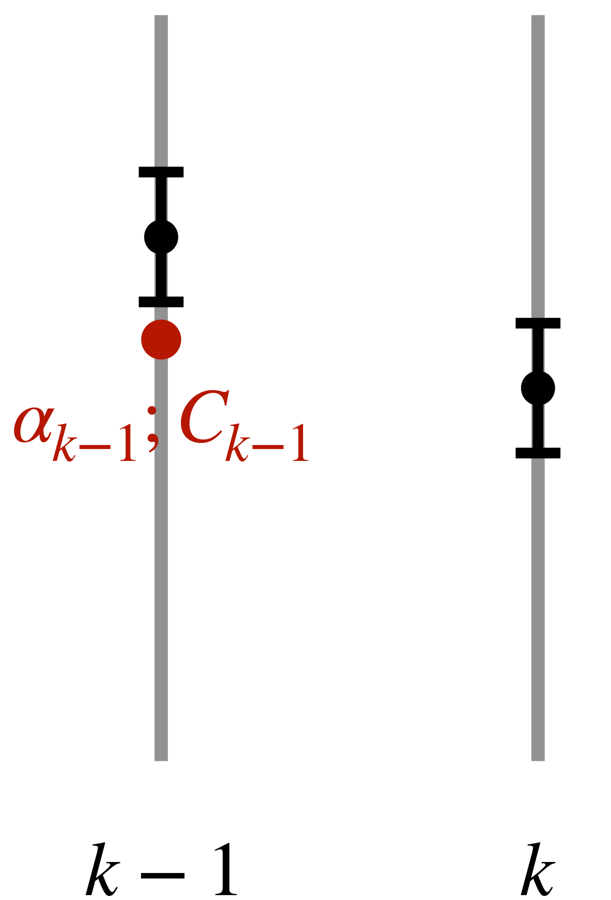
PREDICTION
predict track state at the current node based on track state at the previous node

α: track parameters

C: track parameters variance

Track fitter based on Kalman filter

Prediction



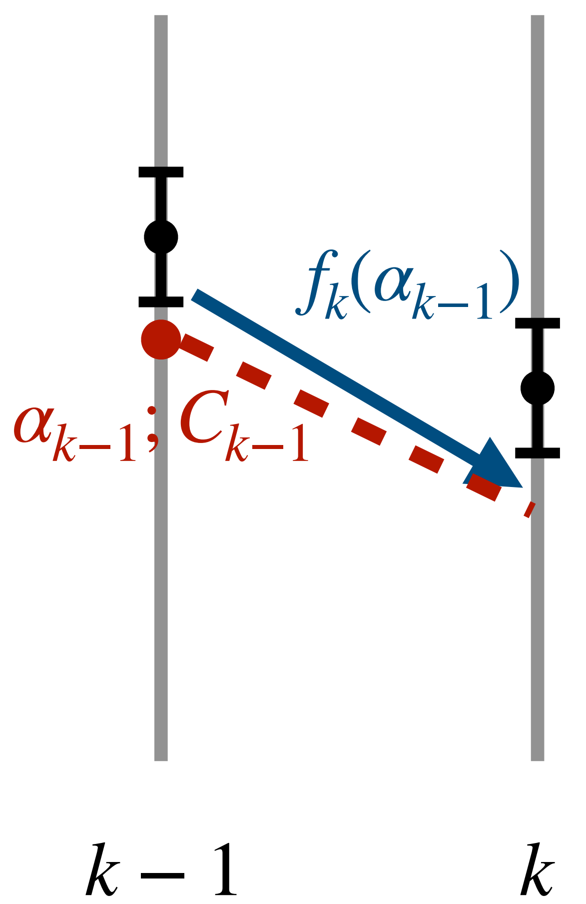
PREDICTION
predict track state at the current node based on track state at the previous node

α: track parameters

C: track parameters variance

Track fitter based on Kalman filter

Prediction



PREDICTION
predict track state at the current node based on track state at the previous node

Uses filtered state from the previous step of the filter

α: track parameters

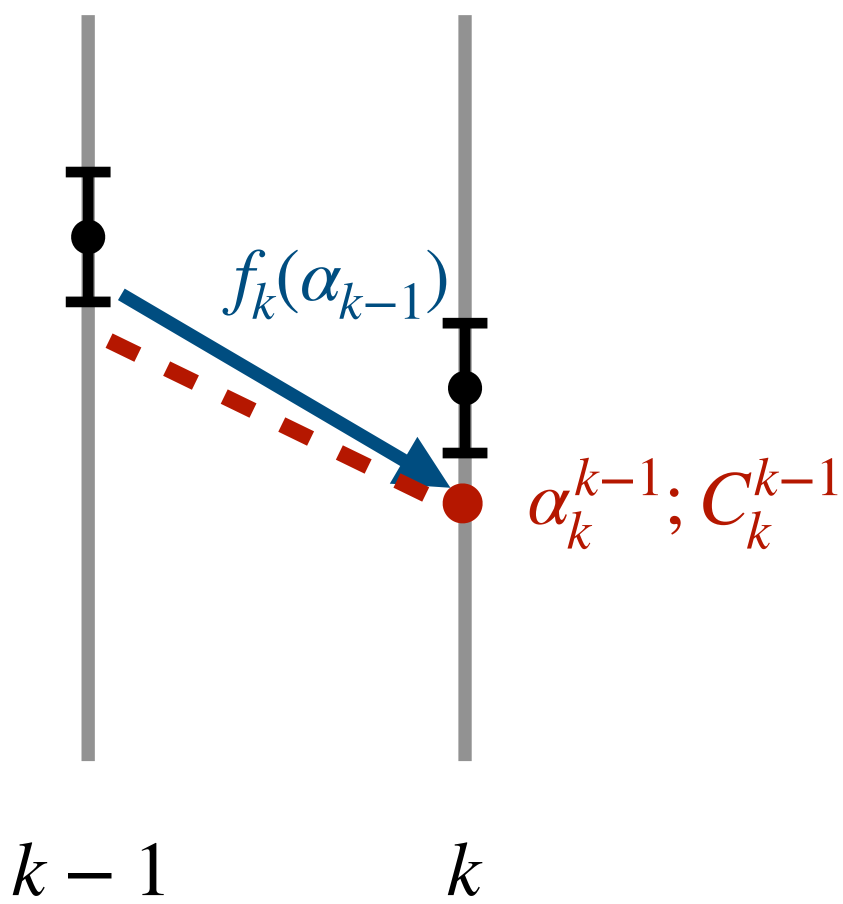
C: track parameters variance

Note: α is a vector

Track fitter based on Kalman filter

Prediction

PREDICTION
predict track state at the current node based on track state at the previous node



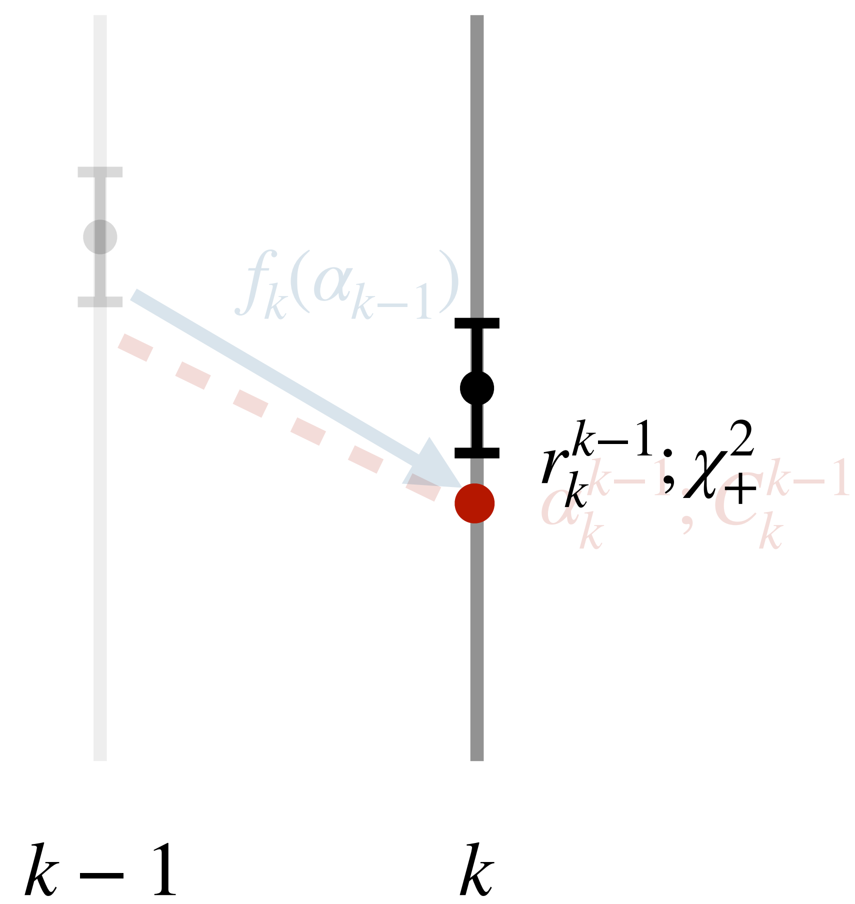
α: track parameters

C : track parameters variance

f : propagation function

Track fitter based on Kalman filter

Prediction



PREDICTION
predict track state at the current node based on track state at the previous node

Goal: minimize χ_+^2

$$r_k^{k-1} = m_k - h_k(\alpha_k^{k-1})$$

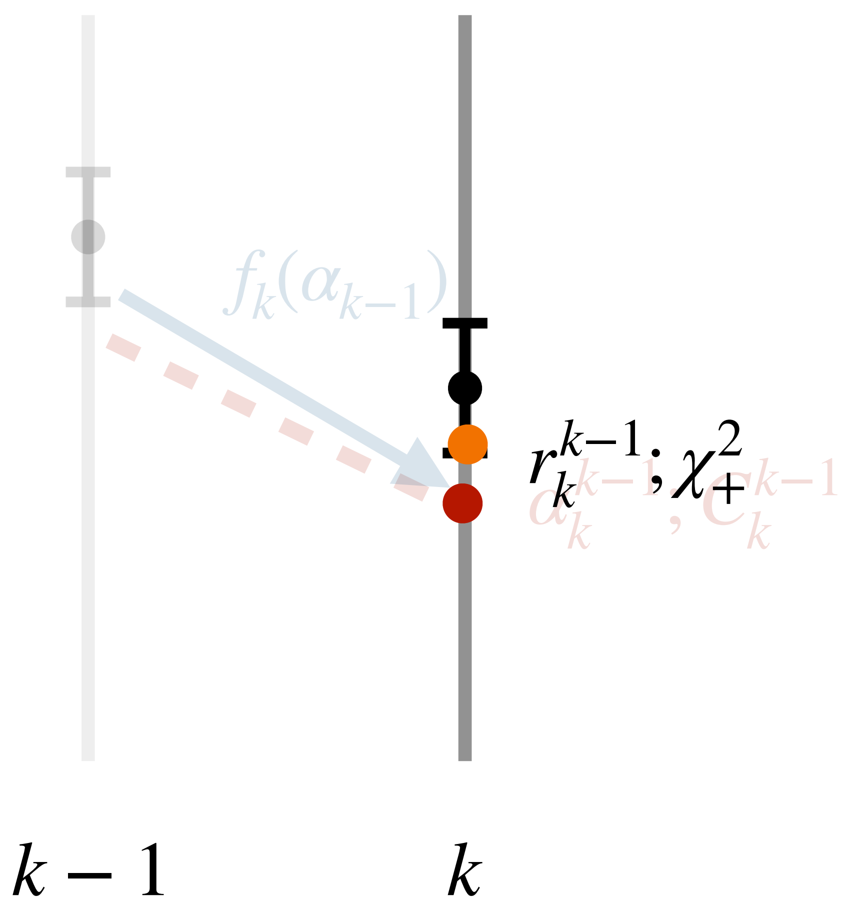
α: track parameters

m: measurement

h: projection function

Track fitter based on Kalman filter

Filter



FILTER
the track state is updated based on the measurements using filter equations

Goal: minimize χ_+^2

$$\alpha_k = \alpha_k^{k-1} + K_k r_k^{k-1}$$

$$C_k = (1 - \boxed{K_k} H_k) C_k^{k-1}$$

gain matrix

α: track parameters

C: track parameters variance

Track fitter based on Kalman filter

Gain matrix K_k

FILTER
the track state is updated based on the measurements using filter equations

Gain matrix tells us how much should our prediction change if adding the information about the measurement
aka the weight of the prediction versus measurement

Very precise measurements:

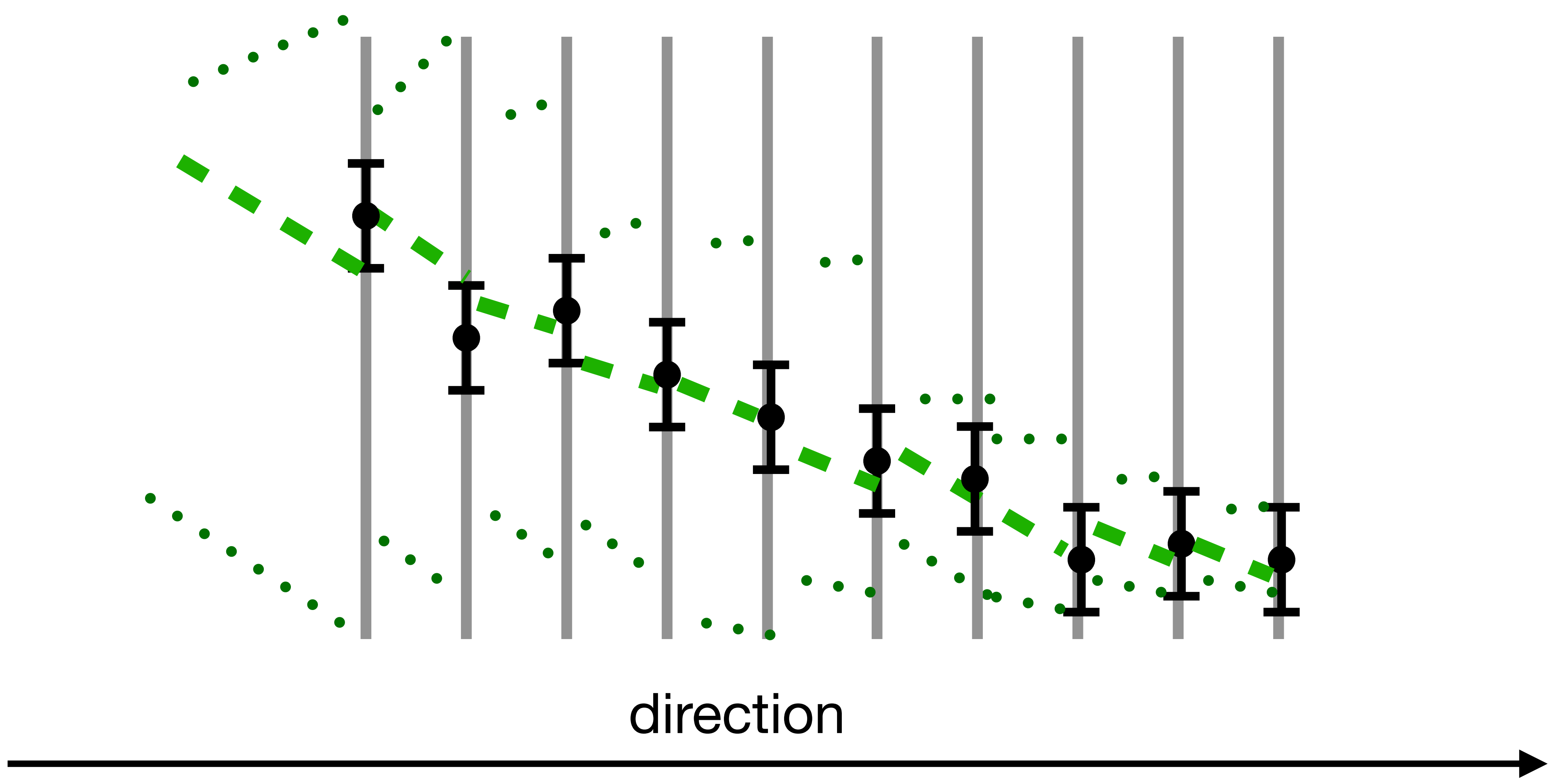
$$V_k \downarrow \Rightarrow K_k \uparrow$$

Very precise prediction:

$$C_k^{k-1} \downarrow \Rightarrow K_k \downarrow$$

*if $dim(\alpha)$ is small you might use the weighted mean formalism instead of gain matrix formalism (faster): more in R.Fruwirth A. of K.F. to T. and V.F.

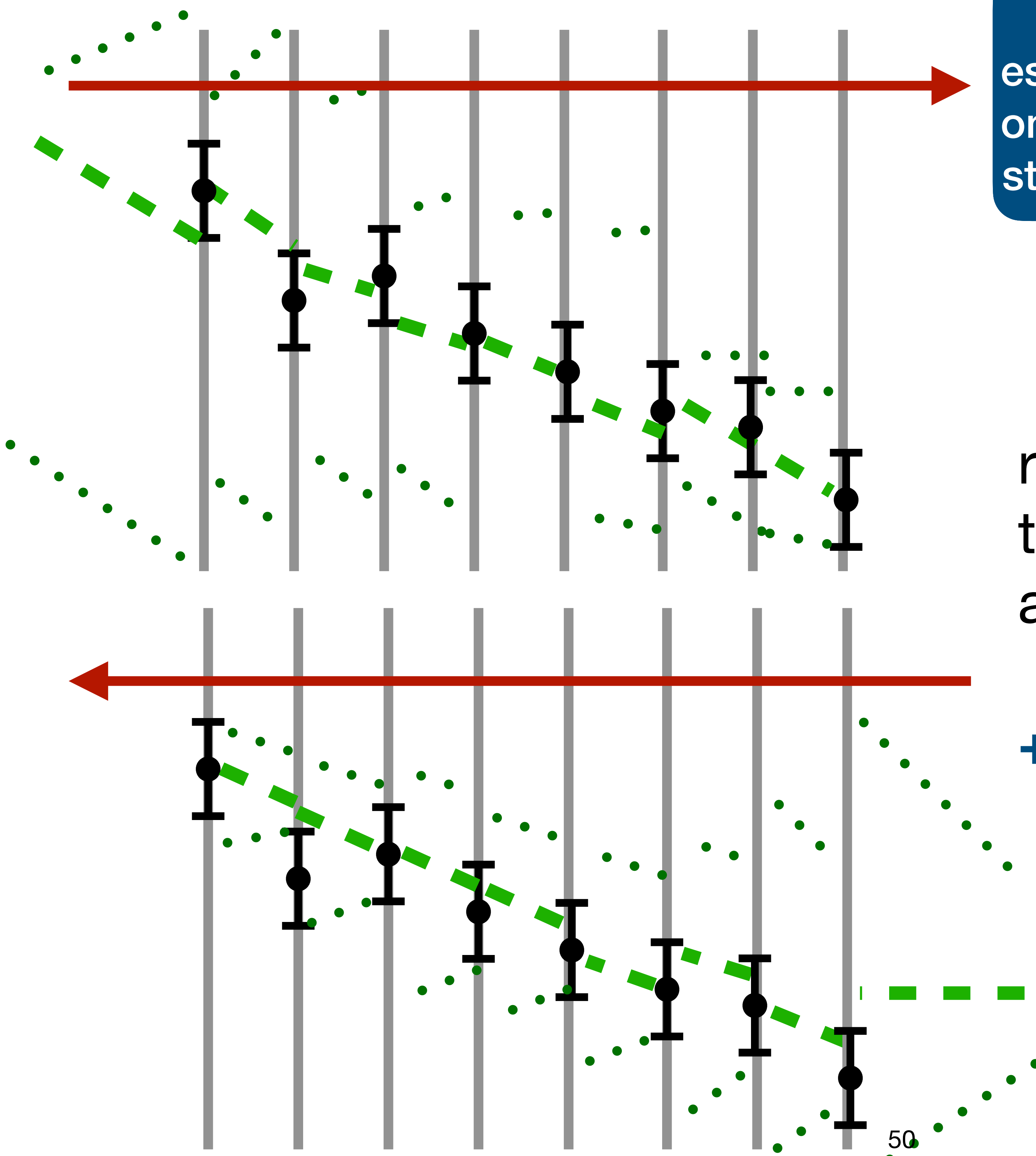
Track fitter based on Kalman filter



- there are no global parameters in Kalman filter
- the best track estimate is the last point

Track fitter based on Kalman filter

Smoother



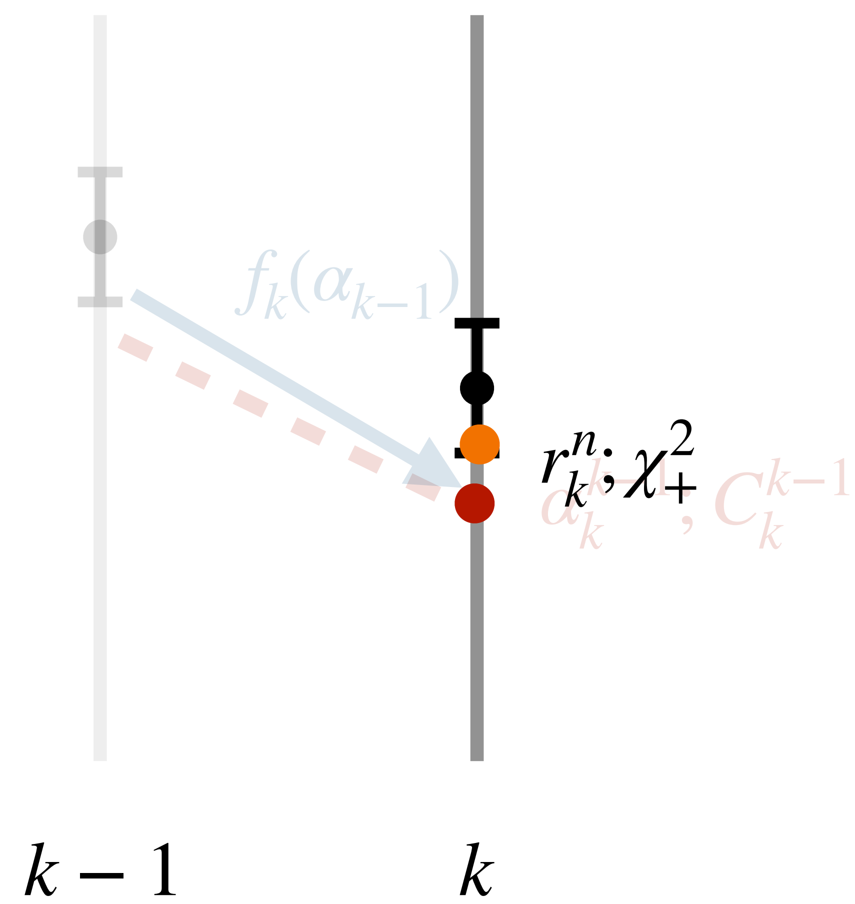
SMOOTHER
estimate precious track states based on everything known from the current state

run Kalman filter in reverse and take a weighted average at each plane

+ simpler math

Track fitter based on Kalman filter

Smoother



SMOOTHER
estimate precious track states based on everything known from the current state

alternative smoothing:

Goal: minimize χ_+^2

$$\alpha_{k-1}^n = \alpha_{k-1} + \boxed{A_{k-1}}(\alpha_k^n - \alpha_k^{k-1})$$

smoother gain matrix

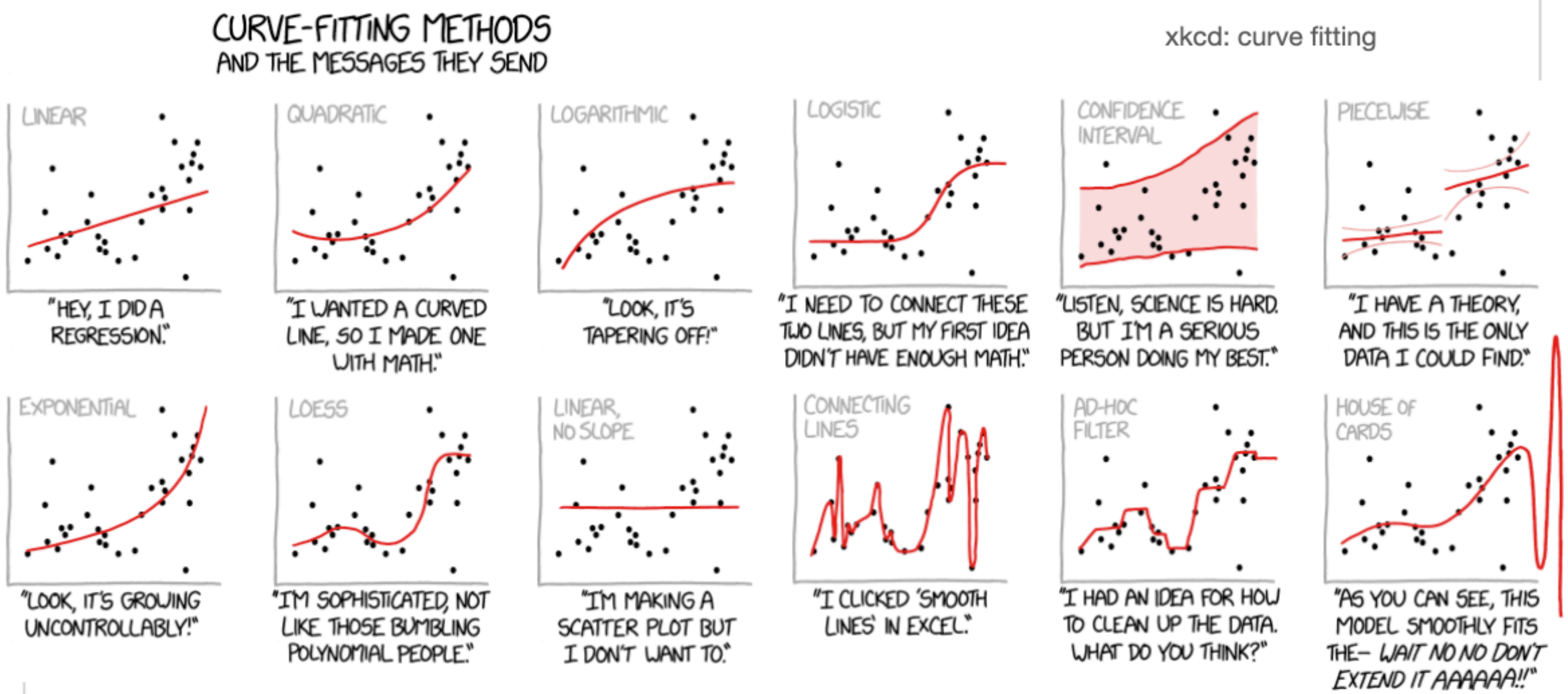
weight of the prediction with information of all measurement versus current state

α: track parameters

What is the track* we draw?

