# Virtual Metrology 1:
## Building the Model

ECE 8803-AI4 · AI for Semiconductor Manufacturing

https://aikhan123.github.io/khan-lab-website/ai-semiconductors-course.html

**Asif Khan**
Associate Professor
School of Electrical and Computer Engineering
School of Materials Science and Engineering
Georgia Institute of Technology
akhan40@gatech.edu

# What you should be able to do by the end of today

Explain why a fab measures three wafers out of twenty-five, and what that costs.

Say precisely what virtual metrology predicts, and what it does not.

Build a wafer-level feature table from segmented sensor traces.

Recognise $p \gg n$, and name the two things it breaks.

Fit and read a PLS model, and choose its number of components by cross-validation.

State the baseline any virtual metrology model has to beat before it is worth deploying.

**Next lecture asks whether the model we build today would survive contact with a fab. Today we build it.**

# Where we left off

**Lecture 3 spent two slides on this idea. Today it gets two lectures.**

## What Lecture 3 claimed

If sensor traces can predict the measurement, predict it for every wafer. You get a number on all 25 wafers instead of 3, at no metrology cost.

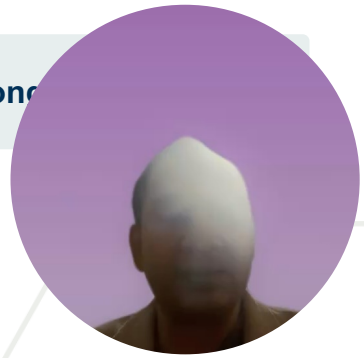
Tighter subgroups, faster detection, and feedback for the run-to-run controller on lots nobody measured.

## What Lecture 3 warned

You are now charting a model output. If the model degrades, the chart moves and the process never changed.

Every deployed model needs a control chart on its own residual.

**Today: how the model gets built. Next lecture: why the number it reports in your notebook is usually wrong**



SEGMENT 1 OF 4

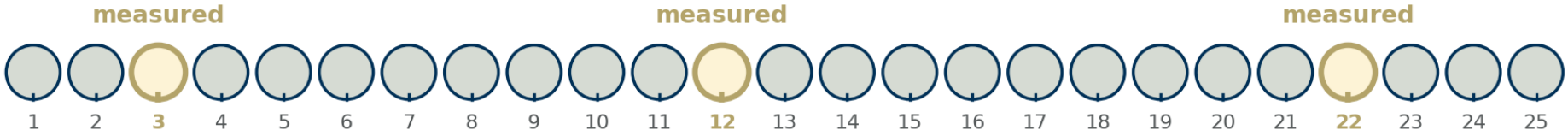
# Why virtual metrology exists

Metrology is not expensive. Metrology is slow, and there is never enough of it.

# The arithmetic of a metrology plan

A lot is 25 wafers. A sampling plan is 3.

## One lot: 25 wafers etched




**3 wafers measured · 22 wafers leave with no number on them**

**Virtual metrology is about the other 22.**

Every process decision you make about that lot rests on three numbers, and you chose which three.

# What the sampling plan costs you

**Three consequences, all of which you met in Lectures 1 to 3**

## Your charts are blunt

n = 3 or n = 5 per lot. Limits are ±3σ/√n, so small n means wide limits and slow detection. Everything in Lecture 2 about average run length applies here.

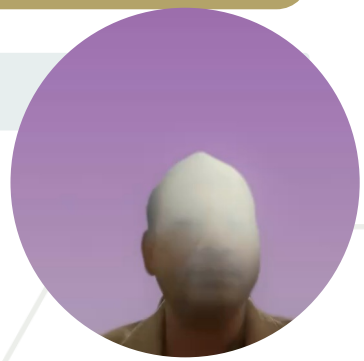
## Your controller is starved

A run-to-run controller updates on measurements. On an unmeasured lot it has nothing, so it holds its last correction and drift accumulates unopposed.

## You ship before you know

The CD-SEM queue is hours. By the time a number arrives, four more lots have run through the same chamber with the same recipe.