Reminder: fitting

xkcd: curve fitting



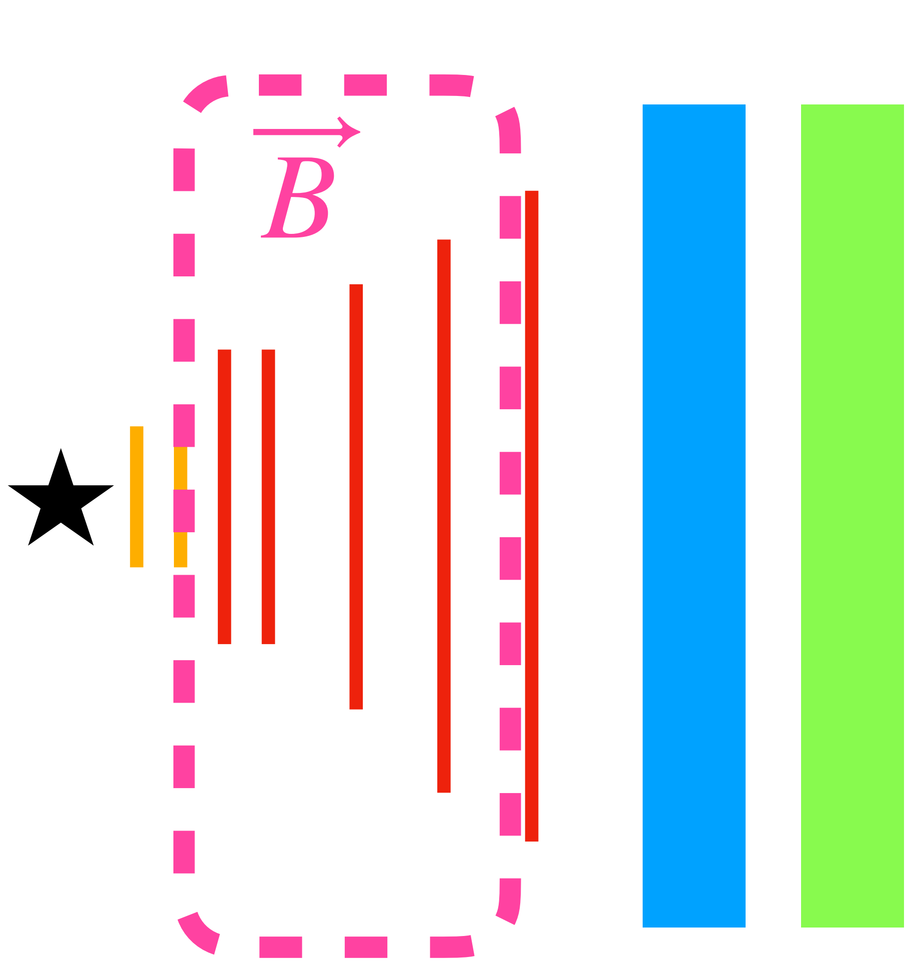
two simple facts to spell out:
- choosing the model is purely **subjective**
- if model **"fits"** it does **not** mean it is **"true"**

*What is α?

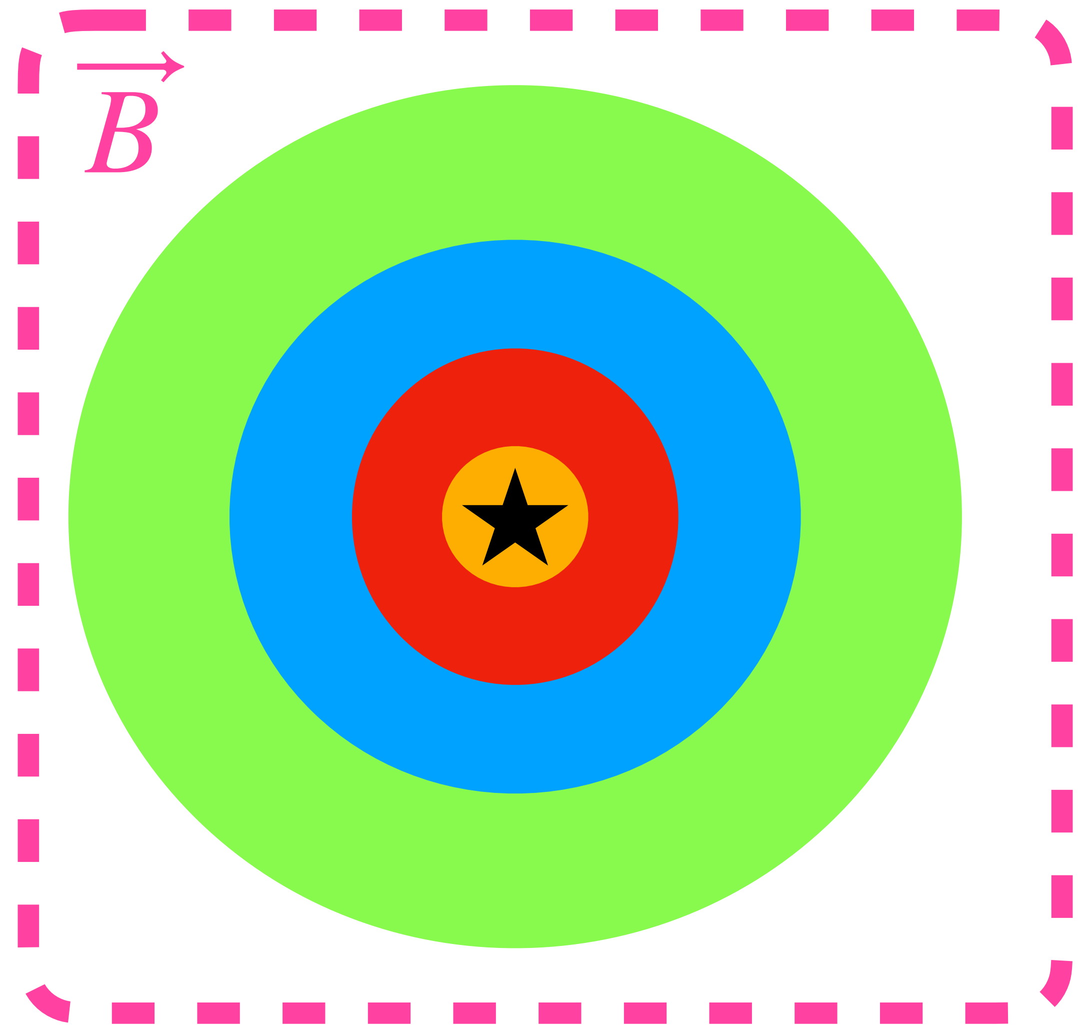
Track models

Detector geometry

- Forward



- Collider



Vertex detector

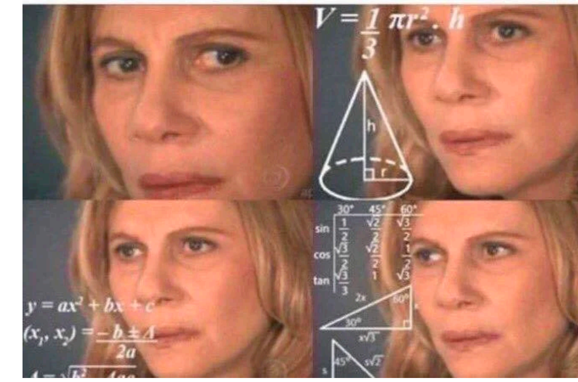
Tracking detector

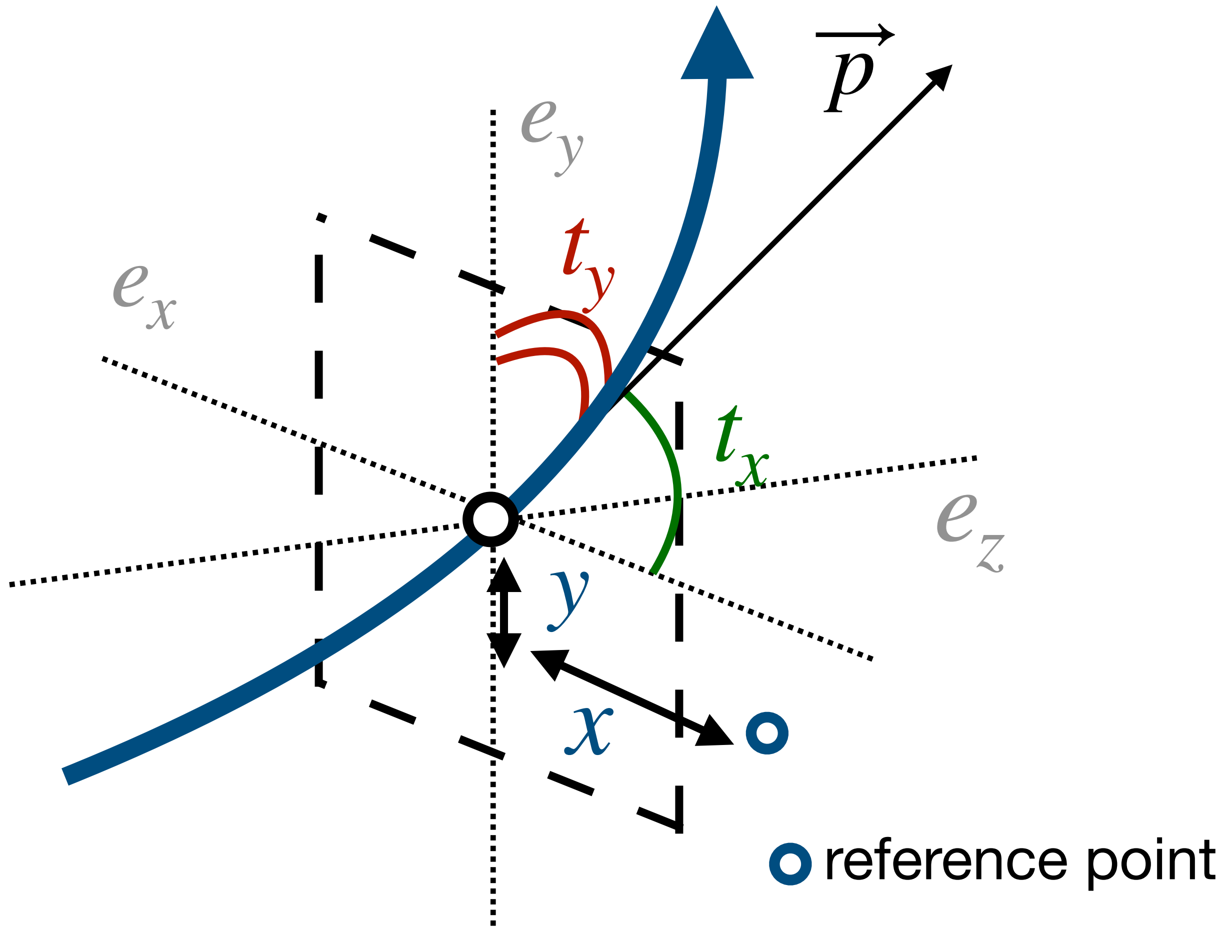
Calorimeter

Muon chambers

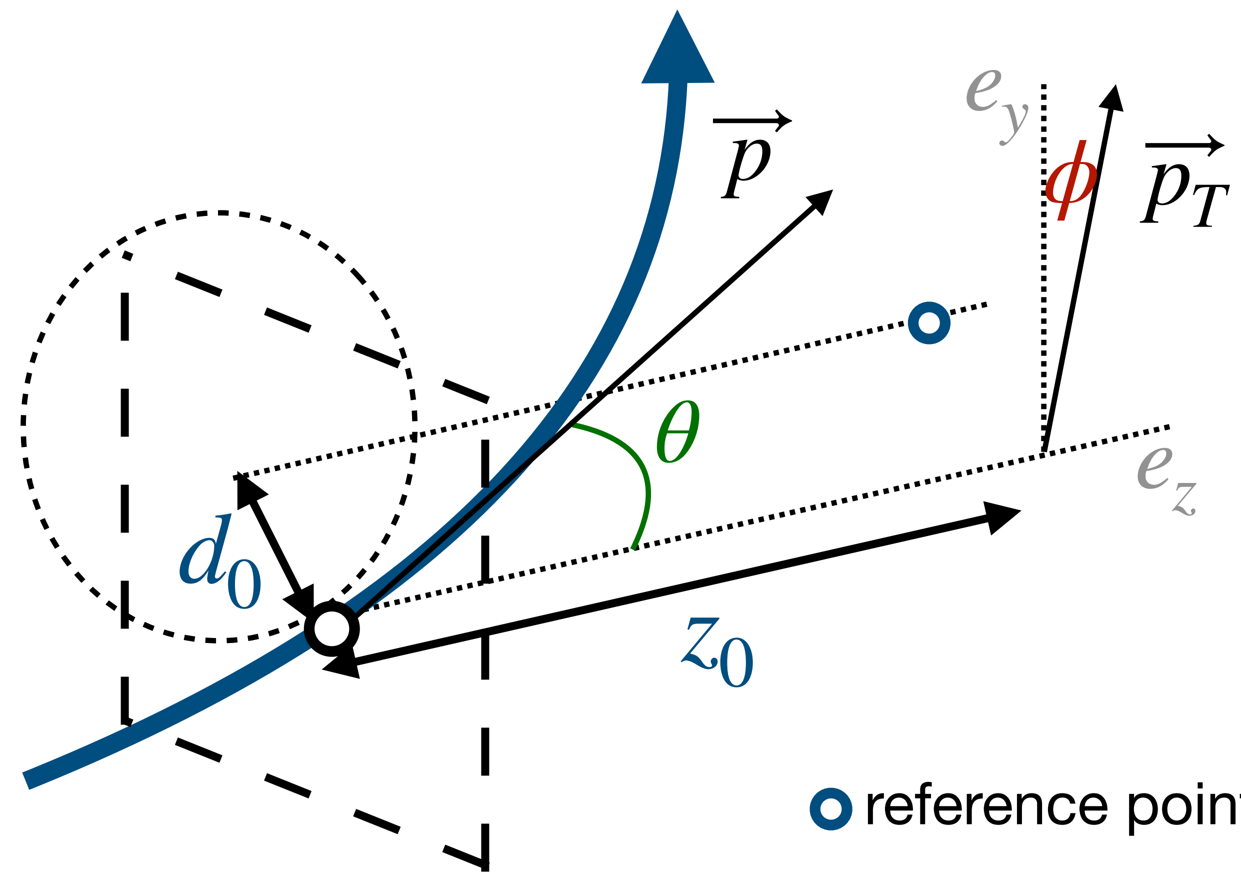
We skip here particle identification detectors, like RICH

Track models





$$\alpha = \begin{pmatrix} x \\ y \\ t_x \\ t_y \\ q/p \end{pmatrix} \begin{array}{l} \text{- coordinate} \\ \text{- coordinate} \\ \text{- slope in x} \\ \text{- slope in y} \\ \text{- charge/momentum} \end{array}$$



$$\alpha = \begin{pmatrix} d_0 \\ z_0 \\ \phi \\ \theta \\ q/p \end{pmatrix} \begin{array}{l} \text{- } \perp \text{ impact param.} \\ \text{- } \| \text{ impact param.} \\ \text{- azimuth} \\ \text{- polar} \\ \text{- charge/momentum} \end{array}$$

How good is my fitter?

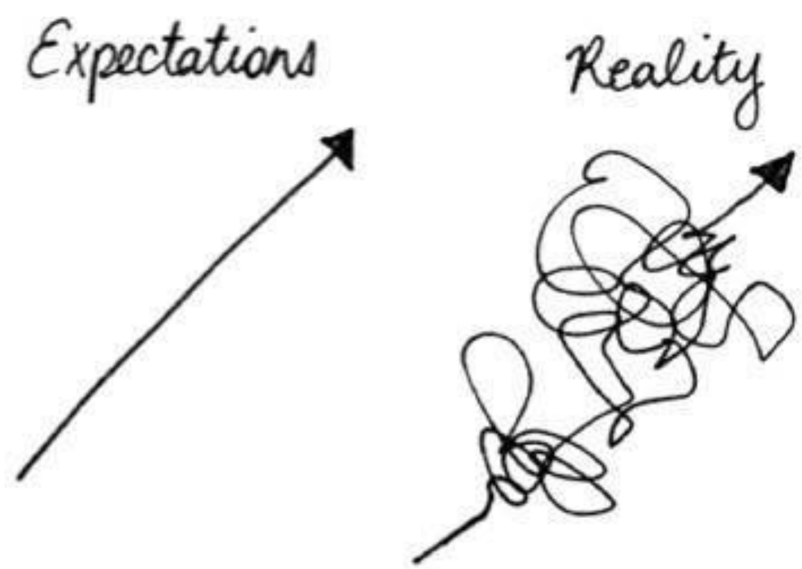
A check list

1. Check pulls of the input data: $p = \dfrac{x_i - x_{true}}{\sigma_i}$

2. Check pull of the track parameters if you know “true” ones

3. Check hit residuals : $\hat{r}_i = m_i - h_i(\hat{\alpha});\ \ var(r) = V - HCH^T$ and residual pull : $p = \dfrac{r_i}{\sqrt{var(r_i)}} \propto G(0,1)$

Track parameters correlate residuals!


Expectations
Reality

Reality

- There is magnetic field/non-linear propagation
- There is noise
- There is energy loss
- Residual is not a point-to-point residual, based on the measurement technique and detector design
- One fit is often not good enough

Will not cover details

Reality Solution

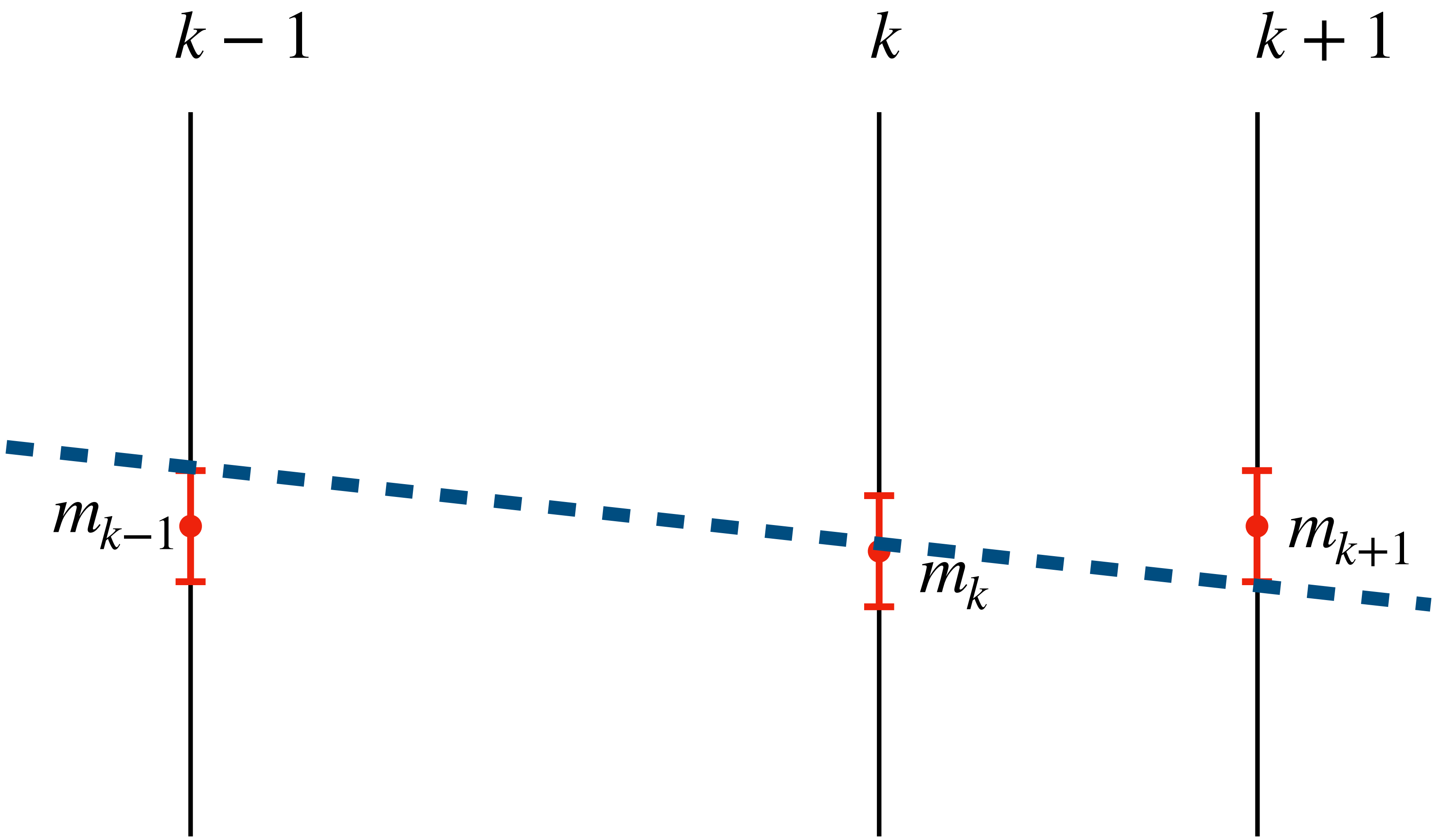
- There is magnetic field/non-linear propagation **- Taylor series**
- There is noise **- inflate prediction uncertainty**
- There is energy loss **- inflate prediction uncertainty**
- Residual is not a point-to-point residual, based on the measurement technique and detector design **- correct projection**
- One fit is often not good enough - **multiple iterations**

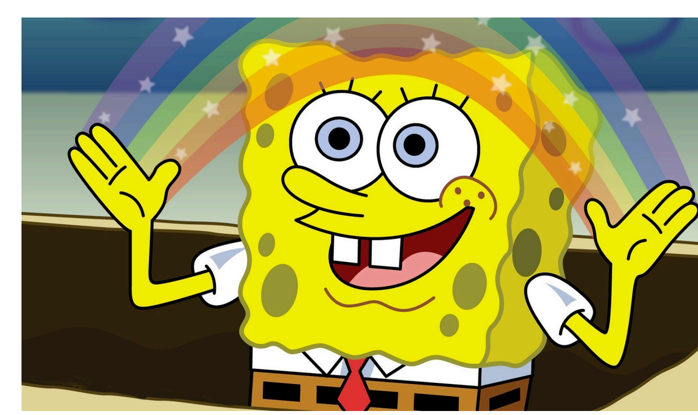
Will not cover details

back on track…

“second lecture”

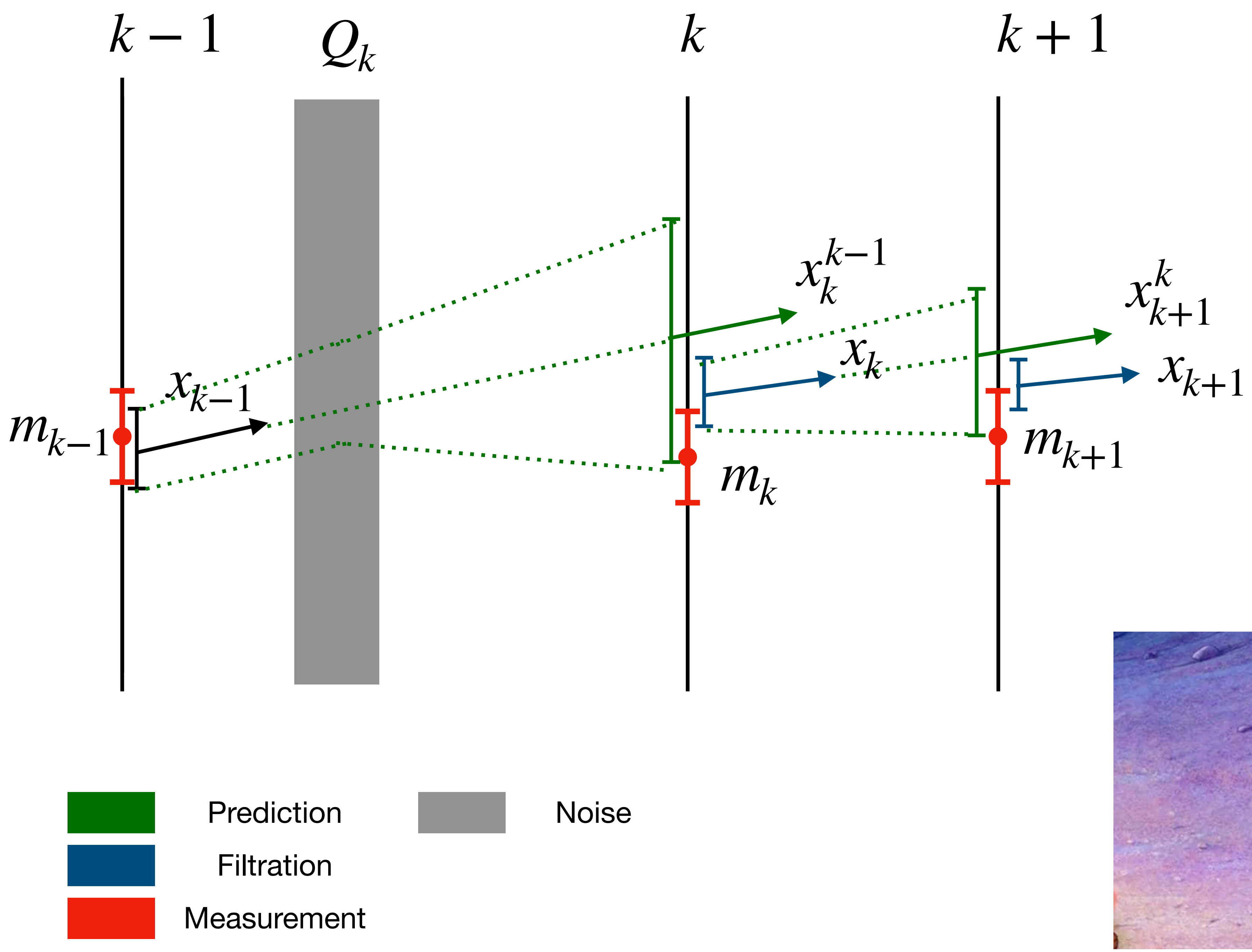
Started with

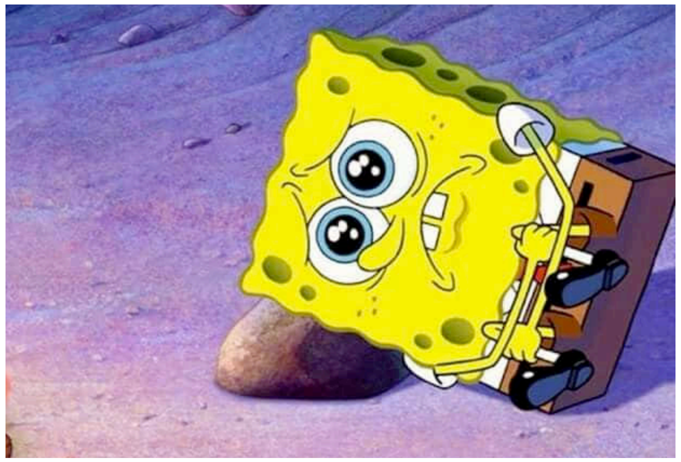




Ended up with

Theoretically Kalman filter is an ideal fitter!





Kalman filter problems

Practical implementation

Reality bites back

1. $K_k \propto \left(R_k^{k-1}\right)^{-1}$: matrix inversion is expensive - $\mathcal{O}(N^3)$ and unstable

2. Kalman filter is prone to numerical instabilities

3. Kalman filter is prone to outlier biases

4. Kalman filter assumes linearity and Gaussian errors

5. Kalman filter uses given assumption on the "zero"-state (initialisation)

Problem 1: Matrix inversion is still a pain

$$K_k = C_k^{k-1} H_k^T (R_k^{k-1})^{-1} \text{ and } \chi^2 \propto R_k^{-1}$$

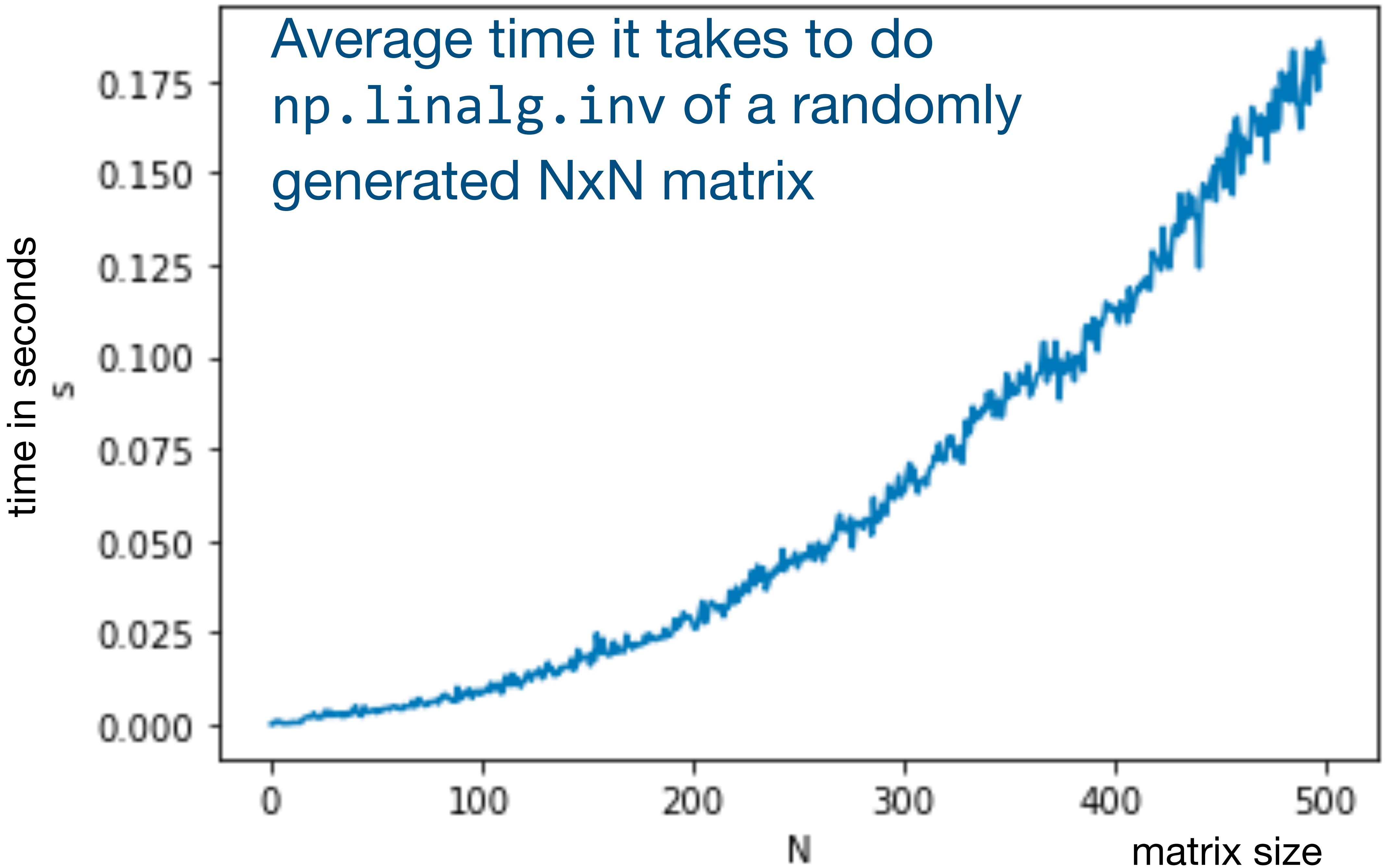
1. computationally expensive : $\mathcal{O}(N^3)$ 😱

2. numerically unstable: prone to **rounding errors** - especially bad for <u>ill-conditioned</u> matrices

For a typical tracking problem $N = 5$

K_K - gain matrix
C_k^{k-1} - predicted variance matrix after propagation
H_k- projected propagation state derivative matrix
R_k^{k-1} - residuals variance

Problem 1: Matrix inversion is still a pain



Problem 1: Matrix inversion is a pain

Computational costs

- Still a fascinating problem for mathematics (even if the results might be of questionable practical use):

Matrix inversion	One $n \times n$ matrix	One $n \times n$ matrix	Gauss–Jordan elimination	$O(n^3)$
			Strassen algorithm	$O(n^{2.807})$
			Coppersmith–Winograd algorithm	$O(n^{2.376})$
			Optimized CW-like algorithms	$O(n^{2.373})$

wikipedia

General truth: avoid inverting matrix at all costs

See this great post

Problem 2: Numerical stability of Kalman filter

Part 1 : Catastrophic cancelation

$$C_k = (1 - K_k H_k) C_k^{k-1}$$

- $K_k H_k \approx 1 \Rightarrow$ **"catastrophic cancellation"**
- Often appears in **the beginning of filter** or when you encounter **the first measurement that matters**

Catastrophic cancellation example

$a = 5.34587; b = 5.34585$

Exact:

$a^2 - b^2 = 28.5783260569 - 28.5781122225 = 0.0002138344$

Rounding:

$a^2 - b^2 = 28.57833 - 28.57811 = 0.00022$

float: 7 decimal digits

double : 15 decimal digits

Problem 2: Numerical stability of Kalman filter

Part 1 : Catastrophic cancelation

$$C_k = (1 - K_k H_k) C_k^{k-1}$$

- $K_k H_k \approx 1 \Rightarrow$ **“catastrophic cancellation”**

Worse iteratively: example* evaluating $sin^2(\frac{\pi}{4})$

$0.5 - 2^{-53}$ (columns 2–3) $0.5 - 2^{-52}$ (columns 4–5)

iteration	χ^2	$\Delta\chi^2$	χ^2	$\Delta\chi^2$
1	2189.6608022288765	2189.6608022288765	2189.6609616291444	2189.6609616291444
2	2205.8925415651697	16.2317393363932131	2205.7475274721887	16.086565843044355
3	2204.1624495187029	1.7300920464667797	2203.8047379025147	1.9427895696740052
4	2203.3737431764907	0.78870634221220826	2204.010212472562	0.20547457004795433
5	2203.7285494796347	0.35480630314395967	2203.94089768533	0.069314787229814101
6	2203.812855194968	0.084305715333812259	-	-

*from S.Ponce

Problem 2: Numerical stability of Kalman filter

Part 2: Stability under $K + \delta K$

$$C_k = (1 - K_k H_k) C_k^{k-1}$$

- C_k must be **posdef** matrix - otherwise not a valid covariance matrix
- C_k can become negdef
- And convergence for fitter is harder
- Many attempts to solve numerical instabilities: choosing correct constraints on the errors (especially first state errors), robust extended Kalman filter [G.A. Einicke, L.B. White], square root filter [P. Kaminski, A. Bryson, S.Schmidt]

Problem 2: Numerical stability of Kalman filter

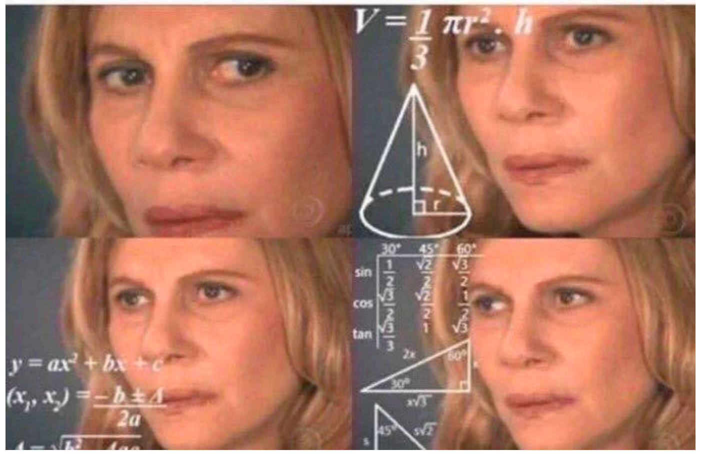
Part 2 : Stability under $K + \delta K$

$$C_k = (1 - K_k H_k) C_k^{k-1}$$

$$K \to K + \delta K$$

$$C_k^{new} = (1 - (K_k + \delta K) H_k) C_k^{k-1} = C_k - \delta K H_k C_k^{k-1}$$

Small deviations in gain matrix K might lead to `negdef` C_k

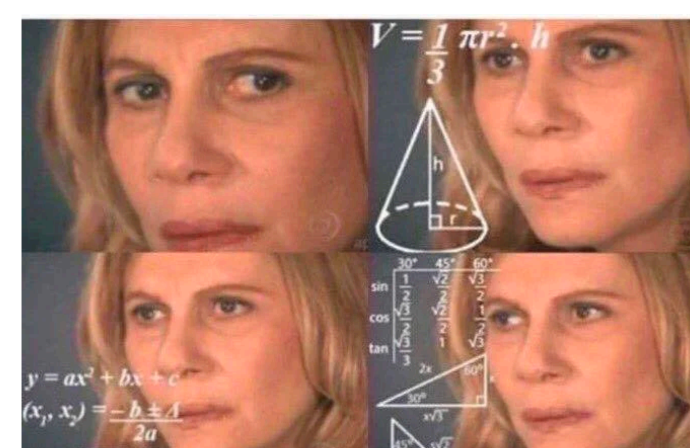


Problem 2: Numerical stability of Kalman filter

Part 2 : Stability under $K + \delta K$

$$C_k = (1 - K_k H_k) C_k^{k-1}$$
$$C_k = (1 - K_k H_k) C_k^{k-1} (1 - K_k H_k)^T + K_k V_k K_k^T$$

This is just an error propagation of $\alpha_k = \alpha_k^{k-1} + K_k r_k^{k-1}$



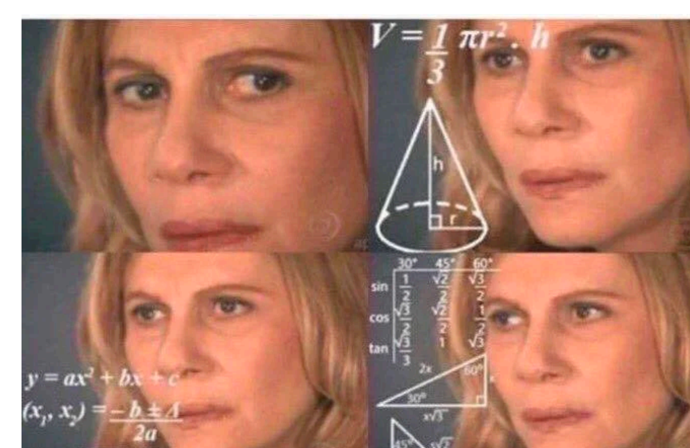
Problem 2: Numerical stability of Kalman filter

Part 2 : Stability under $K + \delta K$

$$C_k = (1 - K_k H_k) C_k^{k-1}$$

$$C_k = (1 - K_k H_k) C_k^{k-1} (1 - K_k H_k)^T + K_k V_k K_k^T$$

unstable stable against $K + \delta K$



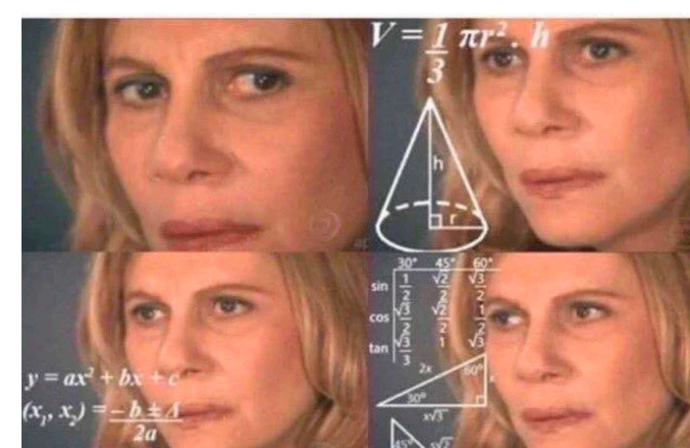
Problem 2: Numerical stability of Kalman filter

Part 2 : Stability under $K + \delta K$

$$C_k = (1 - K_k H_k) C_k^{k-1} \qquad N^3 + \mathcal{O}(N^2)$$

$$C_k = (1 - K_k H_k) C_k^{k-1} (1 - K_k H_k)^T + K_k V_k K_k^T \qquad 3N^3 + \mathcal{O}(N^2)$$

unstable — stable against $K + \delta K$



Problem 2: Numerical stability of Kalman filter

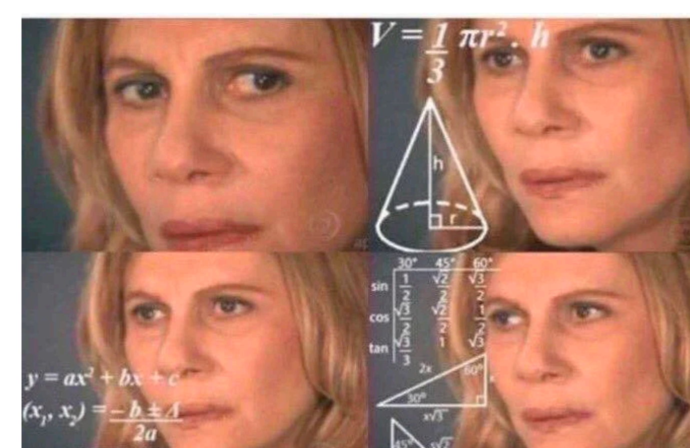
Part 2 : Stability under $K + \delta K$

$$C_k = (1 - K_k H_k) C_k^{k-1} \qquad N^3 + \mathcal{O}(N^2)$$

$$C_k = (1 - K_k H_k) C_k^{k-1} (1 - K_k H_k)^T + K_k V_k K_k^T \qquad 3N^3 + \mathcal{O}(N^2)$$

$$C_k = C_k^{k-1} - K_k \left(2 H_k C_{k-1} - (V_k + H_k C_k^{k-1} H_k^T) K_k^T \right) \qquad N^3 + \mathcal{O}(N^2)$$

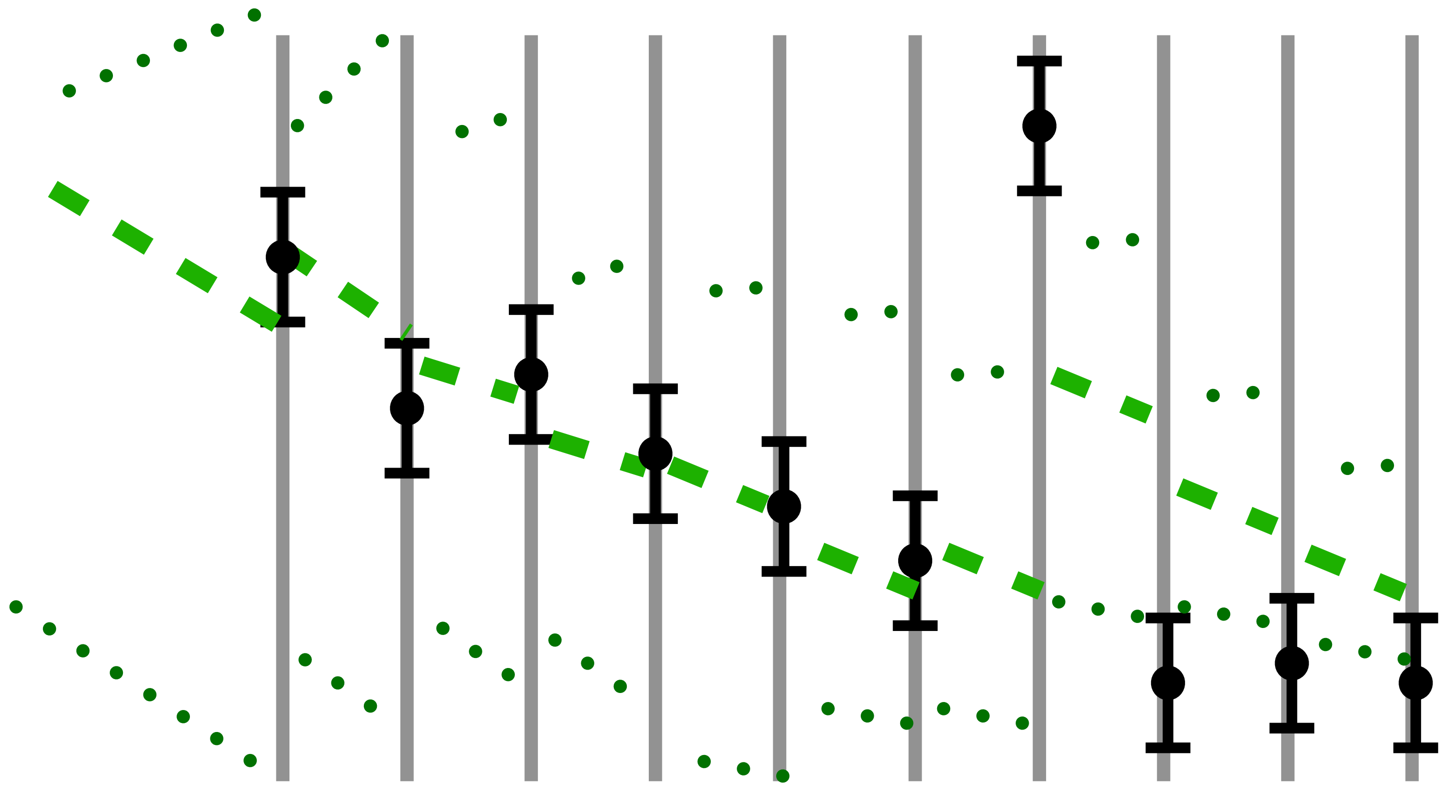
unstable   stable against $K + \delta K$



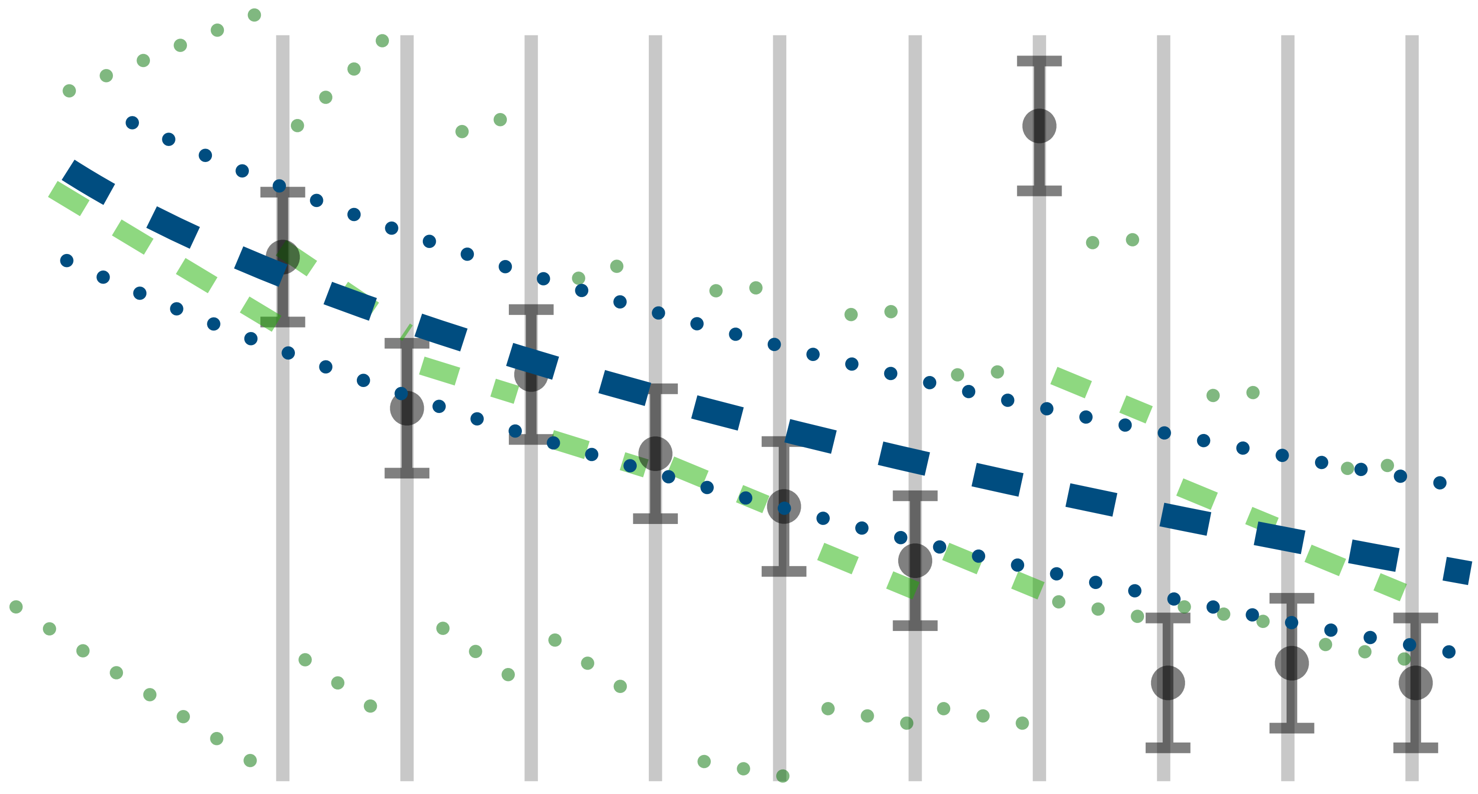
Problem 3: Outliers

- Pattern recognition can make mistakes - it is a wrong track
- It can be an unfortunate event - see δ-rays for example
- Can be electronics noise (if it is often the case in LHCb VELO, you can safely blame me)