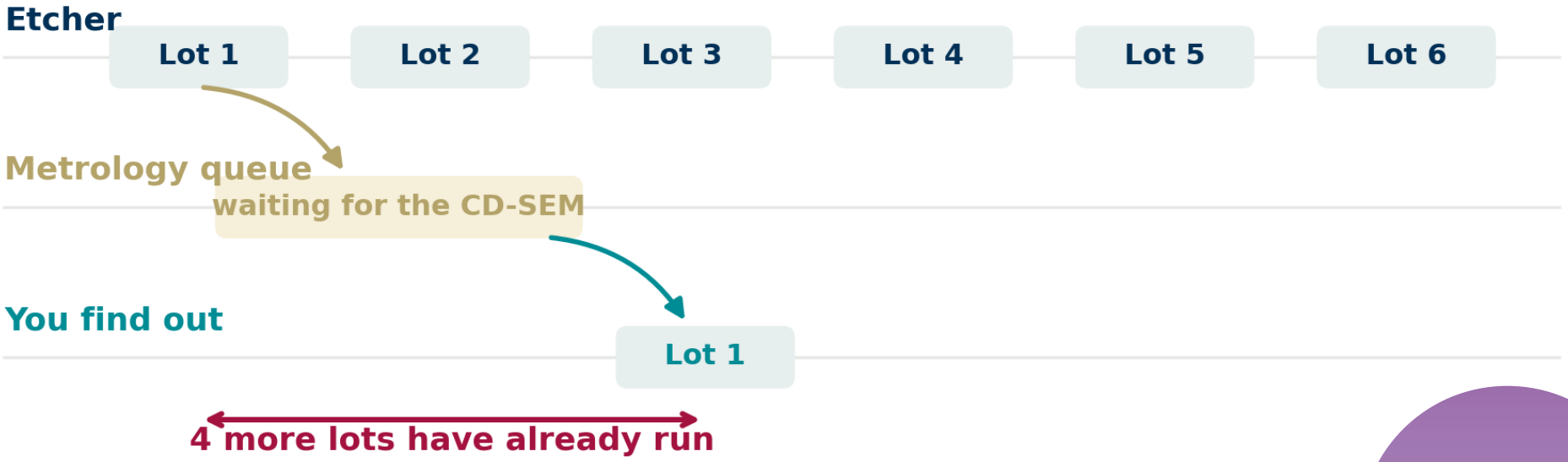
**None of these are metrology problems. They are all consequences of how few wafers carry a number.**



# Measurement delay, drawn to scale

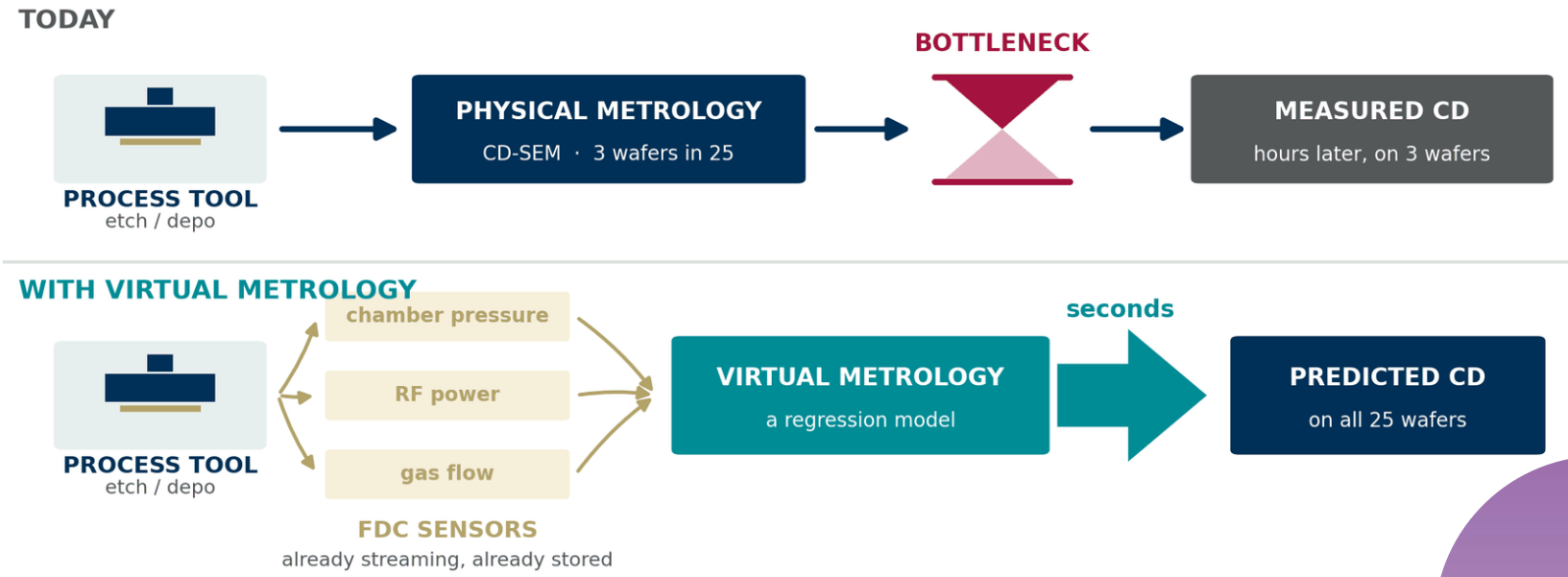
The delay matters more than the cost

**By the time Lot 1's number arrives, Lots 2 to 5 are finished.**

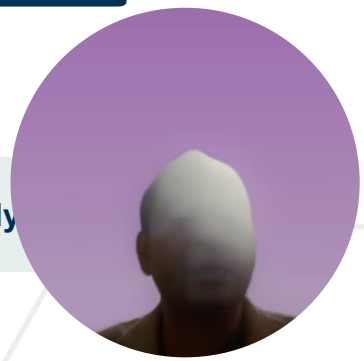



# So: virtual metrology

**Predict the measurement from data you already have.**




**Same tool, same recipe, same wafers. The only new thing is a model — and everything it needs is already**



# Three neighbours it is easy to confuse it with

**All three are real things. None of them is virtual metrology.**

## Inline metrology

A real sensor, in the tool, taking a real measurement — an interferometer watching etch depth, for instance. It produces measurements. Virtual metrology produces predictions of measurements.