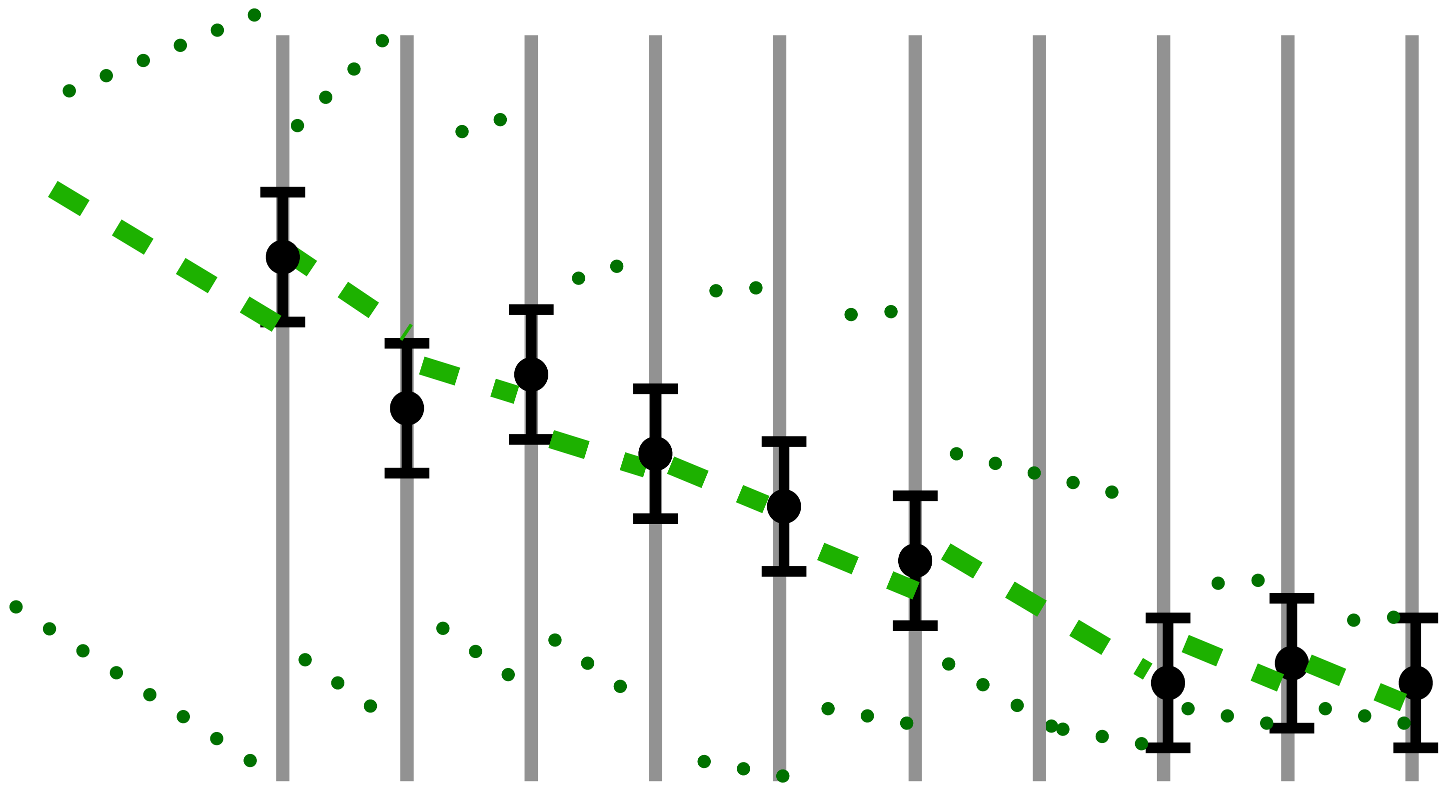
Problem 3: Outliers



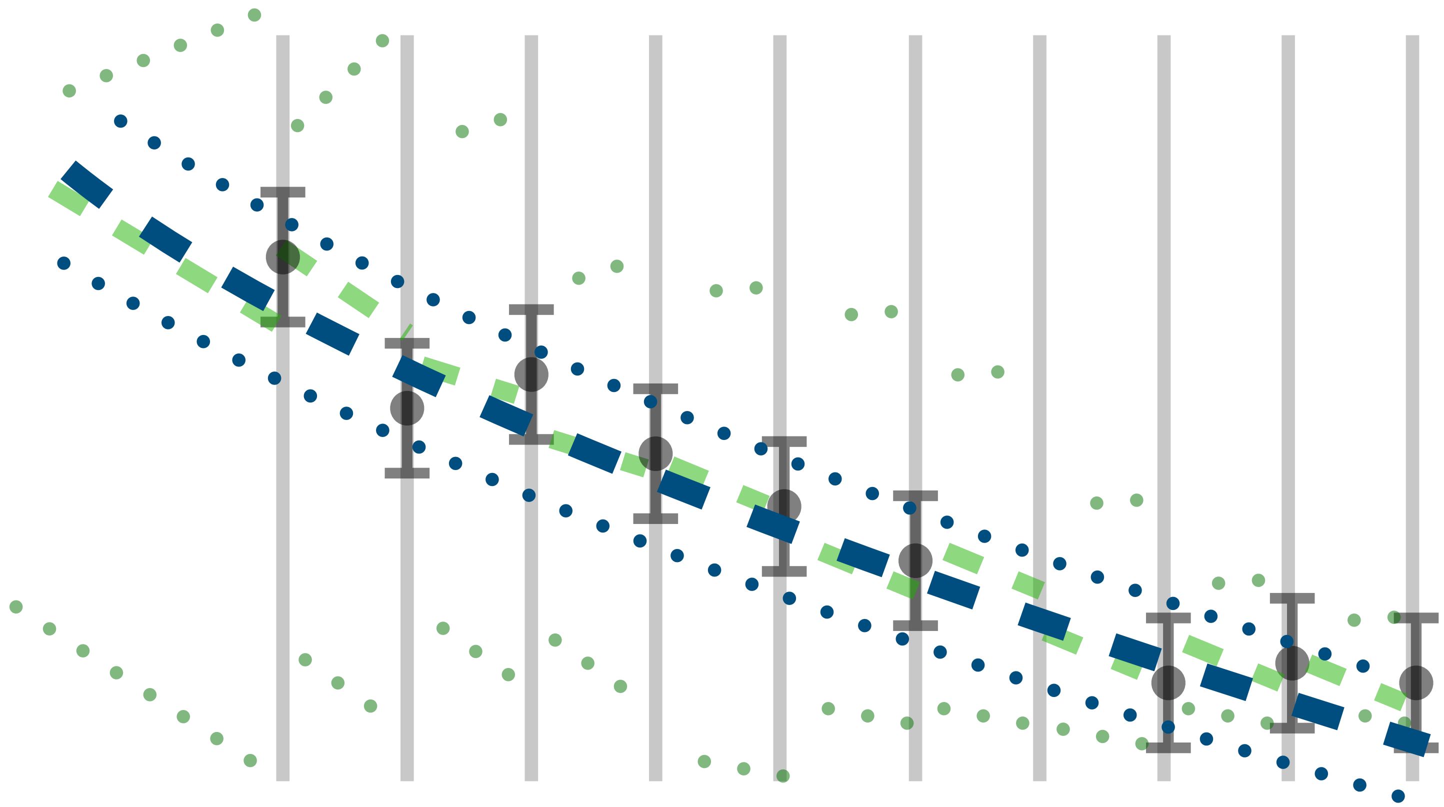
Problem 3: Outliers



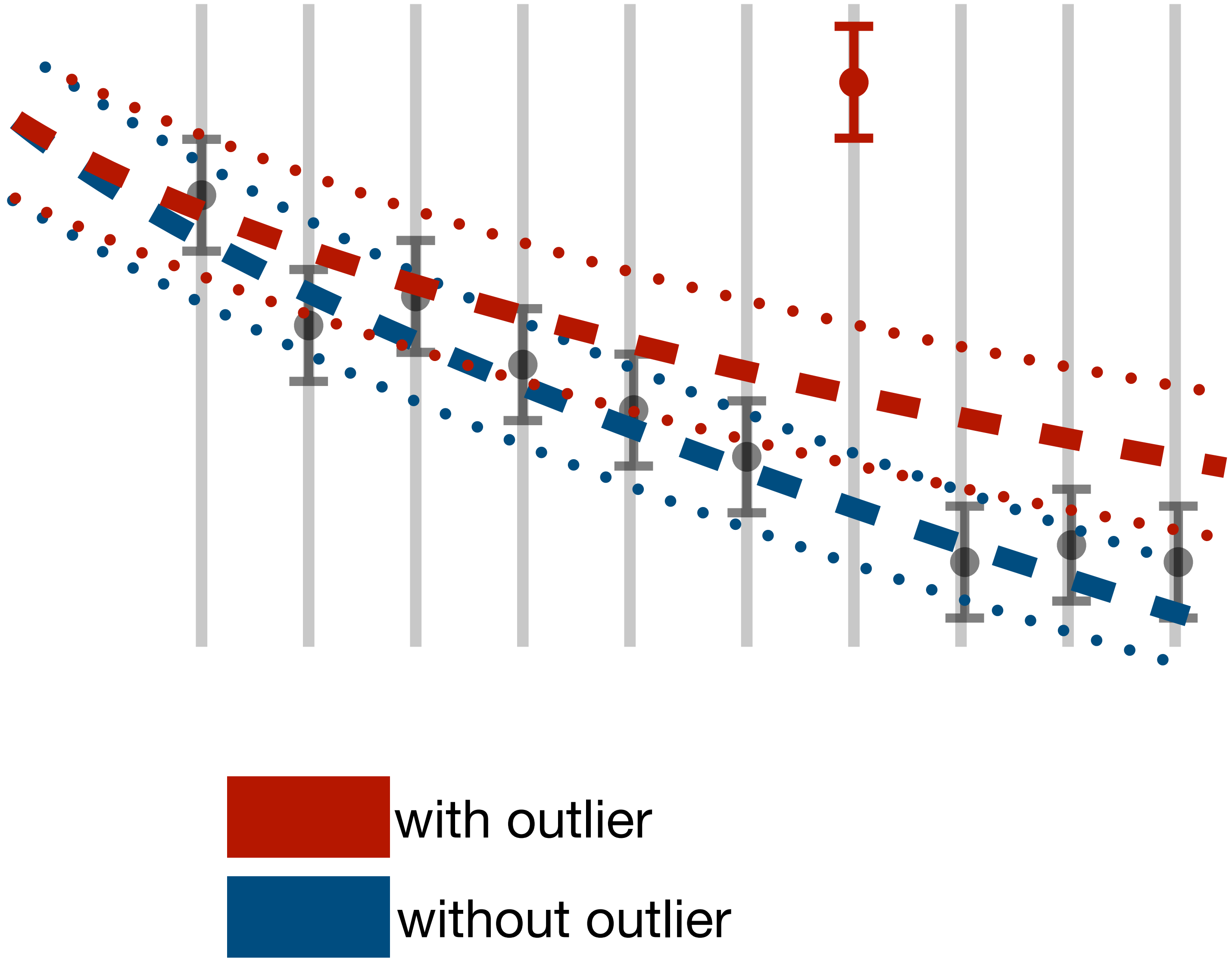
Problem 3: Outliers



Problem 3: Outliers



Problem 3: Outliers



Problem 3: Outliers

Solution: χ^2_+

- If χ^2_+ is too big, reject the measurement

$\chi^2_{+,smoother}$ $\chi^2_{+,filter}$

Which χ^2_+ is better to use?

Problem 3: Outliers

Solution: χ^2_+

- If χ^2_+ is too big, reject the measurement

$\chi^2_{+,smoother}$

refit, high precision

$\chi^2_{+,filter}$

on-flight,
low precision,
first measurement might be
an outlier

Can you already spot a problem with any outlier removal?

Problem 3: Outliers

Outlier removal increases hit purity but decreases hit efficiency

Hit purity: fraction of correct hits per track
Hit efficiency: fraction of all correct hits found

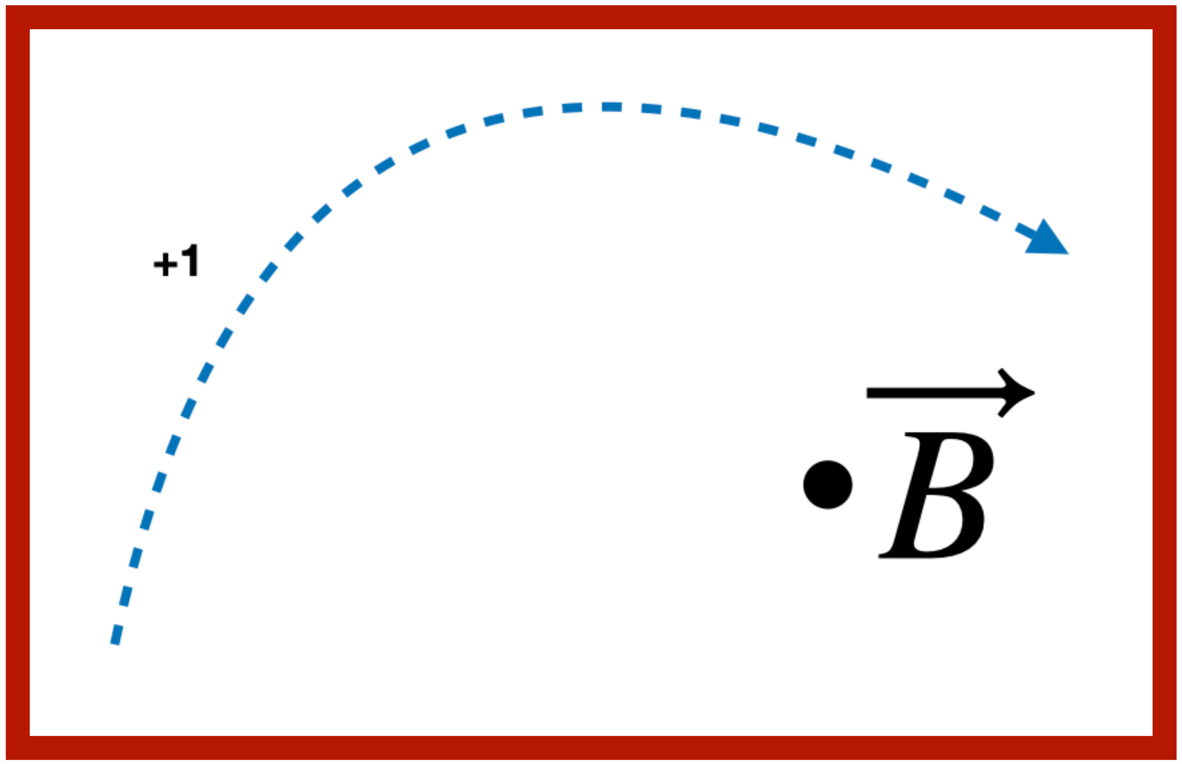
This is by no means the entire story - there are advanced outlier removal techniques and outlier-robust Kalman filters [E. Chabanat, N. Estre], [G.Agamennoni, J.I. Nieto, E.M. Nebot]

Problem 4: Kalman filter assumptions

Part 1: linear propagation

- Basic assumption of Kalman filter - **linear model for propagation,** but

reality



Taylor expansion around reference state

Problem 4: Kalman filter assumptions

Part 2: non-Gaussian errors

- Another assumption of Kalman filter - **Gaussian errors**
- In reality:
    - Non-gausian noise
    - Non-gaussian energy loss
    - Non-gaussian scattering

Especially important for electrons in material heavy detectors, like ATLAS or CMS

Problem 4: Kalman filter assumptions

Part 2: non-Gaussian errors - Gaussian Sum Filter

- Replace non-gaussian effect by a weighted sum of gaussians - **gaussian sum filter**

filtered state: $G(\alpha_k, C_k) \Rightarrow \sum_{i=0}^{L} b_i G(\alpha_k^i, C_k^i)$

b_L - weights, L - number of the Gaussian components

Problem 4: Kalman filter assumptions

Part 2: non-Gaussian errors - Gaussian Sum Filter

- Replace non-gaussian effect by a weighted sum of gaussians - **gaussian sum filter**

filtered state: $G(\alpha_k, C_k) \Rightarrow \sum_{i=0}^{L} b_i G(\alpha_k^i, C_k^i)$

b_L - weights, L - number of the Gaussian components

Problem:

- L filtrated states per measurement - computations complexity increases as L^k

Solutions:

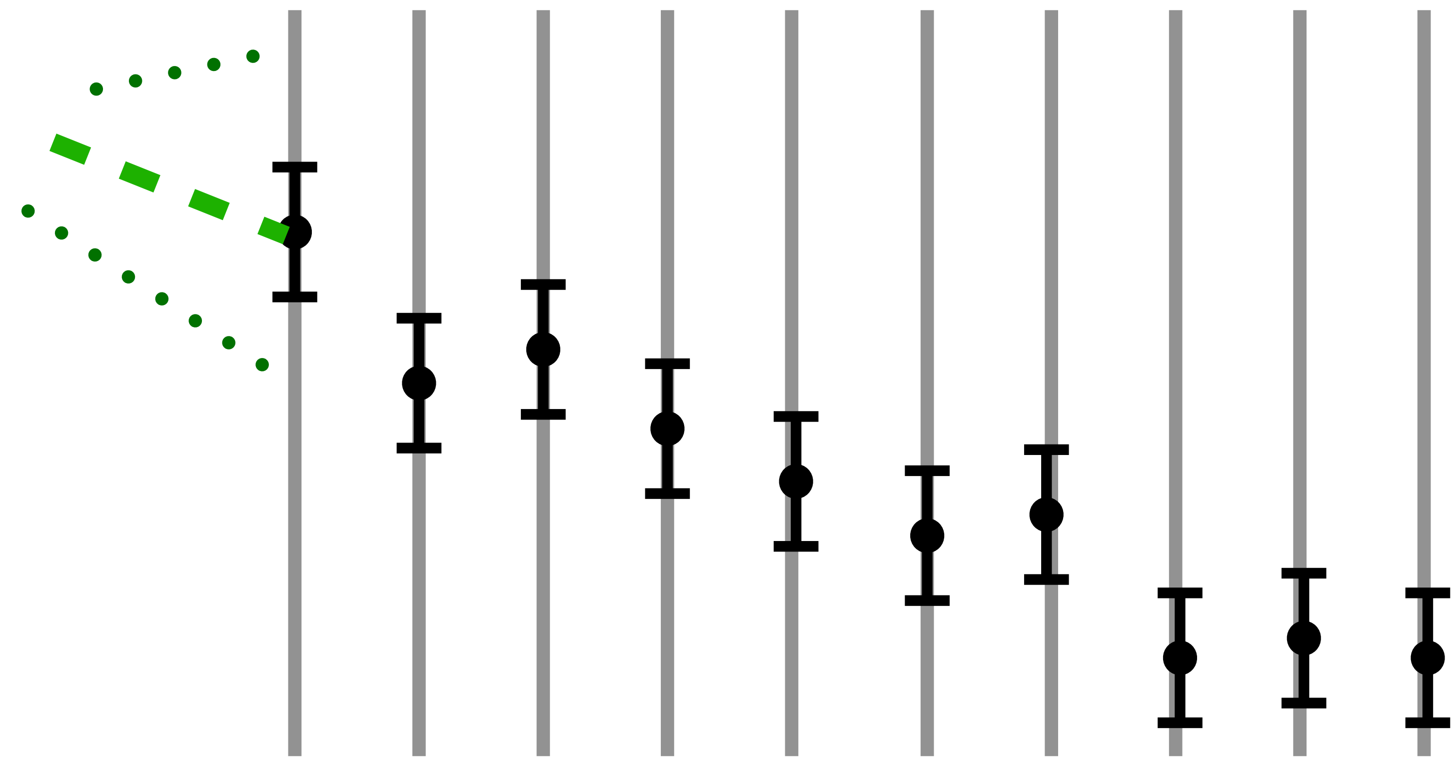
- ignore low-weight Gaussians
- merge Gaussians based on similarity (see Kullback-Leiber distance)

Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0

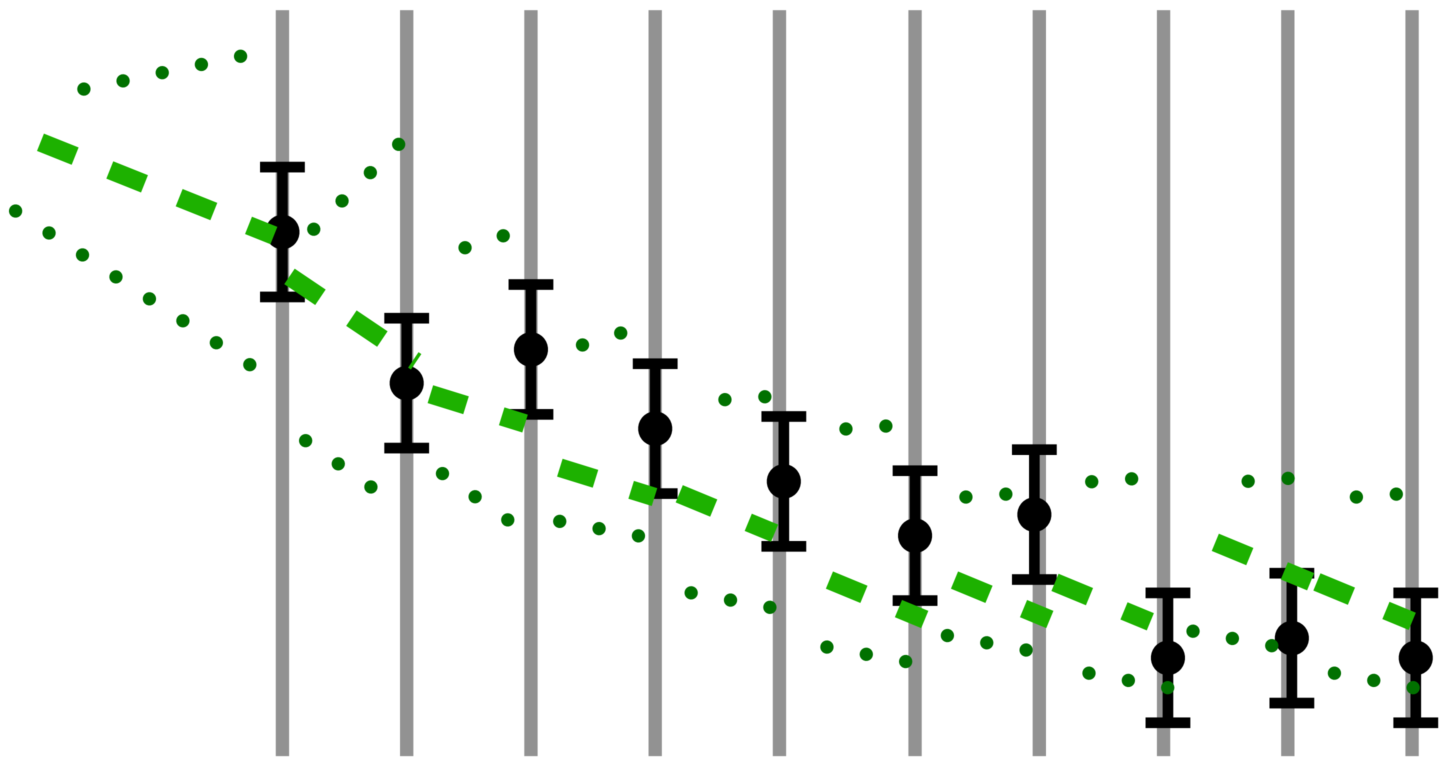
Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0
- Good first guess