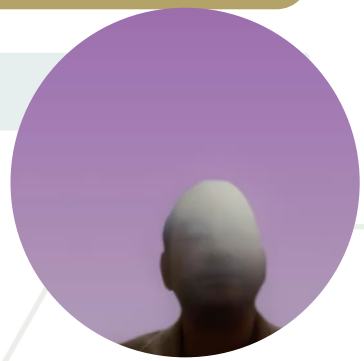
## Soft sensing

The control-theory term for inferring an unmeasured variable from measured ones, usually through a physical model. Virtual metrology is the data-driven, wafer-level case of it.

## Fault detection

Asks whether the run was healthy — a yes or no, or an anomaly score. Virtual metrology asks for a number in nanometres, which is a harder question and a more useful answer.

**Virtual metrology predicts a quantity, on every wafer, in the units the specification is written in.**



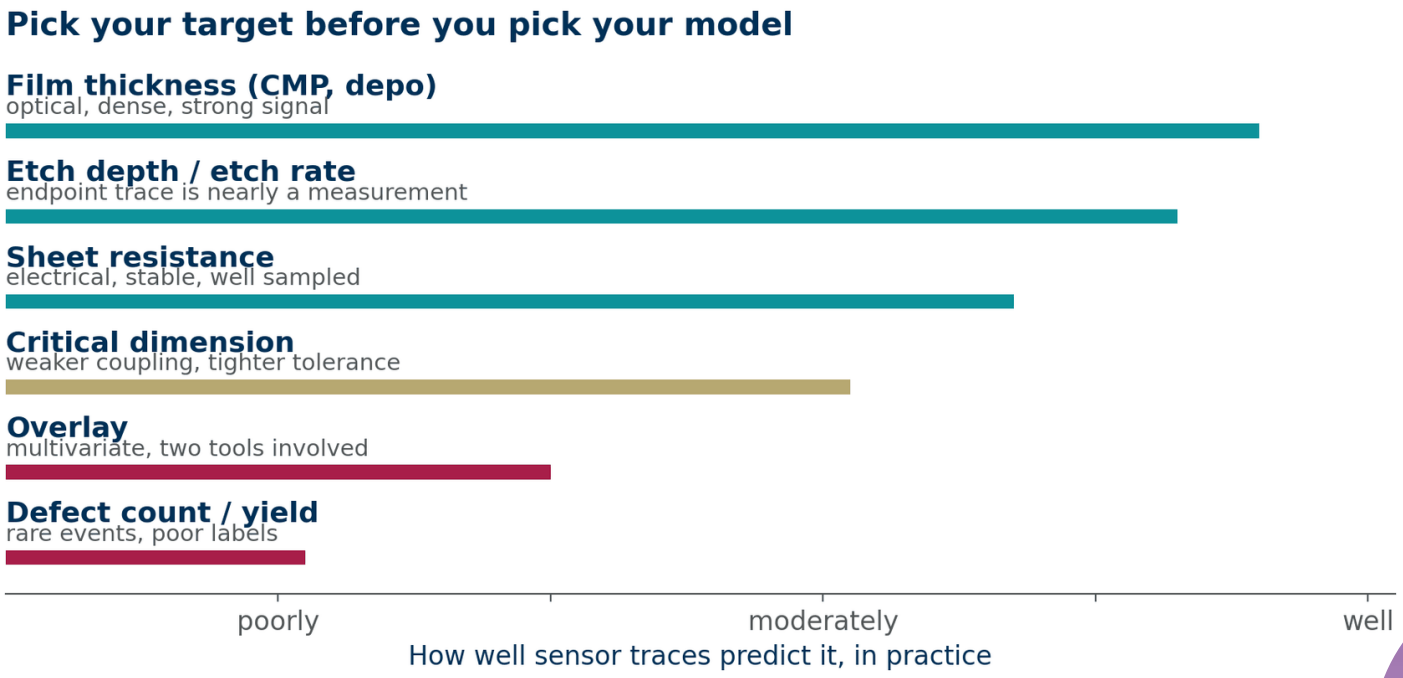
SEGMENT 2 OF 4

# What you predict, and what from

Choosing the target and assembling the table is most of the work. The modelling is the easy part.

# Not every measurement is equally predictable

**Where virtual metrology works well, and where it struggles**




**Thickness and etch depth are the standard first projects because the trace almost contains the answer al**

# The four families of input

**Everything on this slide is already being logged**

## Trace summaries

Statistics from each recipe step of each sensor: mean, standard deviation, slope, range, settling time. Hundreds of columns per run, and the bulk of the feature table.

## Recipe and context

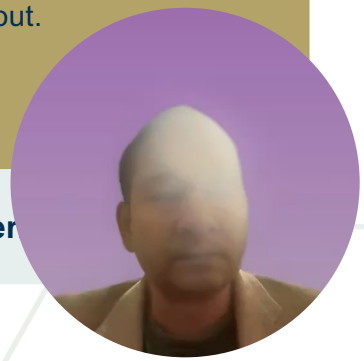
Setpoints as run, product and layer, chamber and tool identity, slot position. Cheap, categorical, and often more predictive than the sensors.