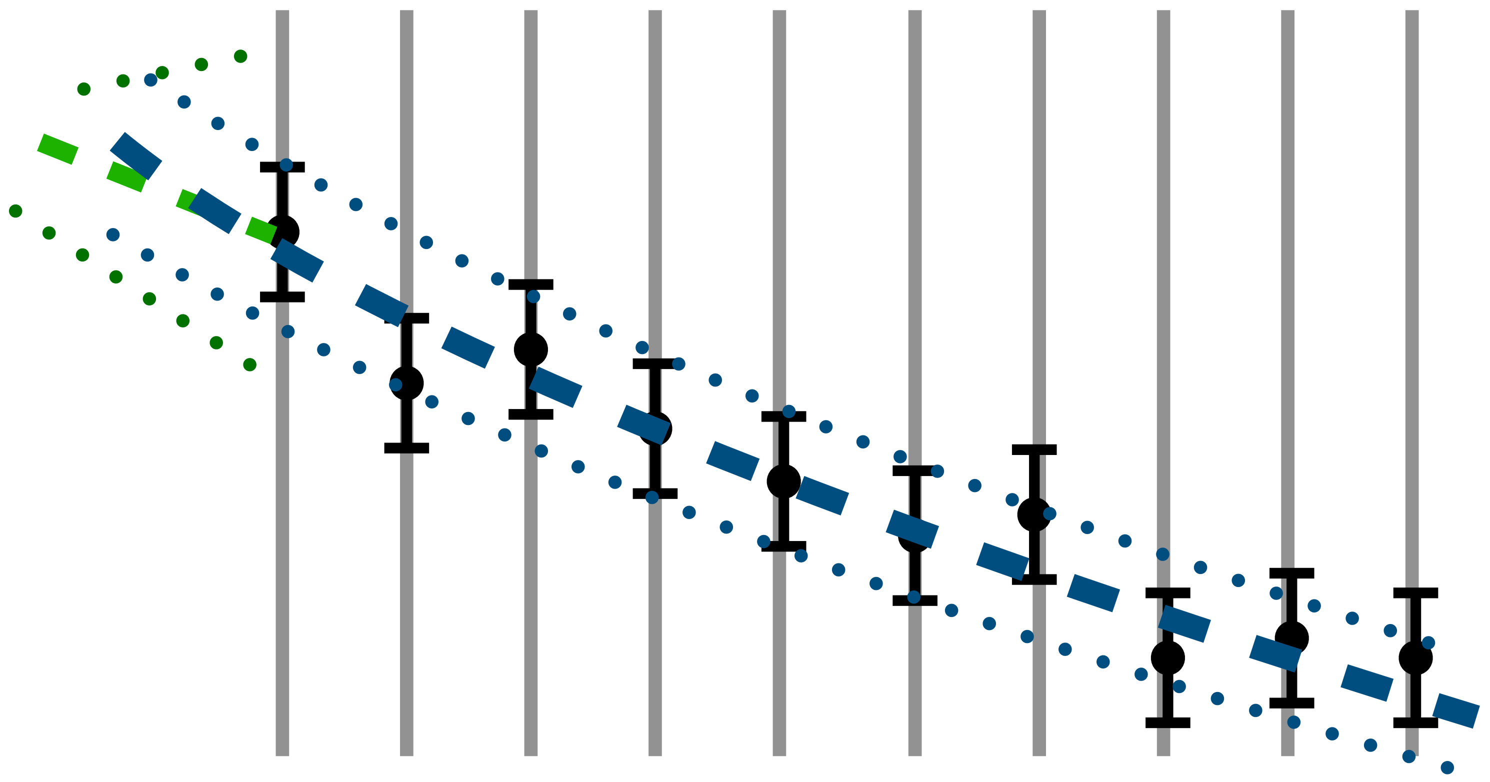
Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0
- Good first guess



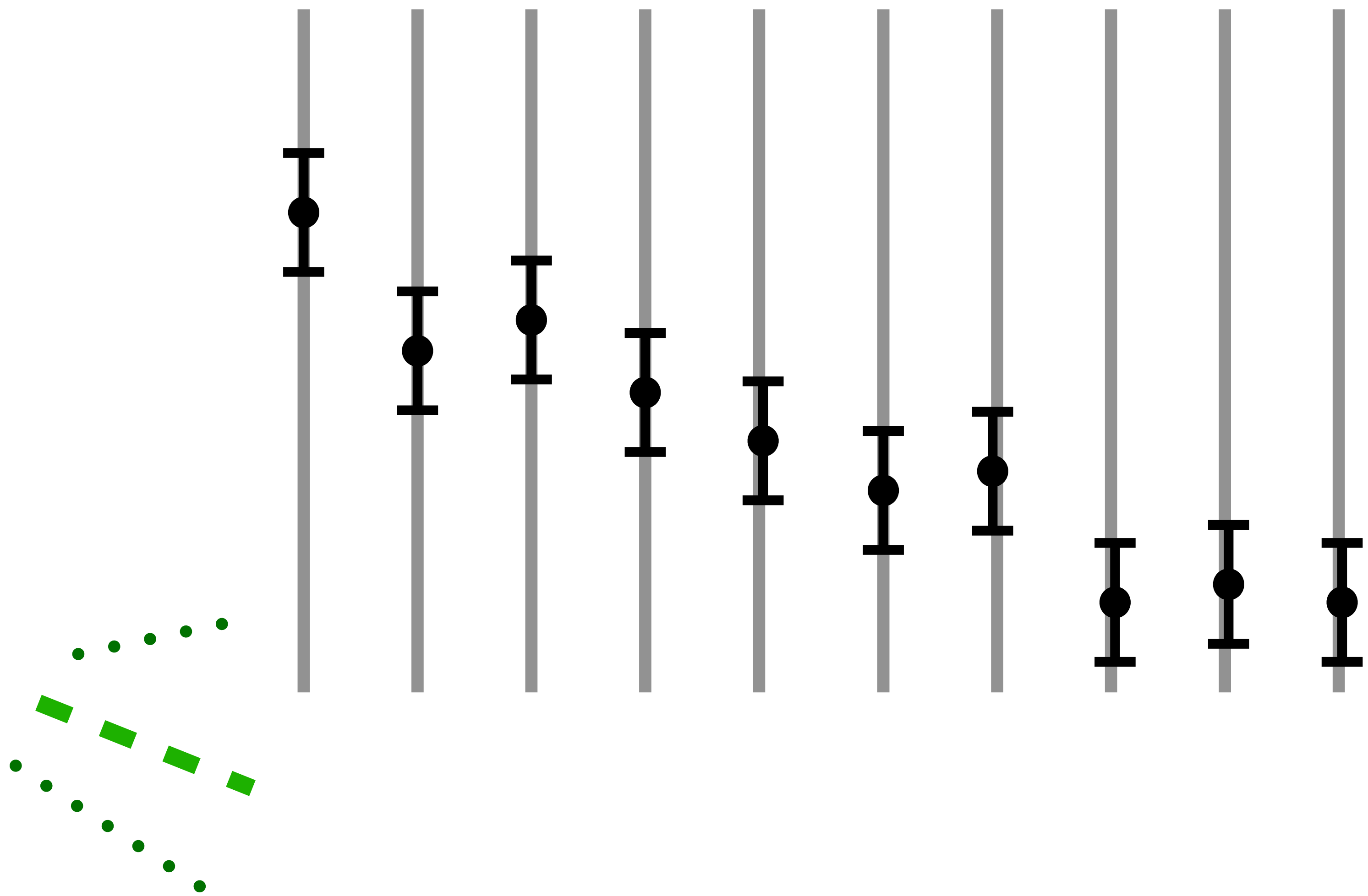
Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0
- Good first guess



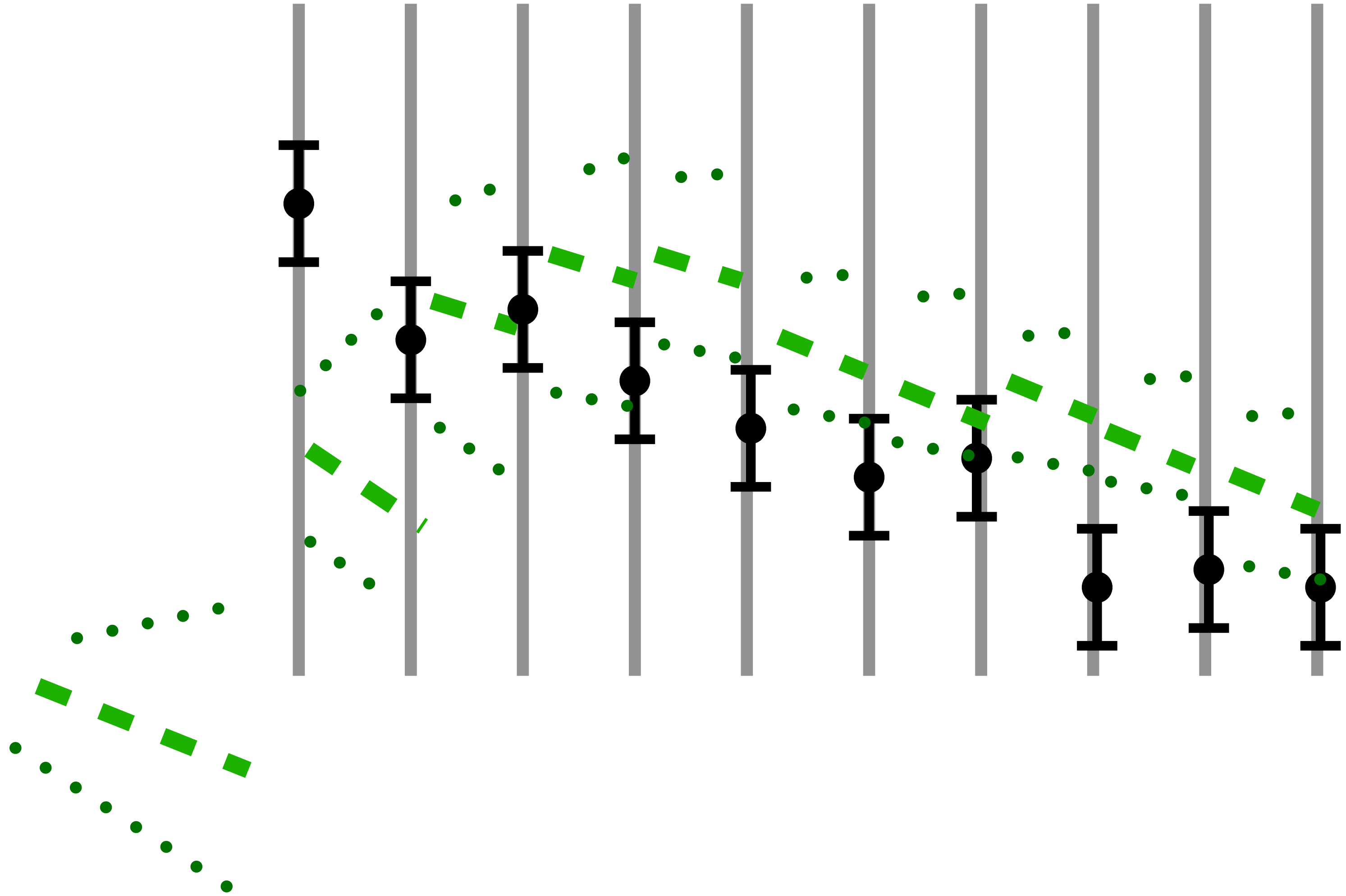
Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0
- Bad first guess



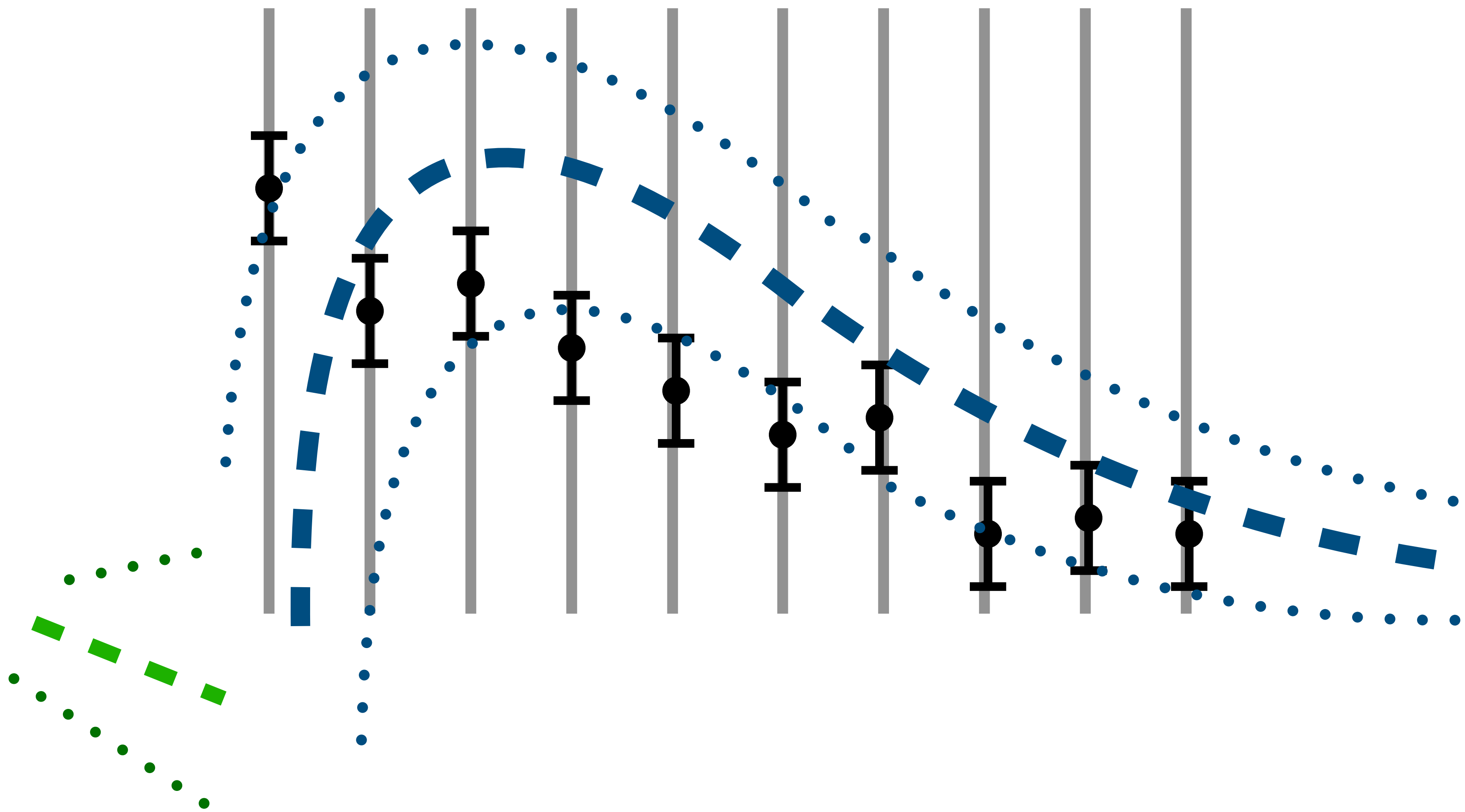
Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0
- Bad first guess



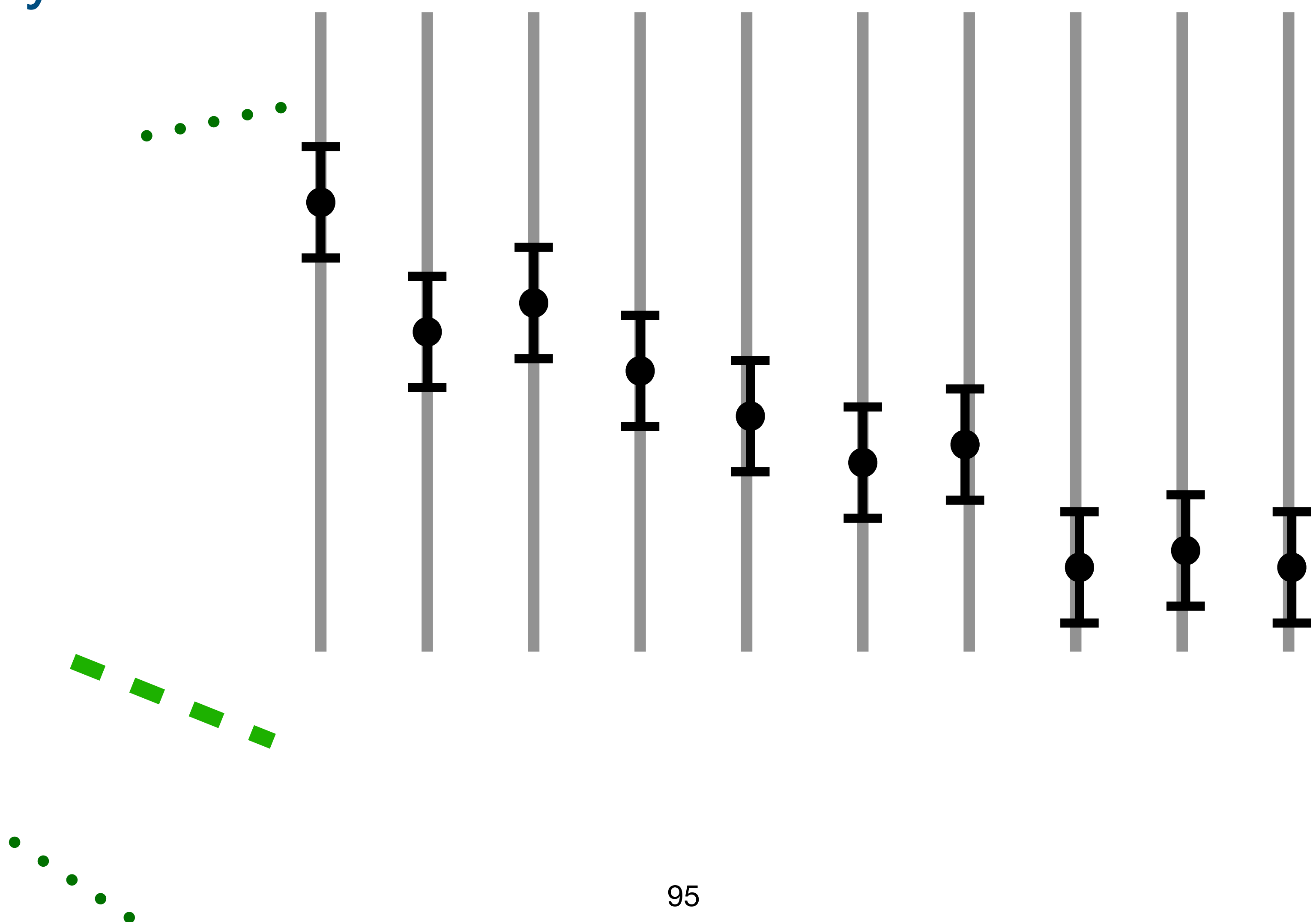
Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0
- Bad first guess



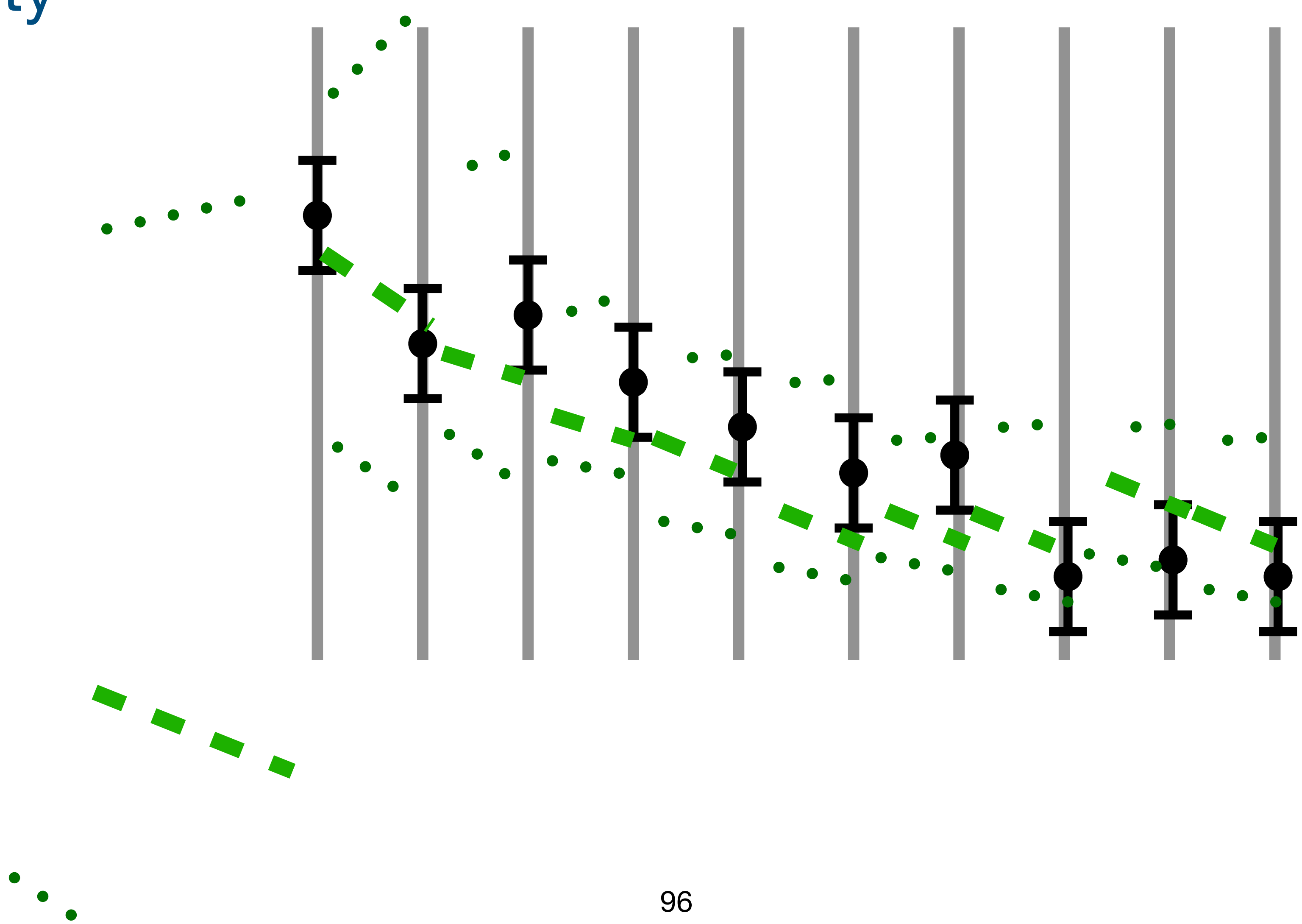
Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0
- Bad first guess but acknowledging it is bad : assign bigger uncertainty



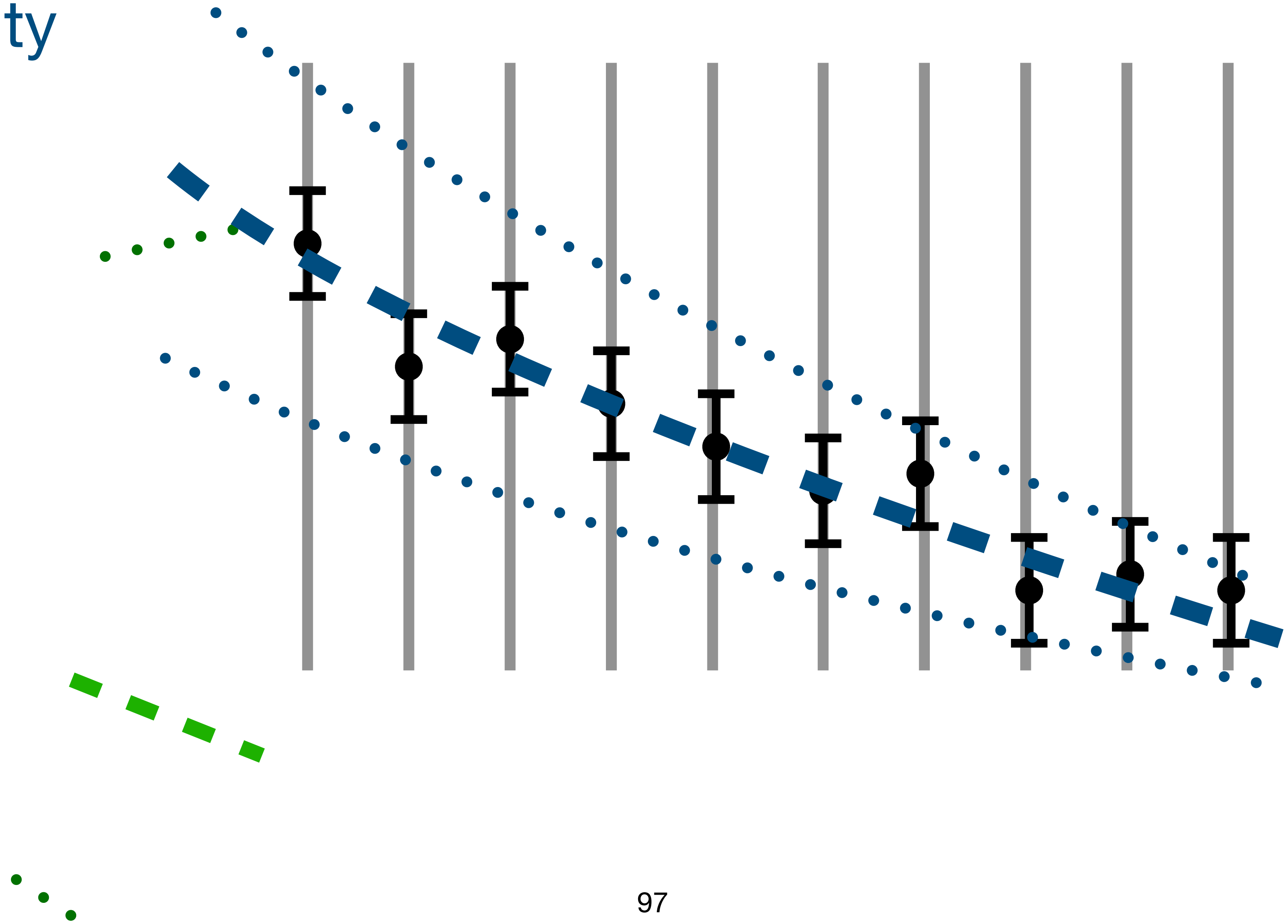
Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0
- Bad first guess but acknowledging it is bad : assign bigger uncertainty



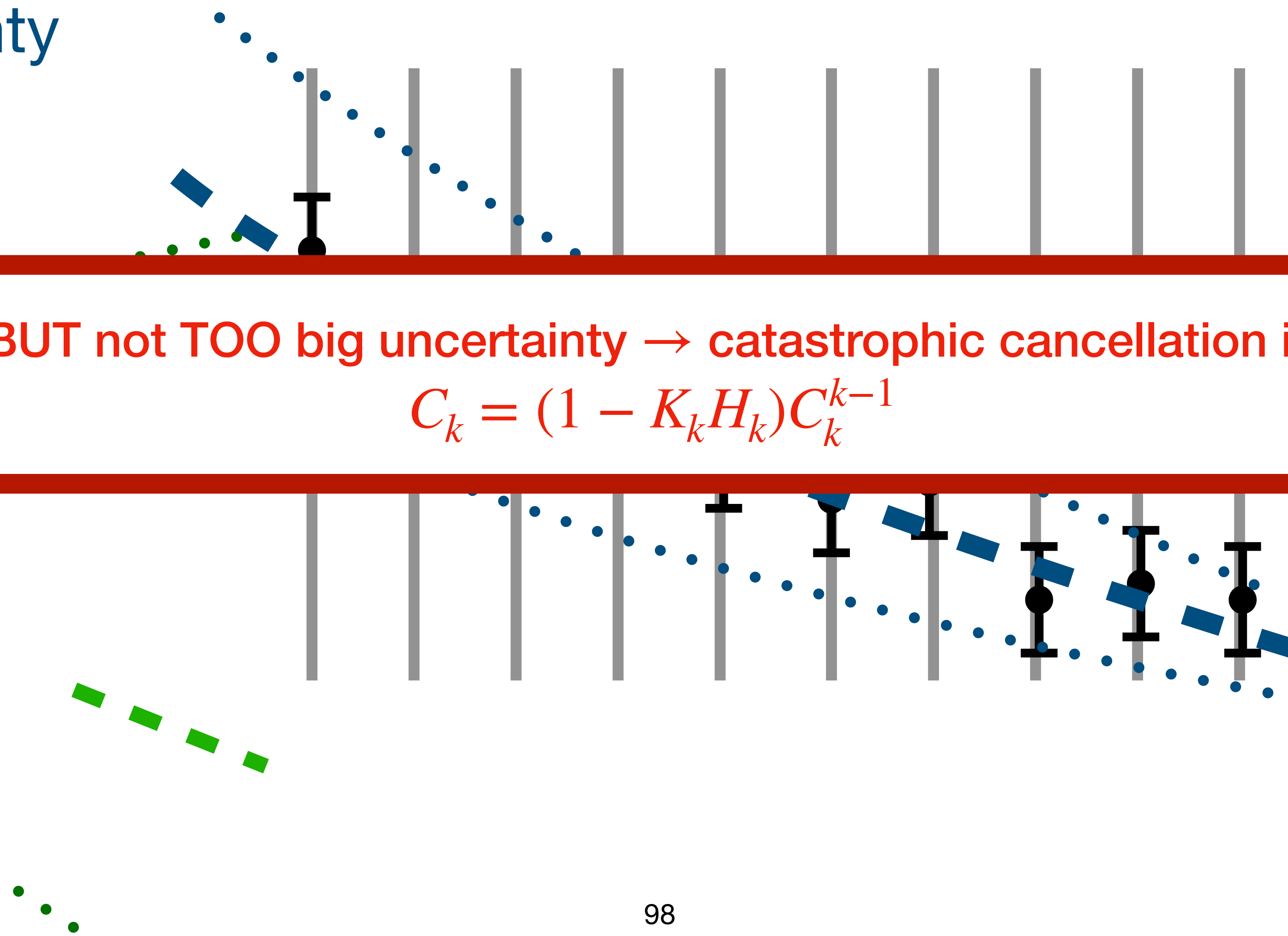
Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0
- Bad first guess but acknowledging it is bad : assign bigger uncertainty



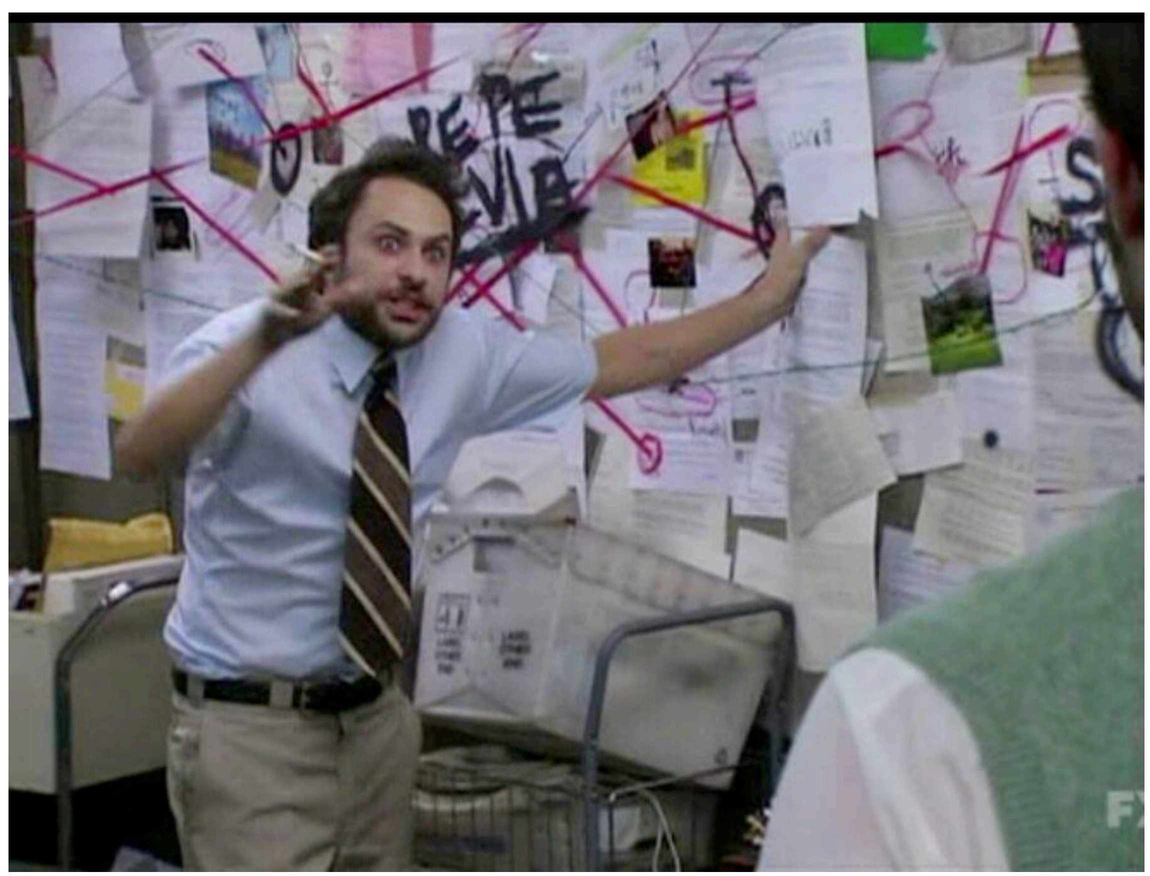
Problem 5: initialisation

- Kalman filter is a recursive algorithm : has to know state 0
- Bad first guess but acknowledging it is bad : assign bigger uncertainty



BUT not TOO big uncertainty → catastrophic cancellation in
$C_k = (1 - K_k H_k) C_k^{k-1}$

More things to keep in mind



Detector aging

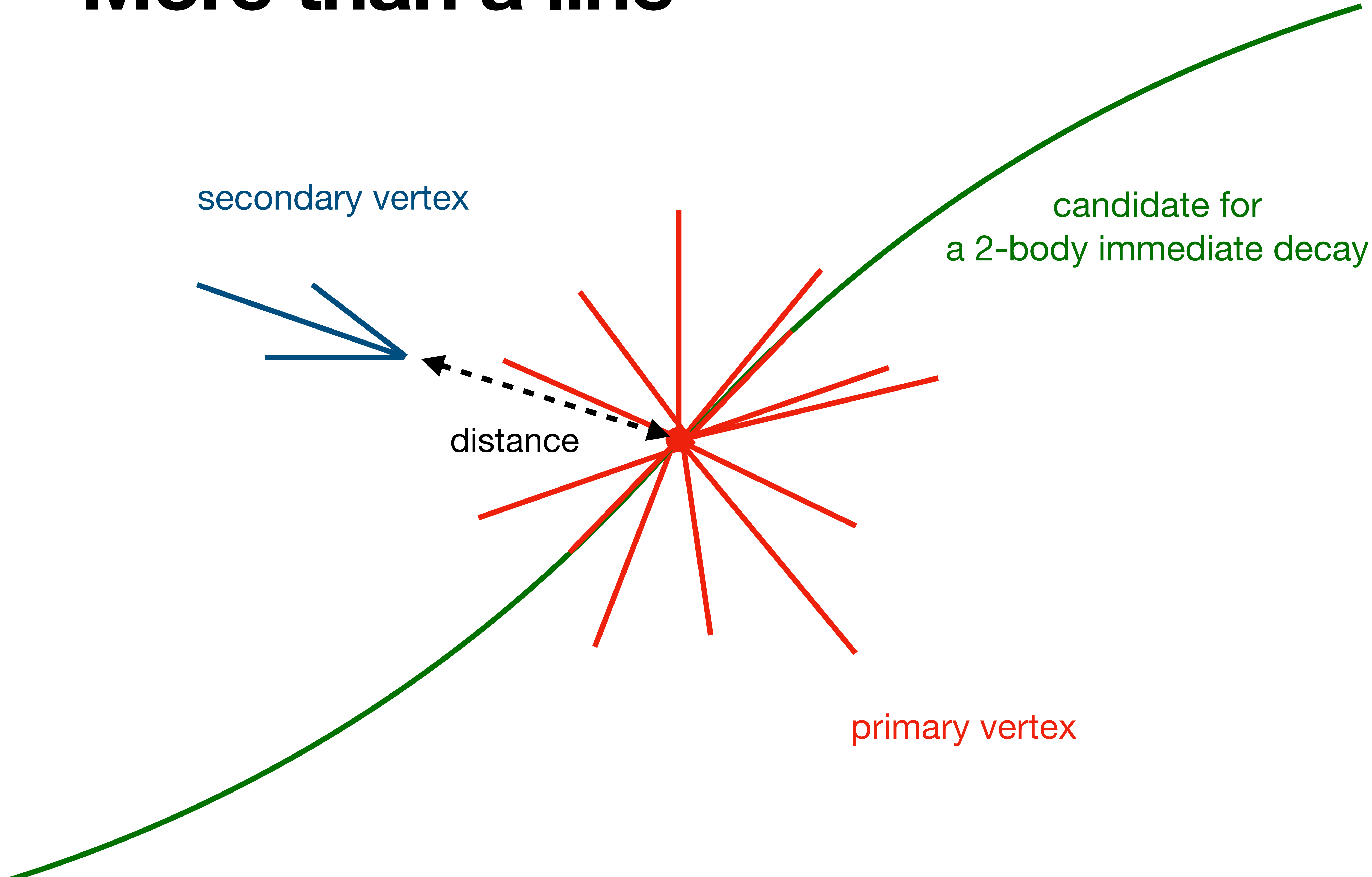
Everything gets older

- irradiation over years leads to worse detector performance

What you should make sure happens:

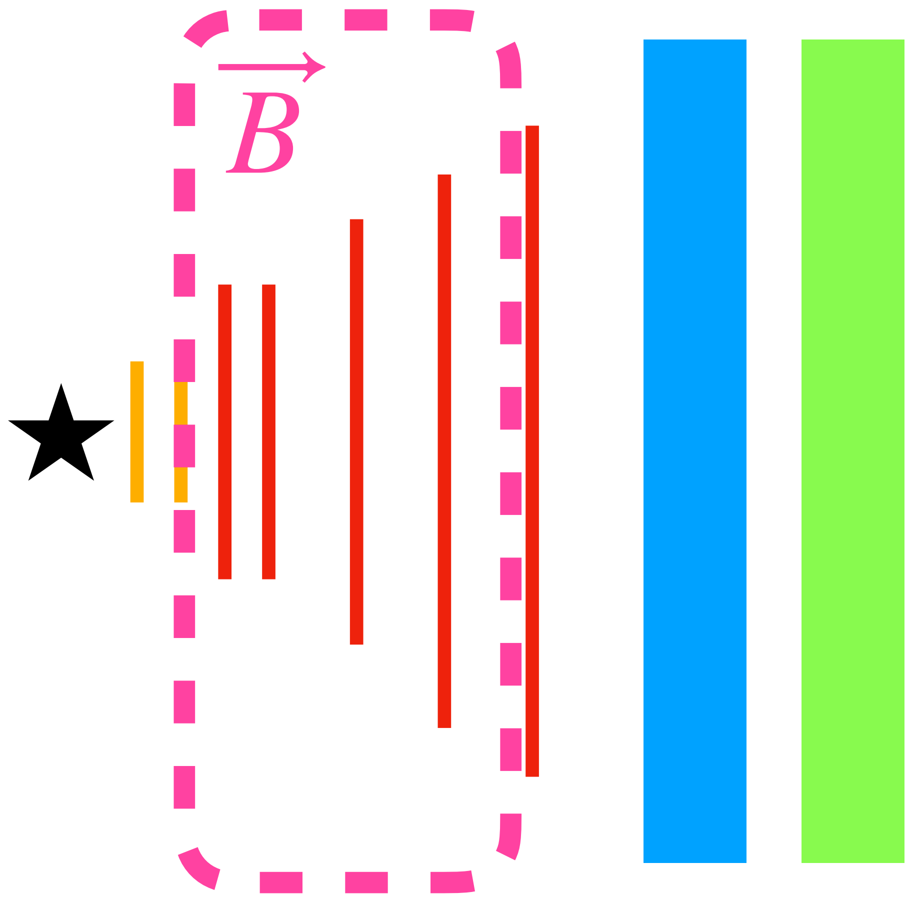
- continuous performance checks
- there is an easy way to change filter hardcoded conditions, like outliers removal χ^2

More than a line



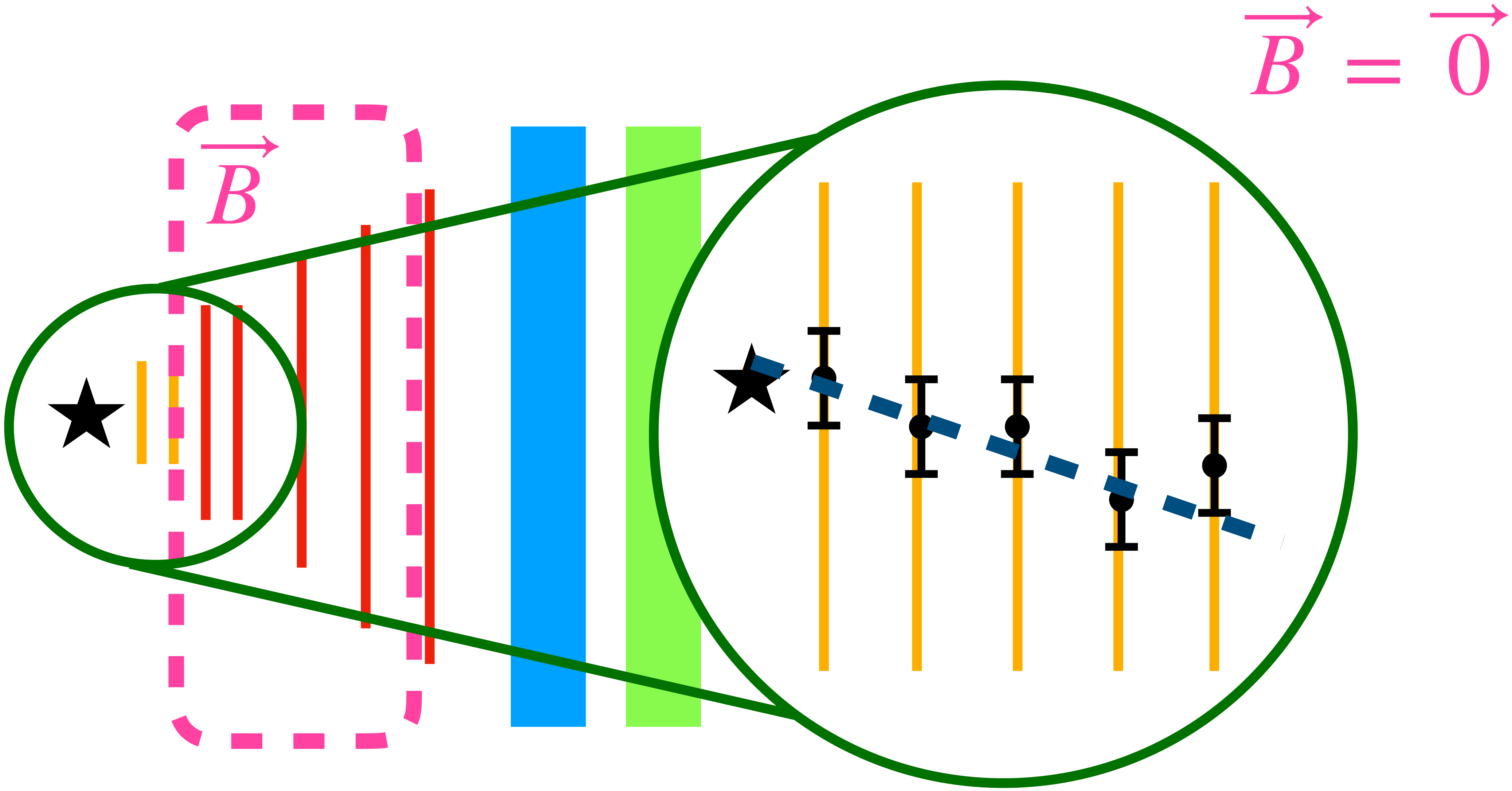
Can I cheat?

A simple LHCb-like example



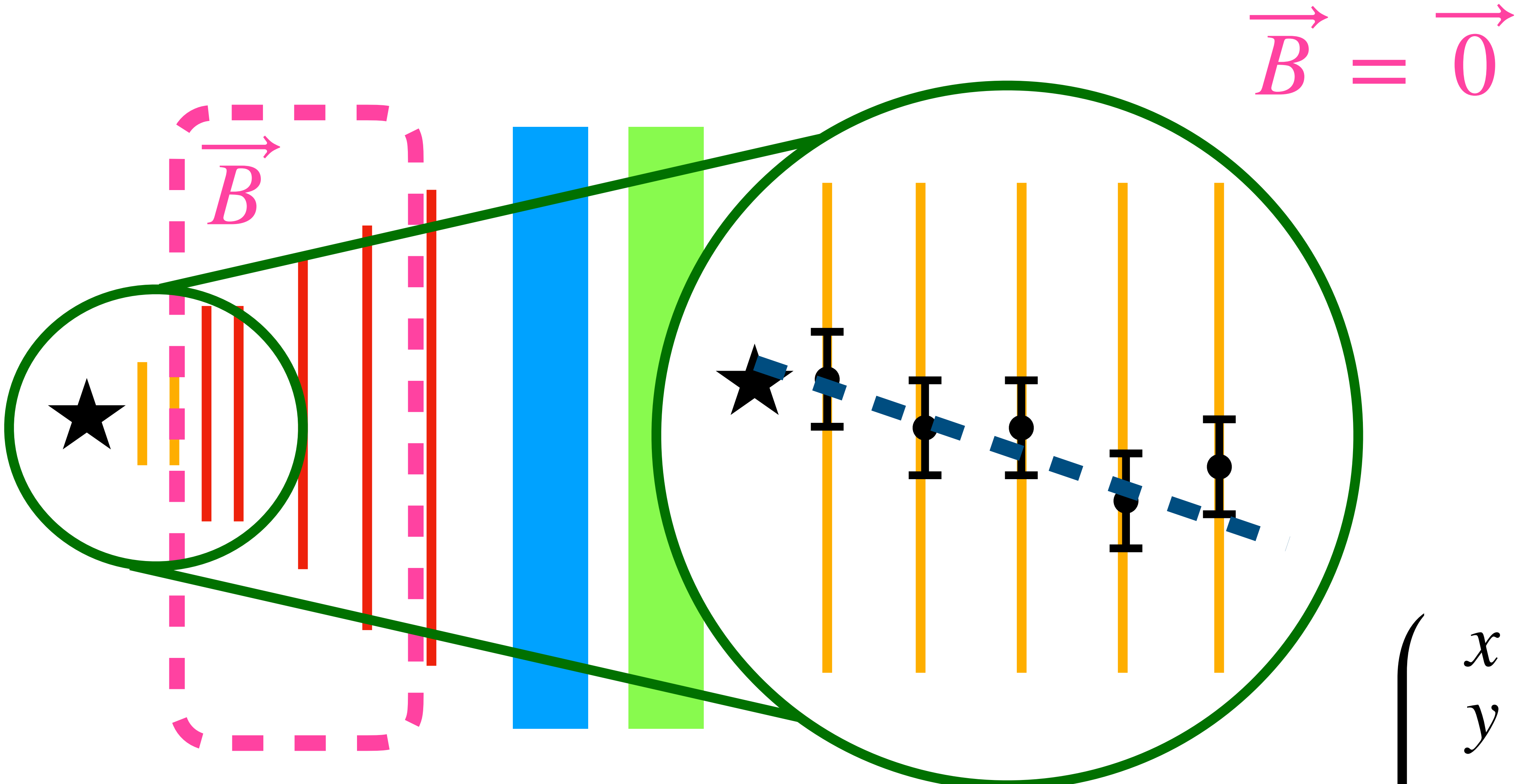
Can I cheat?

A simple LHCb-like example



Can I cheat?