## Chamber state

Runs since clean, runs since PM, consumable age, idle time before the run. This is the tool's memory, and it drives slow drift.

## Incoming wafer state

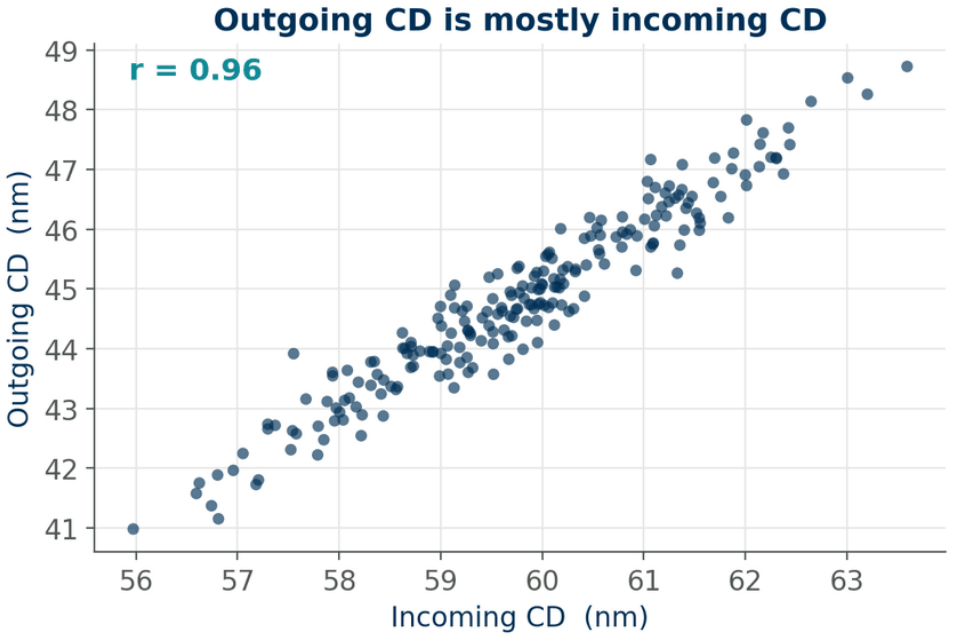
What this wafer measured at the previous step. Routinely the single strongest predictor, and routinely left out.

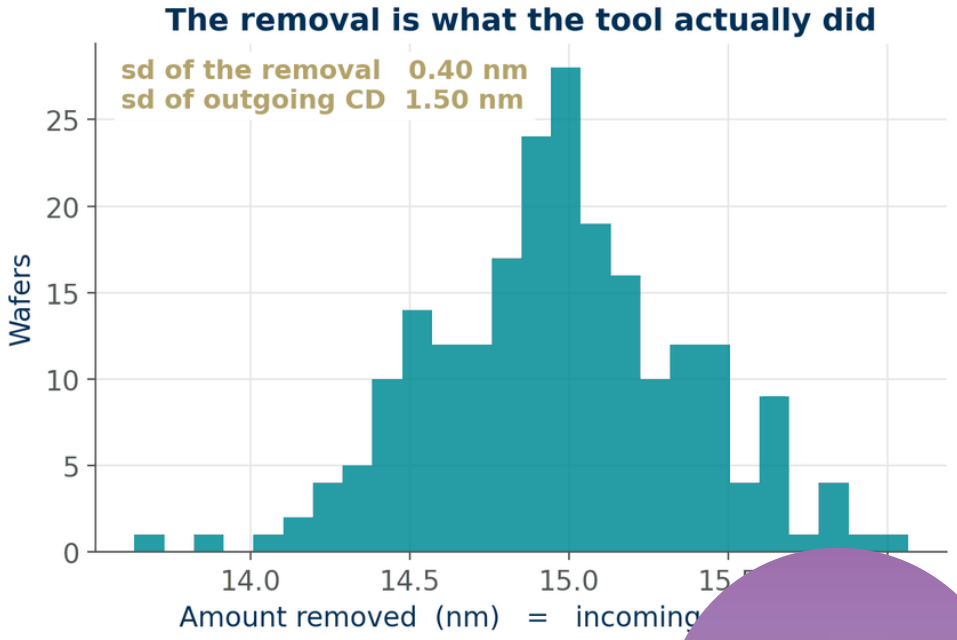


**The engineering effort is in joining these four sources onto one row per wafer. The modelling comes after**

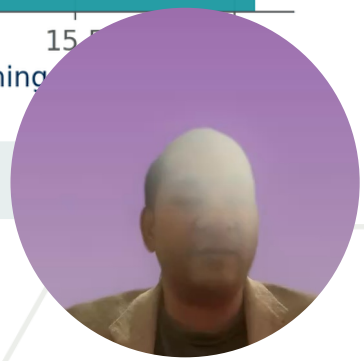
# Why incoming state matters so much

Outgoing CD is mostly a function of incoming CD




**Predict what the tool did — the removal — rather than what came out. It is a smaller, cleaner target.**



# Predict the change, not the value

**CD_out = CD_in − removal** and the tool only controls the removal

## Predicting CD_out directly

Your model has to reproduce the incoming distribution as well as the tool's behaviour. Most of the variance it is asked to explain has nothing to do with the chamber.

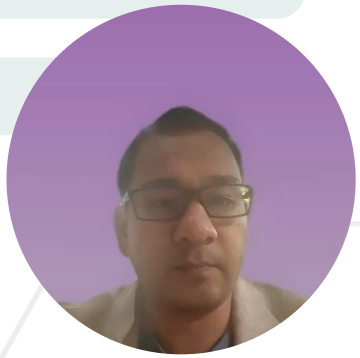
The sensor traces know nothing about incoming CD, so that variance is unexplainable and your $R^2$ absorbs the loss.

## Predicting the removal

The target is now exactly what the traces are a record of. Smaller spread, stronger coupling, and a physically meaningful quantity.

Add the incoming measurement back afterwards. Same answer, better model.

**Choosing the target well buys more accuracy than any change of algorithm.**



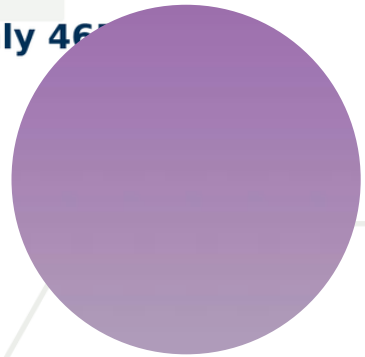
# The table you are actually building

One row per wafer, and the target present on some of them

| wafer_id | lot_id | chamber | runs_since_clean | f000 … f039 sensor | f040 … f059 lot-level | f060 … f119 nuisance | cd_measured |
|---|---|---|---|---|---|---|---|
| W00001 | L001 | A | 1 | … | … | … | 45.31 |
| W00002 | L001 | A | 2 | … | … | … | 45.02 |
| W00003 | L001 | A | 3 | … | … | … | — |
| W00004 | L001 | A | 4 | … | … | … | — |
| W00005 | L001 | A | 5 | … | … | … | 44.88 |

**One row per wafer. 1,200 rows, 120 feature columns, and the target present on only 46**

**Every dash is a wafer virtual metrology exists to fill in.**



# The join is where these projects stall

**Not the modelling — the plumbing**

## Wafer identity

Trace data is keyed by tool, chamber and timestamp. Metrology is keyed by lot and slot. Connecting them needs the tool log, and slot maps are not always right.

## Clock skew

The tool's clock, the MES clock and the metrology clock disagree by seconds to minutes. Join on the wrong one and you attribute a trace to the wrong wafer.

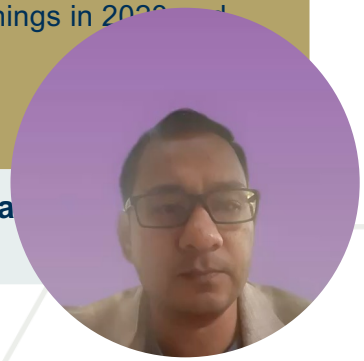
## Rework and reorder

A reworked wafer has two traces and one measurement. A lot that was split across chambers has two genealogies. Both quietly corrupt a naive join.

## Units and revisions

Sensor calibrations change, recipe revisions rename steps, and a column called 'pressure' means different things in 202[illegible] and 2025.

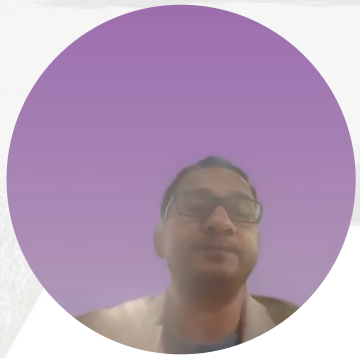
**Budget most of the project for this. A clean table with a simple model beats a dirty table with a sophistica[illegible] time.**