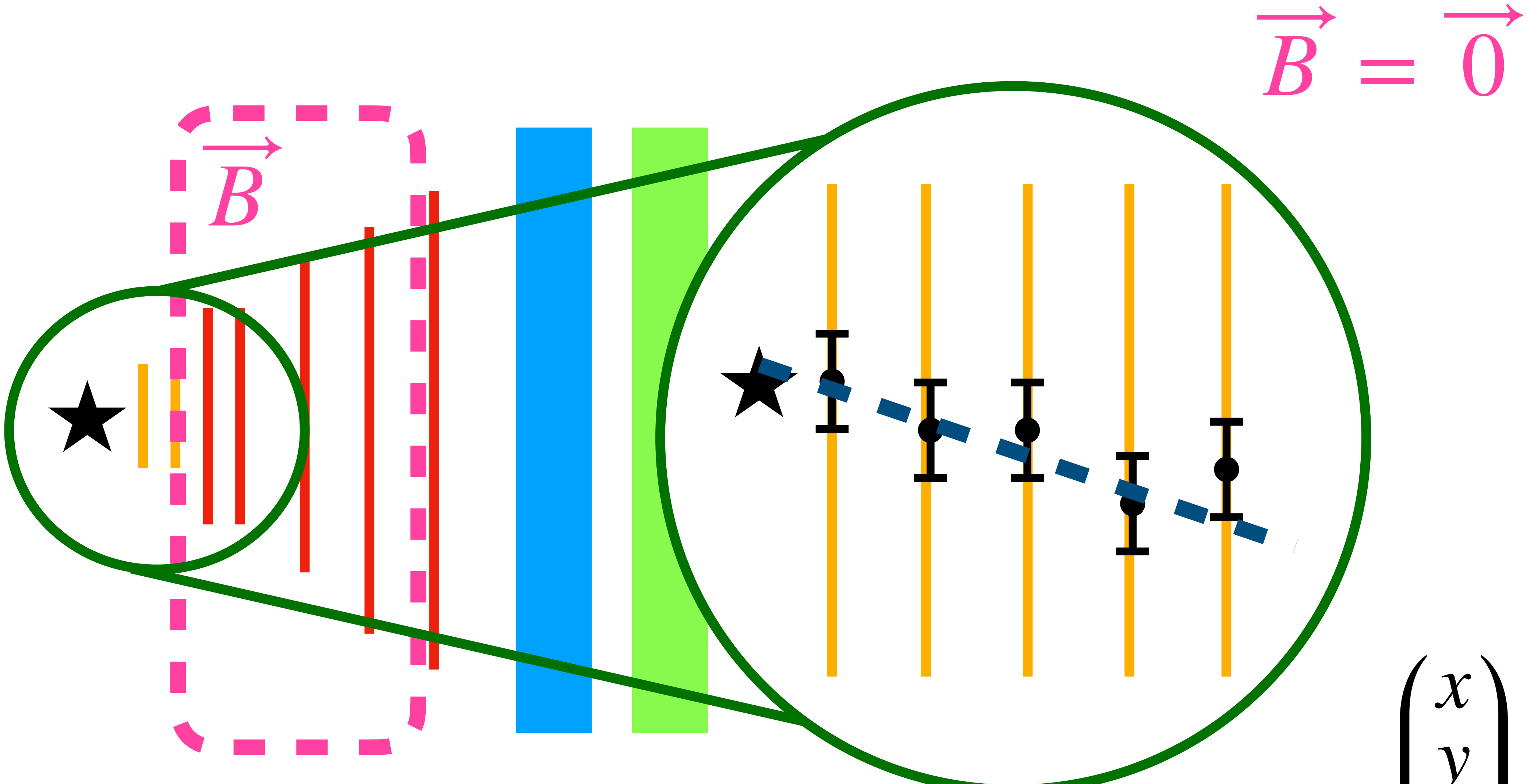
A simple LHCb-like example



$$\alpha\big|_{z_i} = \begin{pmatrix} x \\ y \\ t_x \\ t_y \\ q/p \end{pmatrix}$$

Can I cheat?

A simple LHCb-like example

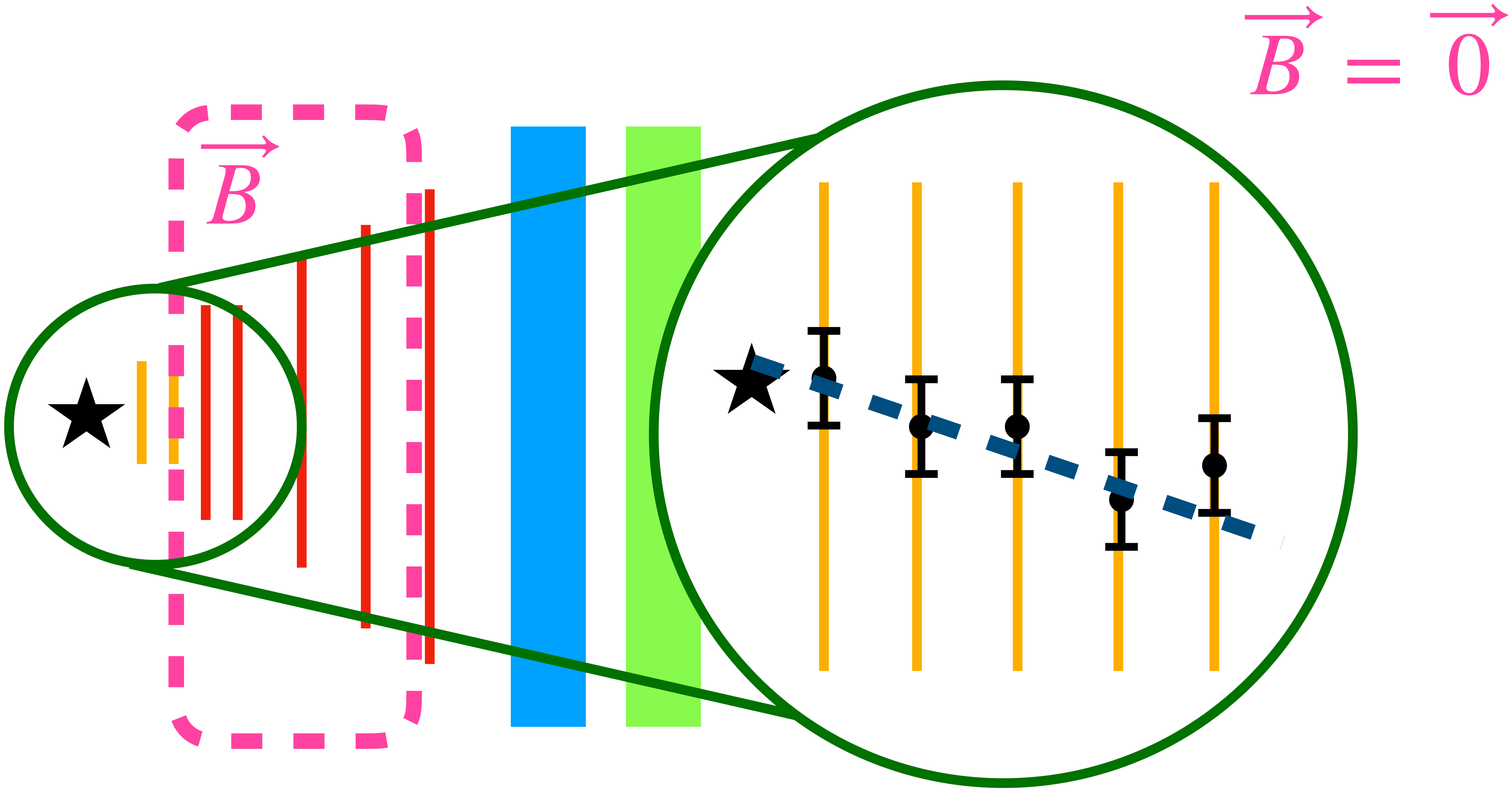


$$\alpha\big|_{z_i} = \begin{pmatrix} x \\ y \\ t_x \\ t_y \end{pmatrix}$$

q/p is not required in the computation but might still be associated to the track from the track finding algorithm

Can I cheat?

A simple LHCb-like example



$$\alpha\,\Big|_{z_i} = \begin{pmatrix} x \\ t_x \end{pmatrix} \quad \alpha\,\Big|_{z_i} = \begin{pmatrix} y \\ t_y \end{pmatrix}$$

q/p is not required in the computation but might still be associated to the track from the track finding algorithm

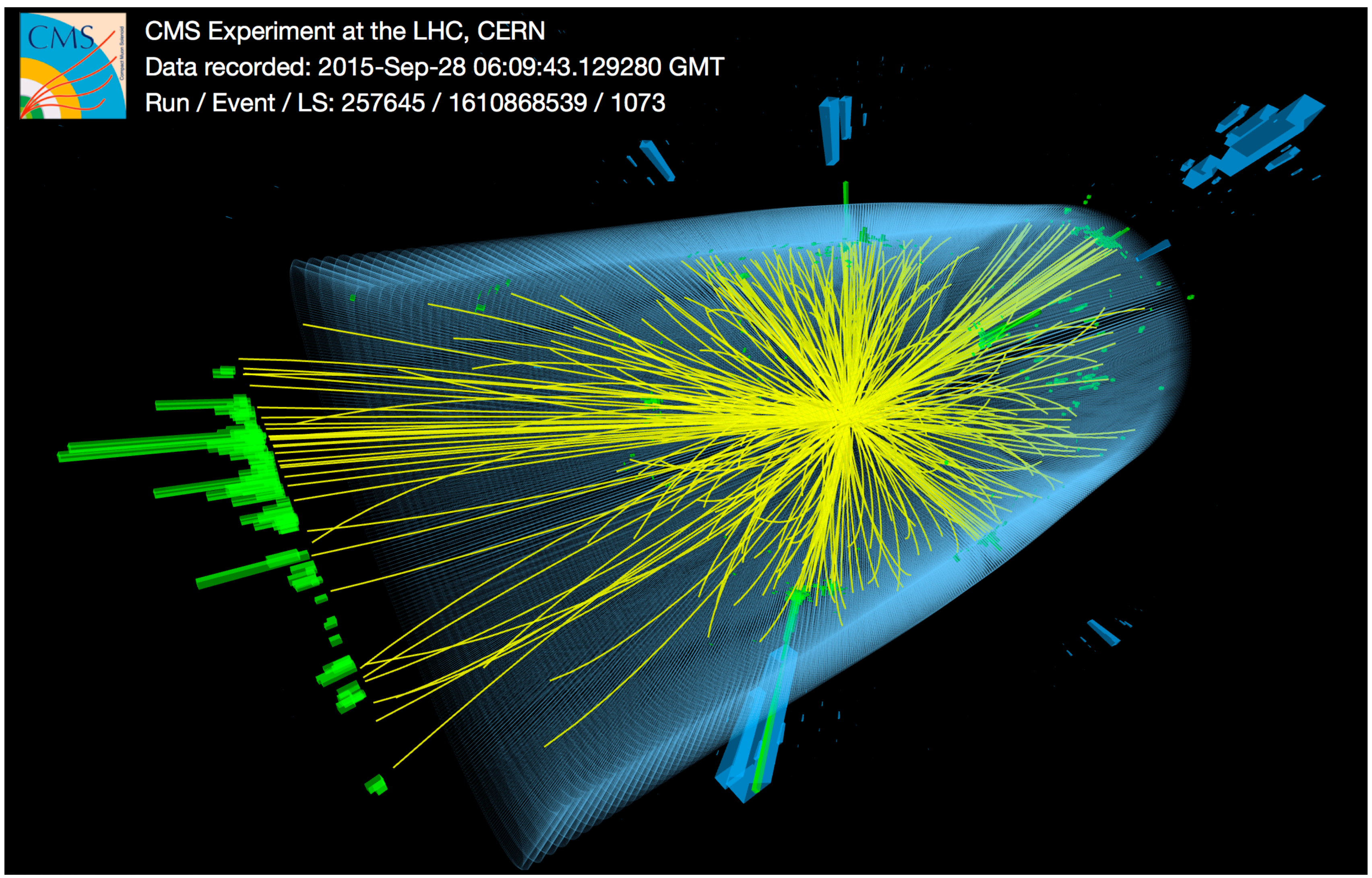
Parallelisation

Simply parallelizable?

PREDICTION FILTER . . . SMOOTHER → Track 1

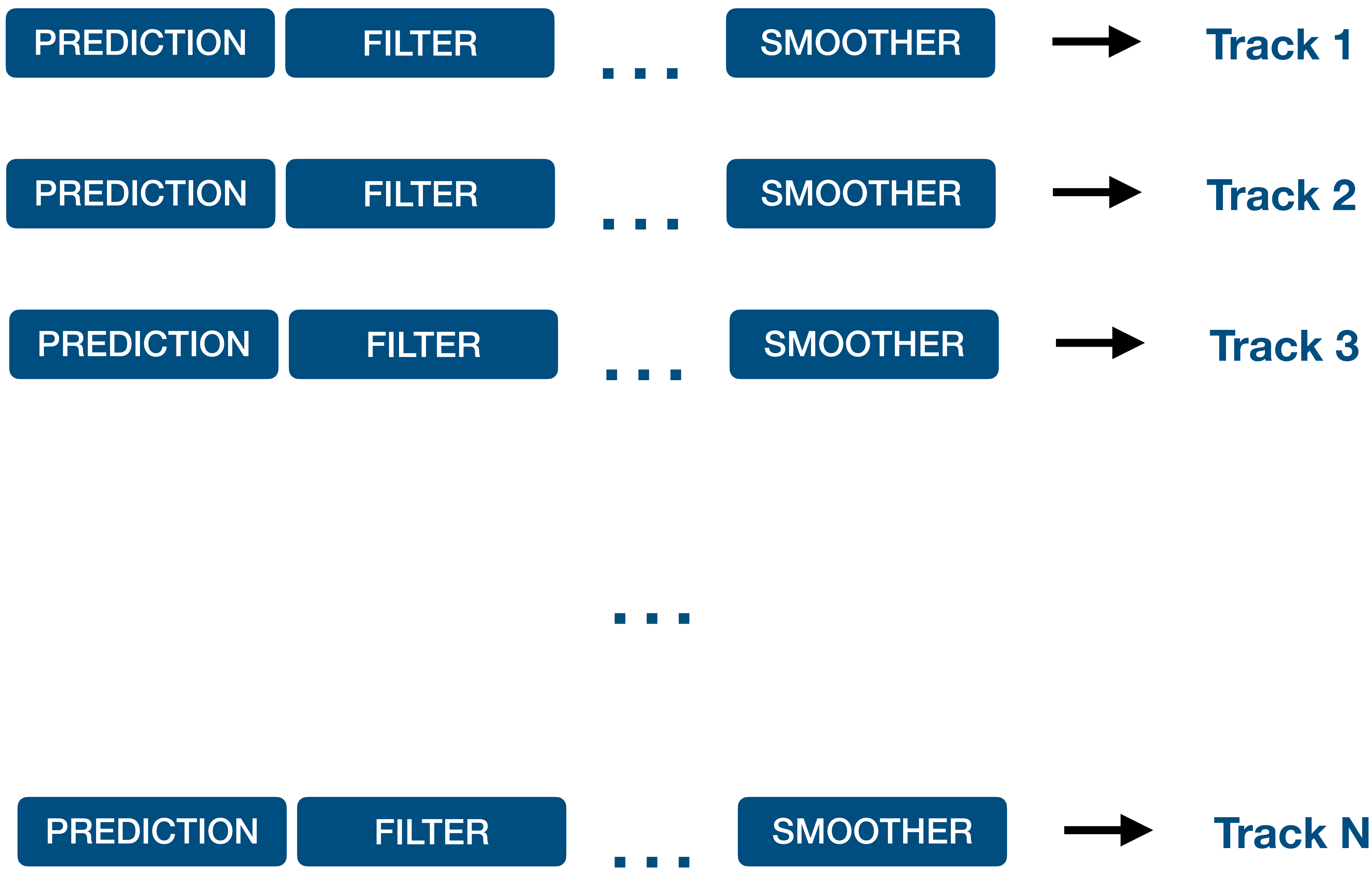
Simply parallelizable?

PREDICTION FILTER . . . SMOOTHER → **Track 1**



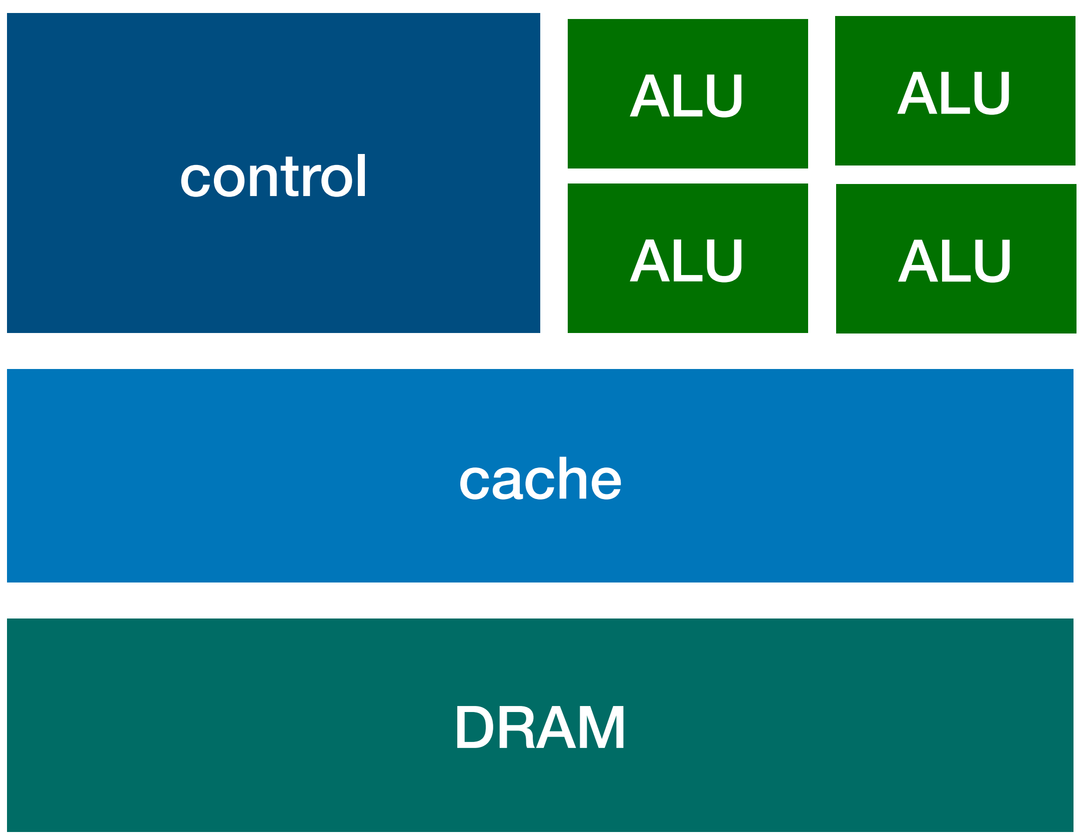
But in reality you have hundreds of tracks

Simply parallelizable?



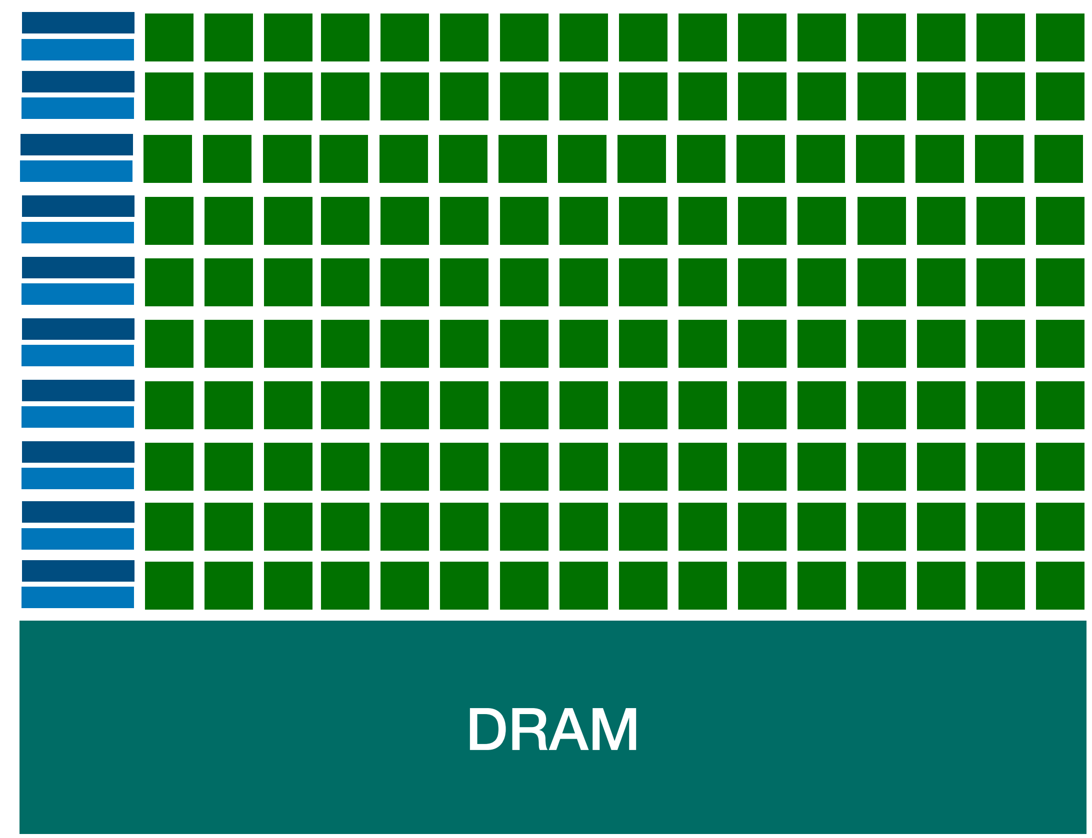
CPU vs GPU

CPU



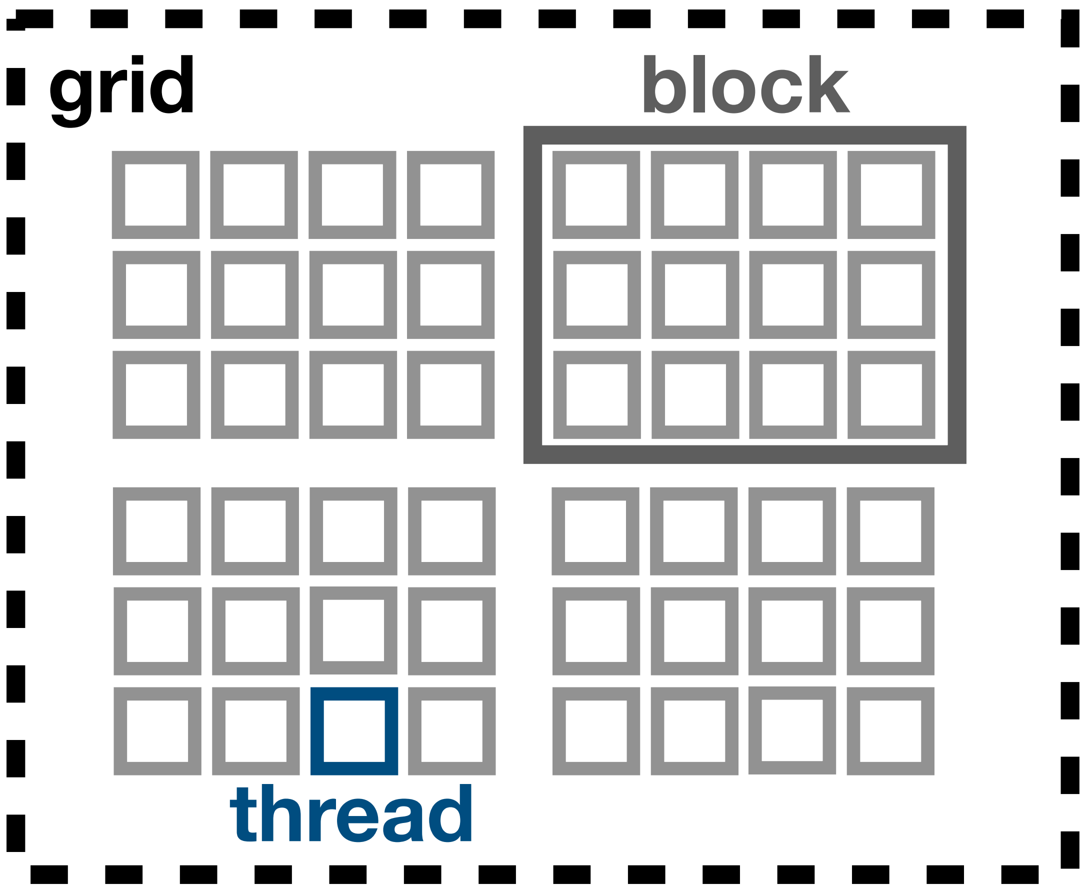
- Serial-oriented
- Low-latency
- Fewer cores, but powerful
- SIMD