SEGMENT 3 OF 4

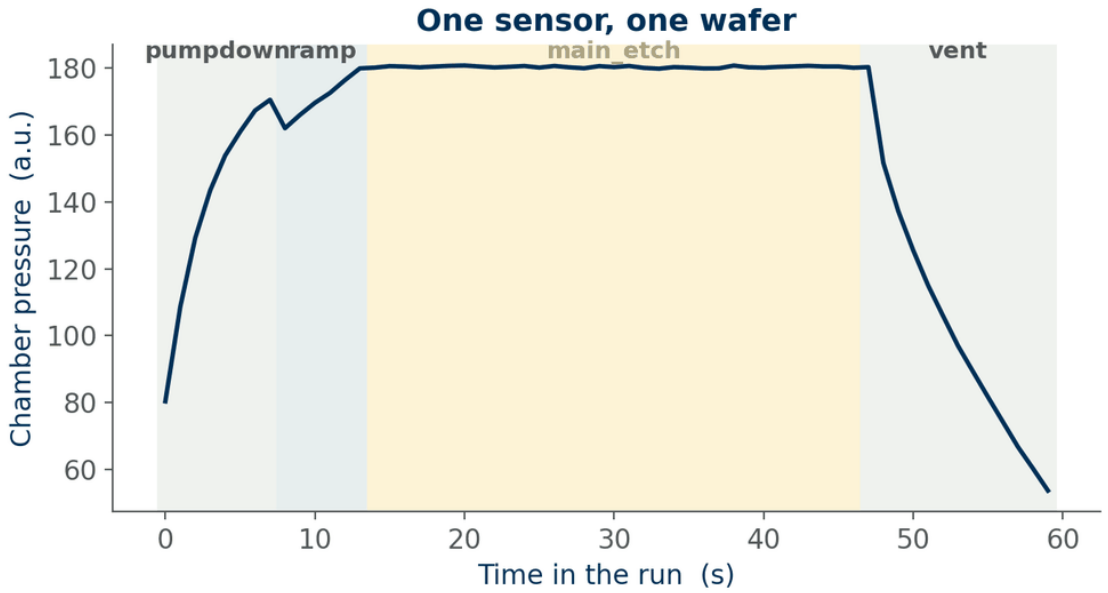
# From traces to features

Lecture 3 showed you the pipeline. Here is what it takes to make it work.



# Segment by step, then summarise each window

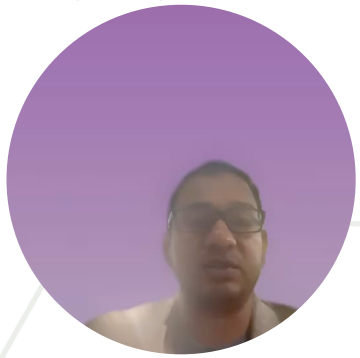
The recipe has phases, and they are not equally informative




**What actually goes in the feature table**

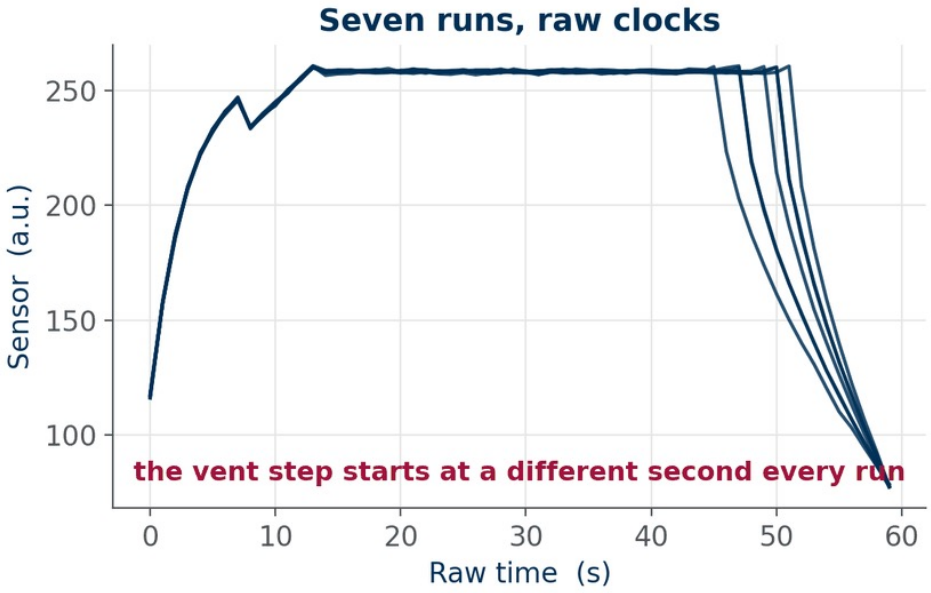
| | |
|---|---|
| **mean**<br>the level it held | **180.36** |
| **standard deviation**<br>how steady it was | **0.261** |
| **slope**<br>whether it drifted | **-0.0042** |
| **range**<br>the worst excursion | **0.959** |

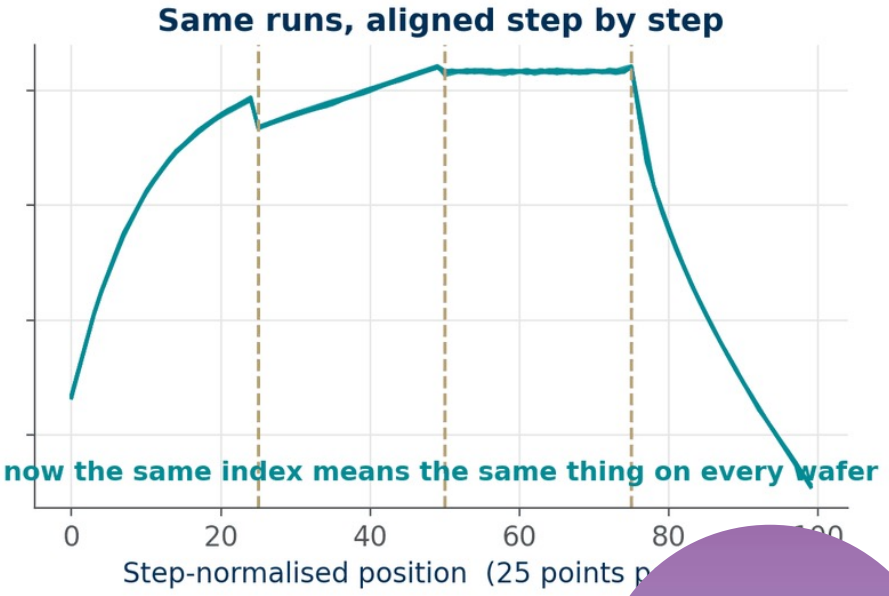
Four numbers per sensor, per recipe step



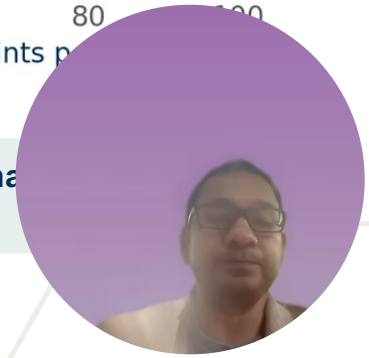
# Runs are not the same length

Which is why the same second means different things on different wafers



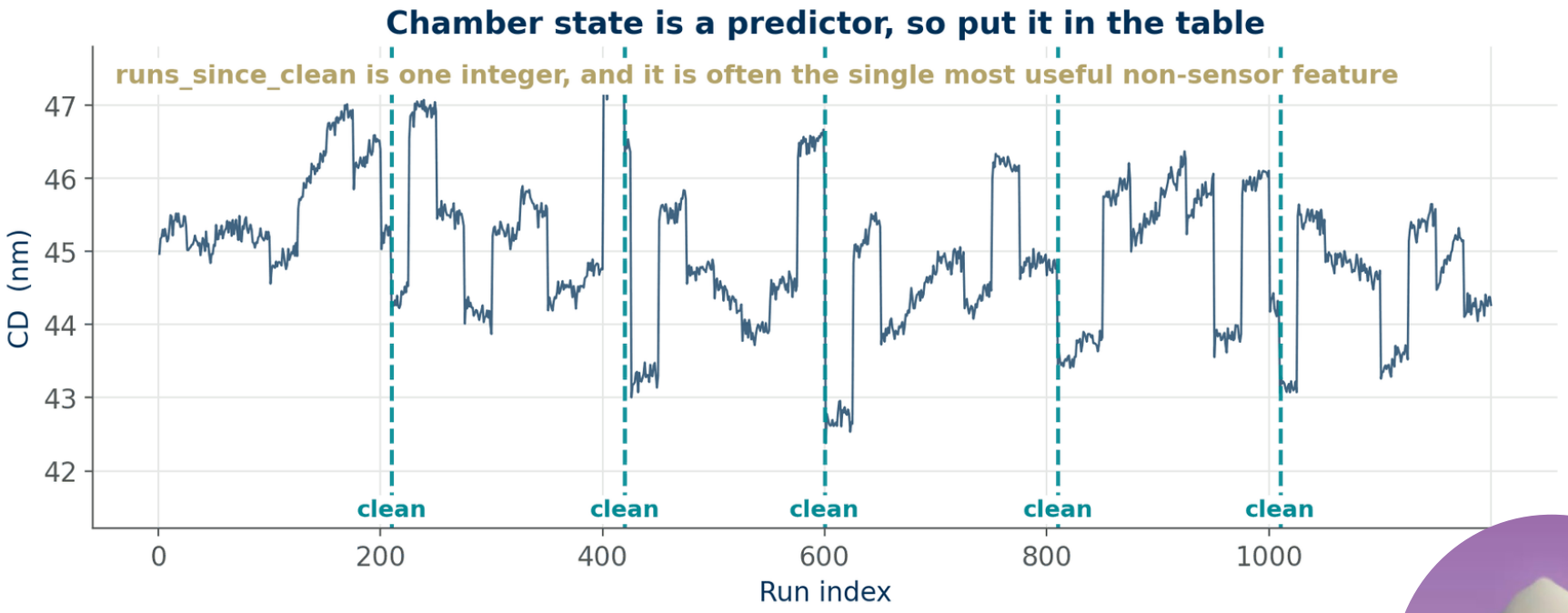




**Normalise within each recipe step, or align with dynamic time warping. Averaging across ragged runs ma**
**noise.**



# Chamber state belongs in the table

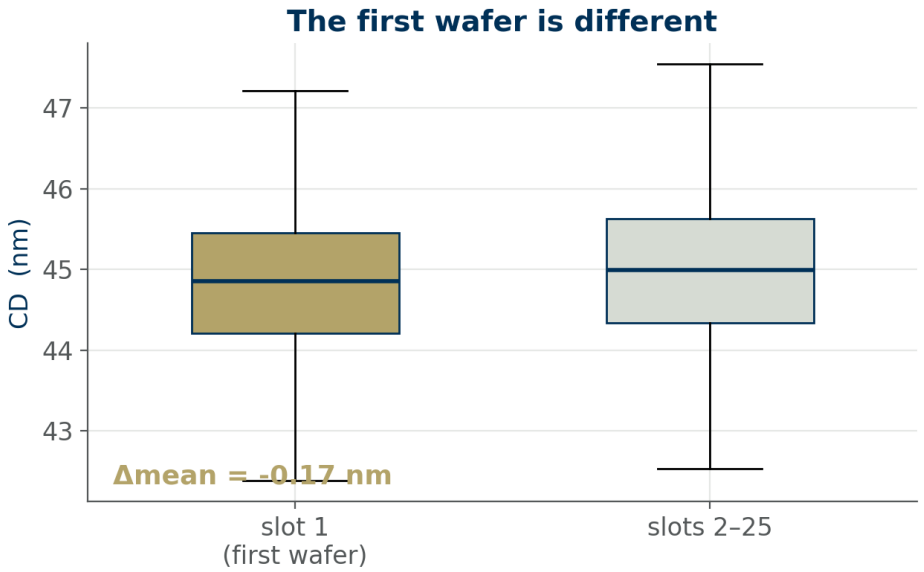
Runs since clean, drawn against the quantity you are predicting




**Two integers — runs since clean and runs since PM — often add more predictive power than fifty sensor**

# Two small features that earn their place

**Both are free, and both are commonly missing**




## Why slot 1 runs low

The chamber has been idle. Walls are cooler, the first wafer conditions the chamber for the rest. A single binary column captures it.