GPU



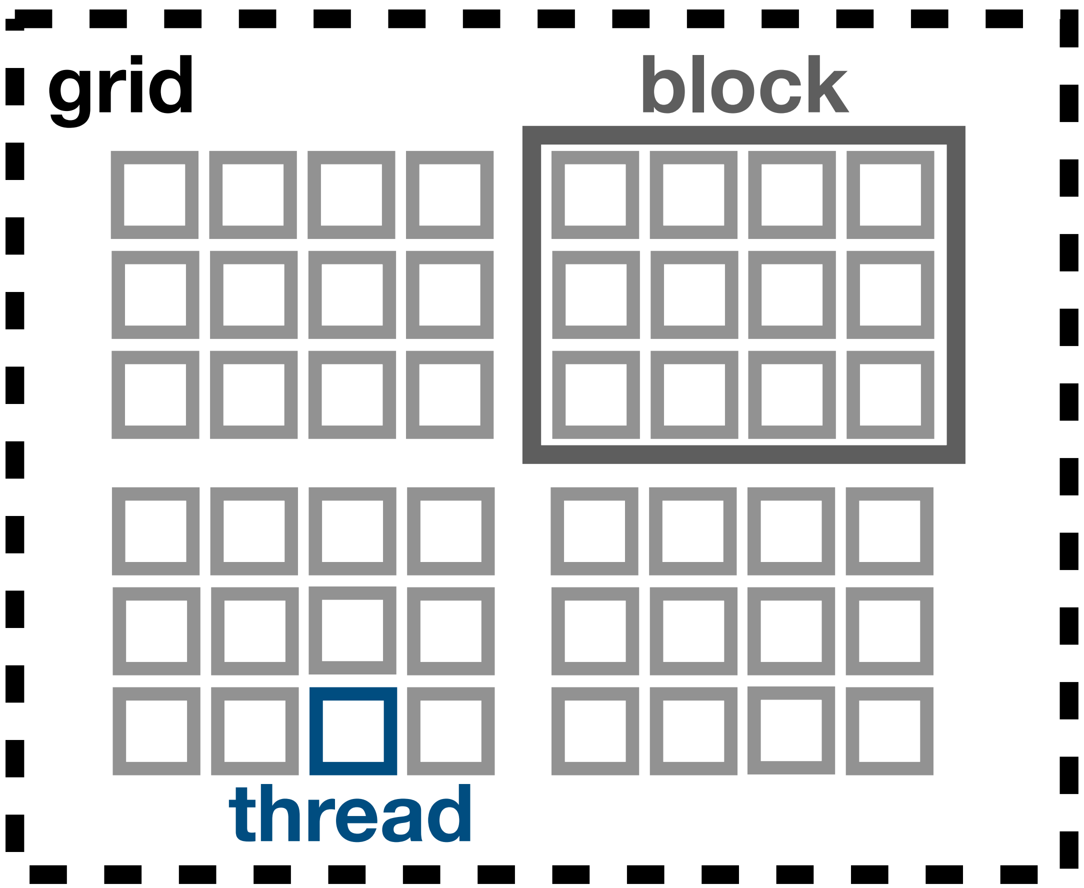
- Parallel-oriented
- High-latency
- More cores, but less powerful
- SIMT

GPU fitter



- each block of threads has shared memory
- two parallelisations
    - **track level** : each thread is a track
    - **intra-track** : each block is a track, each thread is a parallelised operation

GPU fitter



- each block of threads has shared memory
- two parallelisations
  - **track level** : each thread is a track
  - **intra-track** : each block is a track, each thread is a parallelised operation

Problems with GPU

1. Handling code divergence
2. Limited memory
3. Slow transfer of data to/from GPU

GPU fitter

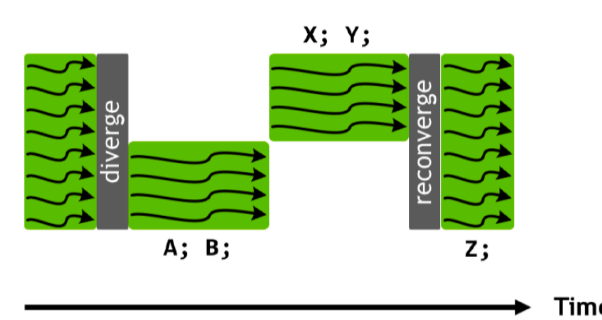
Problem 1 : command divergence

- Single Instruction Multiple Thread : assumes **commands are the same for all tracks**, if not - inefficiency

	FILTER 1	FILTER 2	FILTER 3	FILTER 4
TRACK 1	PLANE 1	PLANE 2	PLANE 3	wait
TRACK 2	PLANE 1	PLANE 2	wait	PLANE 4
TRACK 3	PLANE 1	wait	PLANE 3	PLANE 4
TRACK 4	PLANE 1	PLANE 2	wait	PLANE 4
TRACK 5	PLANE 1	wait	PLANE 3	PLANE 4

if-else blocks are dangerous
for the same reason (branch divergence):

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```

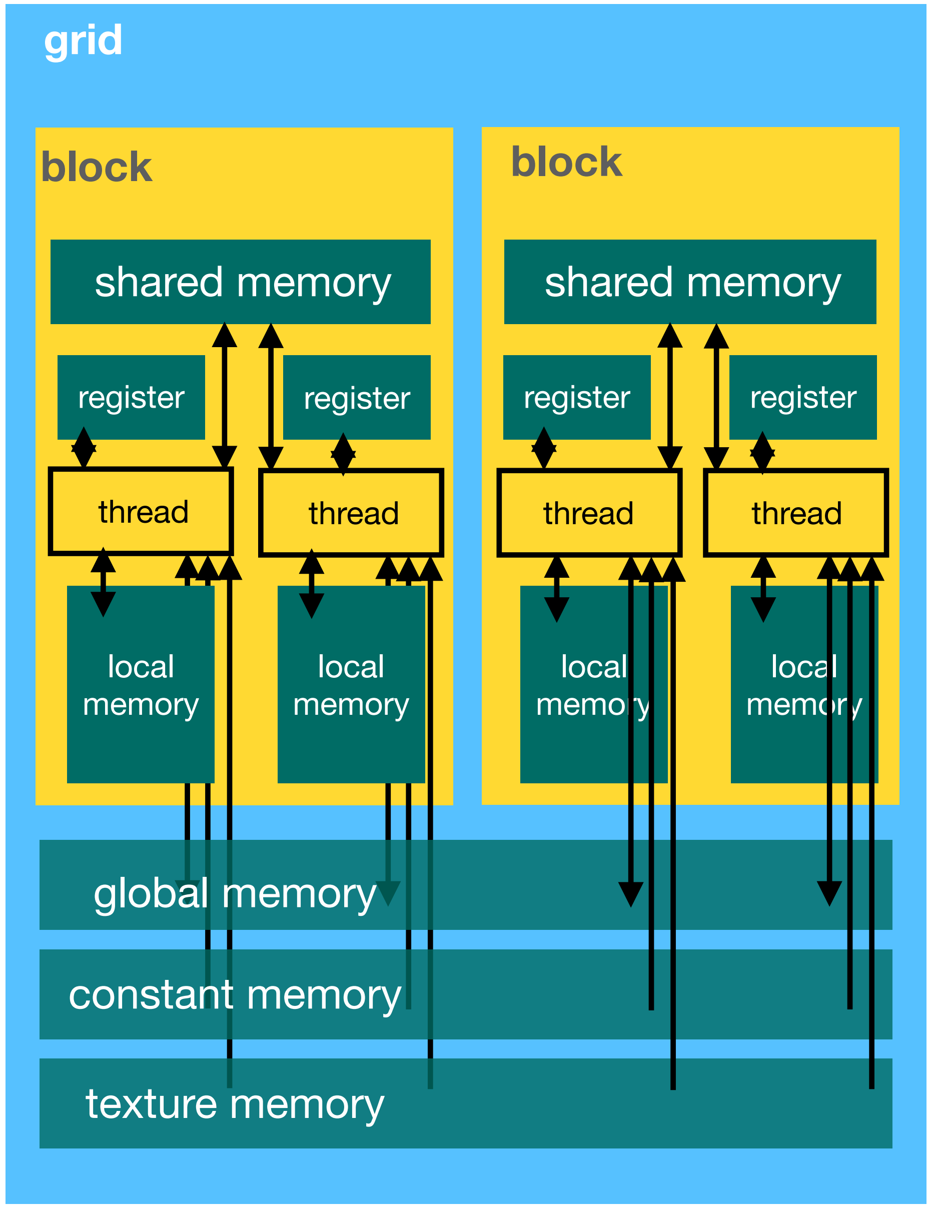


Note: on modern GPUs there are ways to improve on divergence

GPU fitter

Problem 2 : limited memory

- Limited memory per thread : especially problematic for recursive functions
- Numeric precision and rounding is typically worse



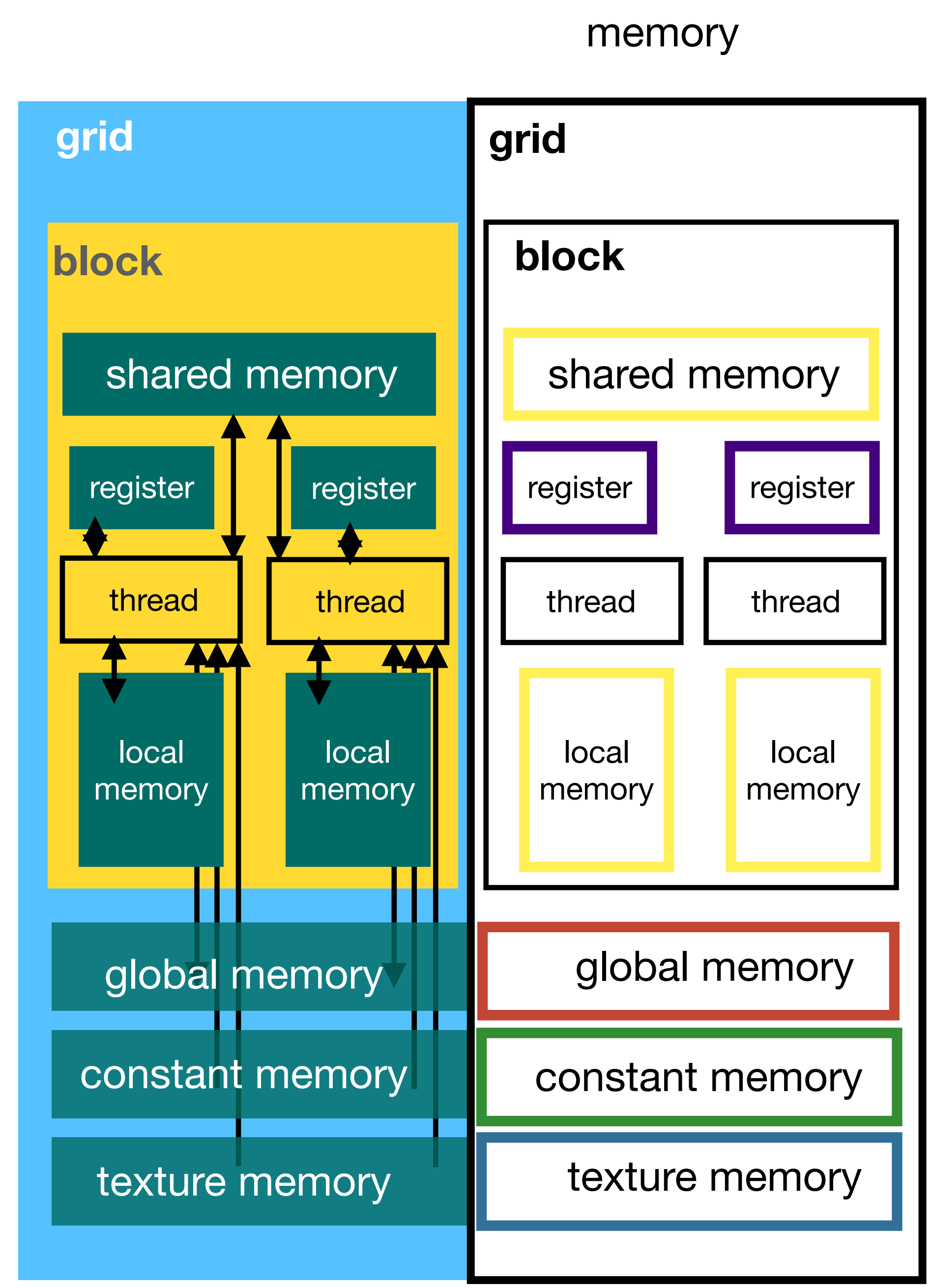
GPU fitter

Problem 2 : limited memory

- Limited memory per thread : especially problematic for recursive functions
- Numeric precision and rounding is typically worse

size small → big

register	texture	constant	shared and local	global
$\mathcal{O}(0.1kB)$	$\mathcal{O}(kB)$	$64kB$	$\mathcal{O}(10-100kB)$	$\mathcal{O}(GB)$



Note: this is all approximate as concrete numbers depend on the card, but gives you a rough idea of orders

GPU fitter

Problem 2 : limited memory

- Limited memory per thread : especially problematic for recursive functions

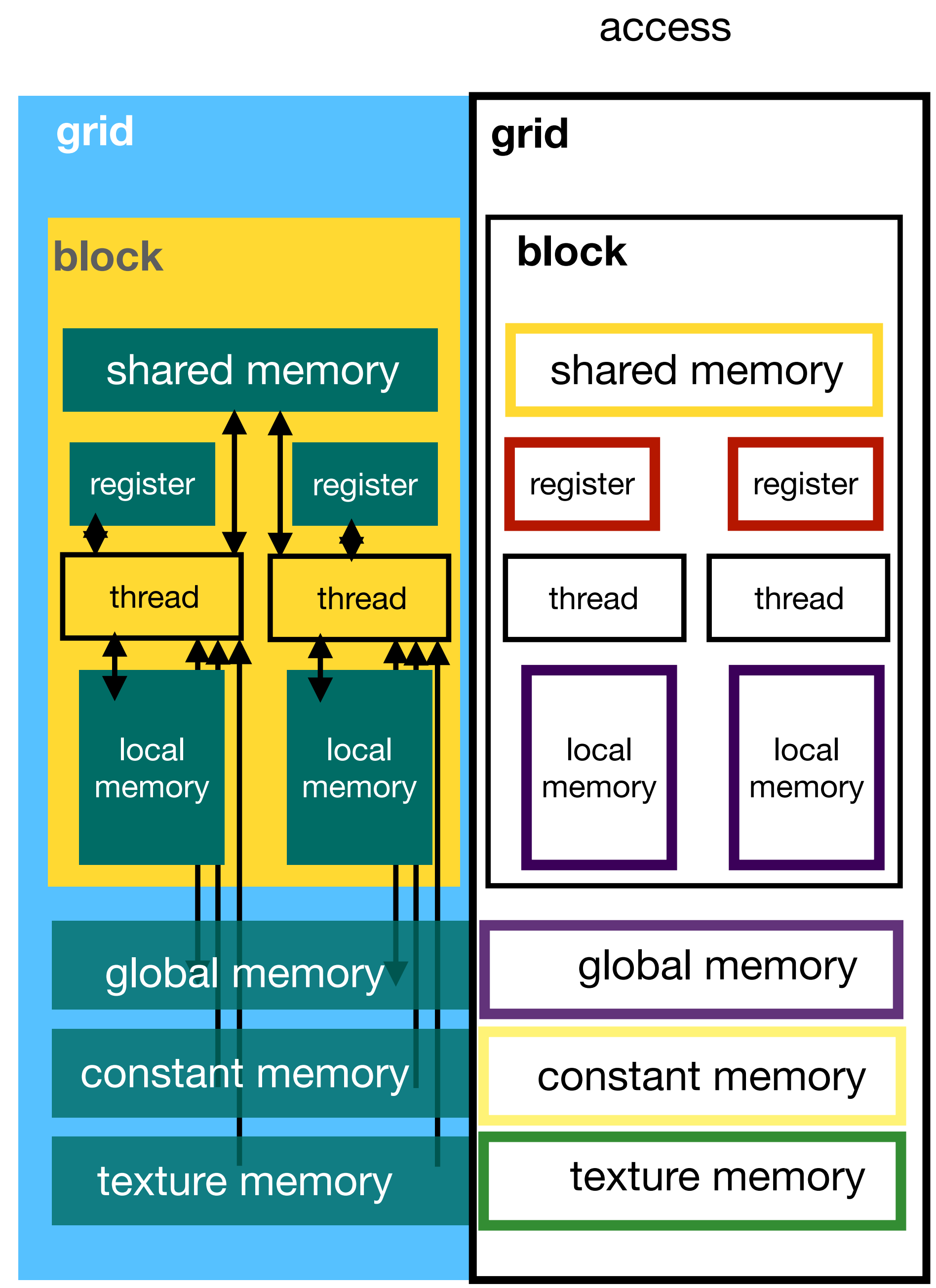
- Numeric precision and rounding is typically worse

size (small → big)

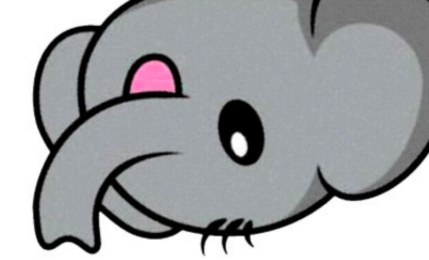
register	**texture**	**constant**	**shared and local**	**global**
$\mathcal{O}(0.1kB)$	$\mathcal{O}(kB)$	$64kB$	$\mathcal{O}(10-100kB)$	$\mathcal{O}(GB)$

access (slow → fast)

global and local	**texture**	**constant**	**shared**	**register**
300-800 clock cycles	*50-100* clock cycles	*1-3* clock cycles	*1-3* clock cycles	*0.5-1* clock cycles



Note: this is all approximate as concrete numbers depend on the card, but gives you a rough idea of orders



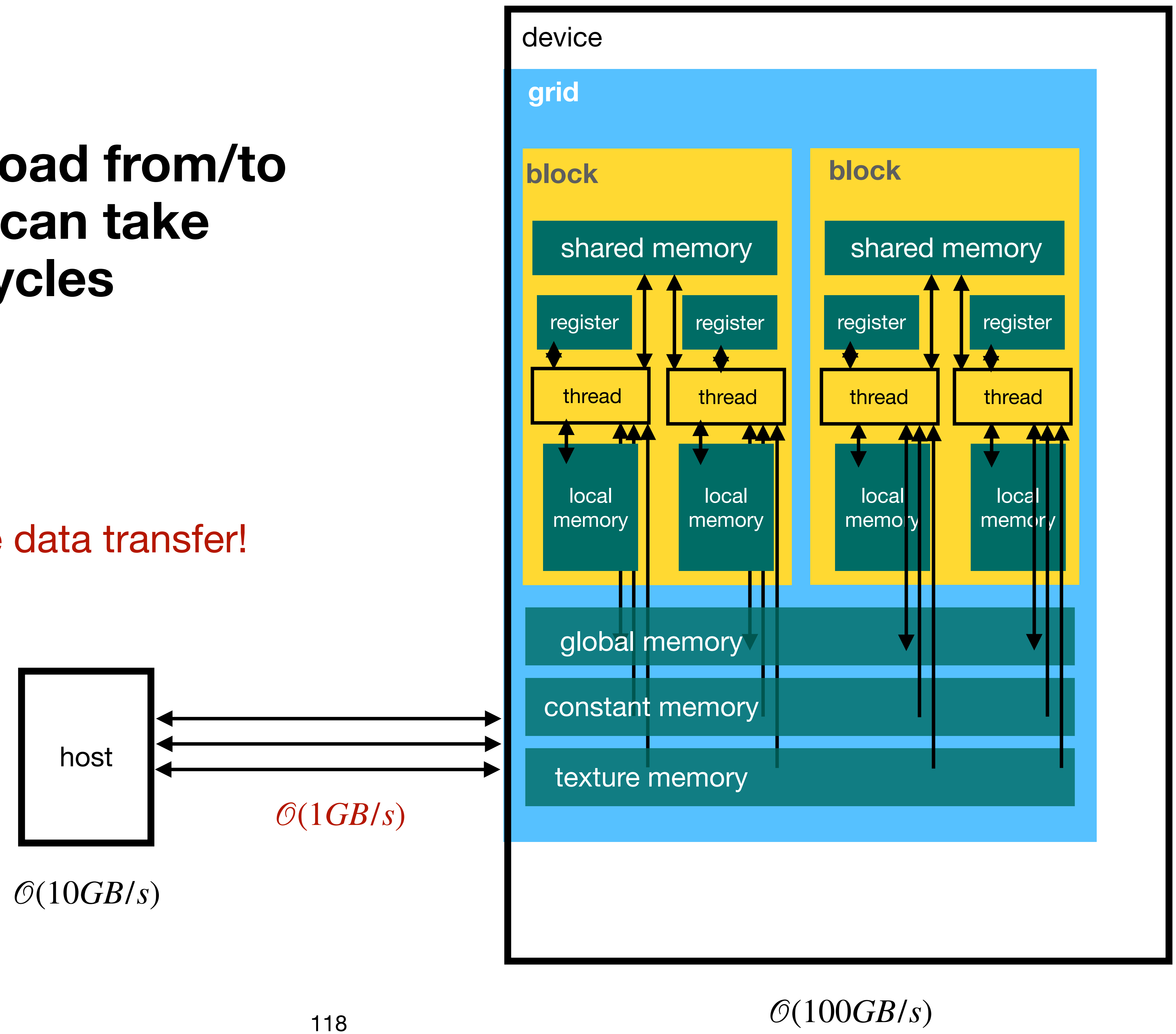
GPU fitter

Problem 3 : costly transfer

- **Upload/download from/to GPU is slow - can take 1000s clock cycles**

minimize host-device data transfer!

heterogeneous architecture

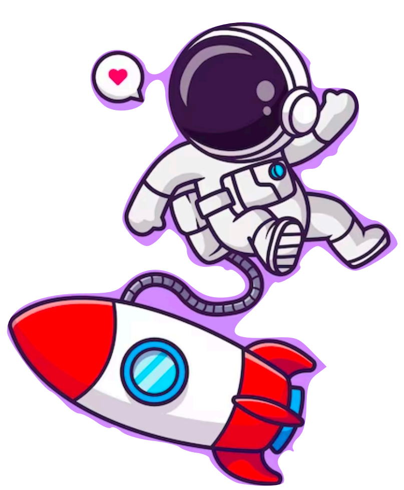


Big ideas to take home

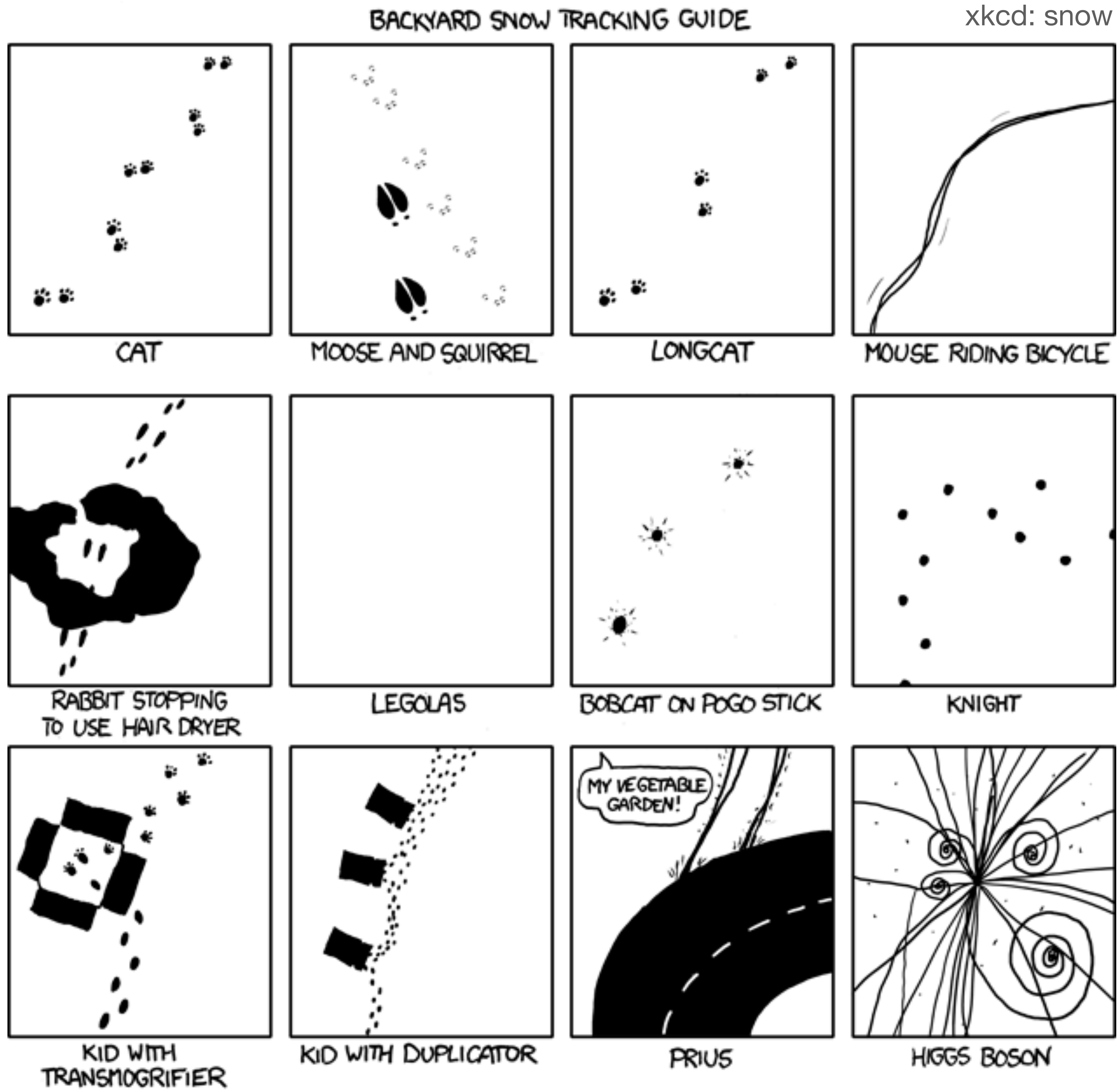
1. **Kalman filter** is a **powerful** fitting tool : problem is simplified to 1-equations solving for M-times

2. **Kalman filter** implementation is **tricky**: numerical instabilities, outliers, initialisation, non-linearity etc.

3. Track fitting is a **good candidate for parallelisation**



The end?



xkcd: snow tracking