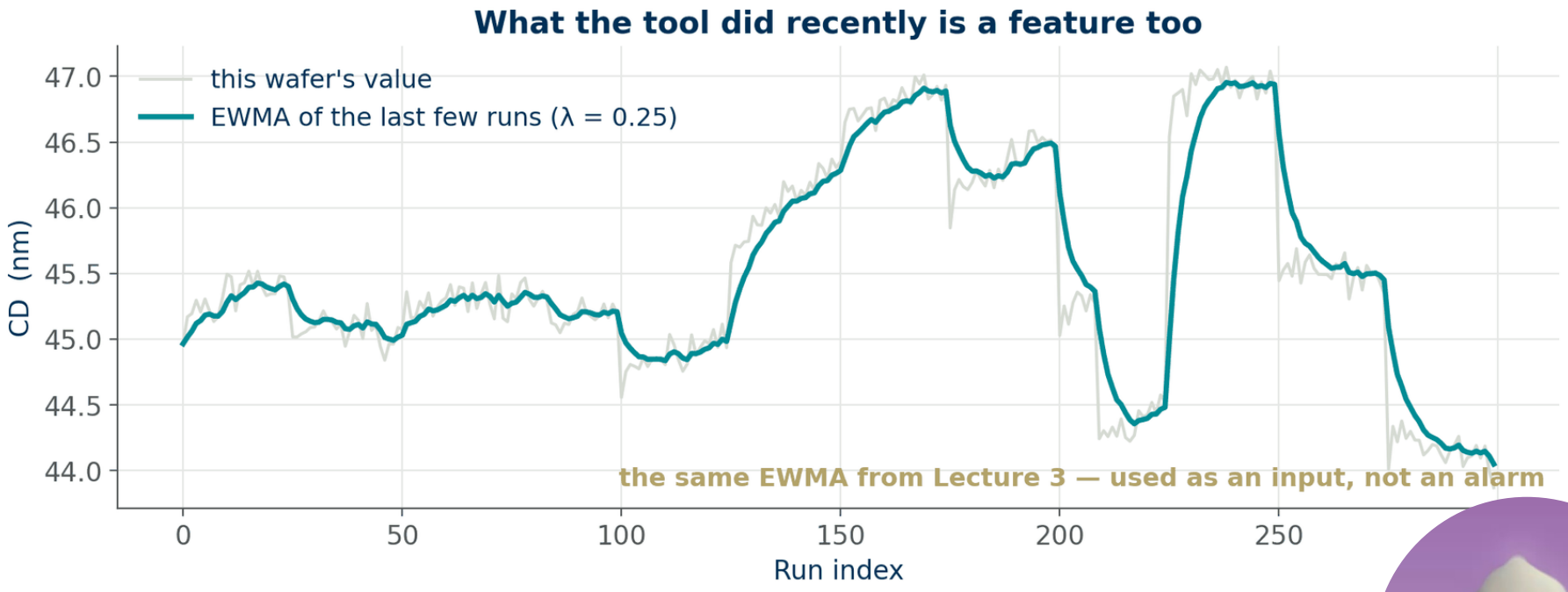
## And the slot number too

Position in the carrier correlates with thermal history and with handling order. Cheap to include, and it frequently shows up in the selected features.

**Ask a process engineer which conditions they think matter. Those are your features.**

# What the tool did recently is also a feature

The same EWMA as Lecture 3, used as an input rather than an alarm




**A wafer is not independent of the one before it. Encode that instead of hoping the model discovers it.**

# And now the problem

Multiply it out before you write any modelling code.

**300 sensors × 5 recipe steps × 6 statistics = 9,000 columns**

**9,000**

**candidate features**

and you can generate more in an afternoon

**465**

**labelled wafers**

eighteen months of production metrology

**19 : 1**

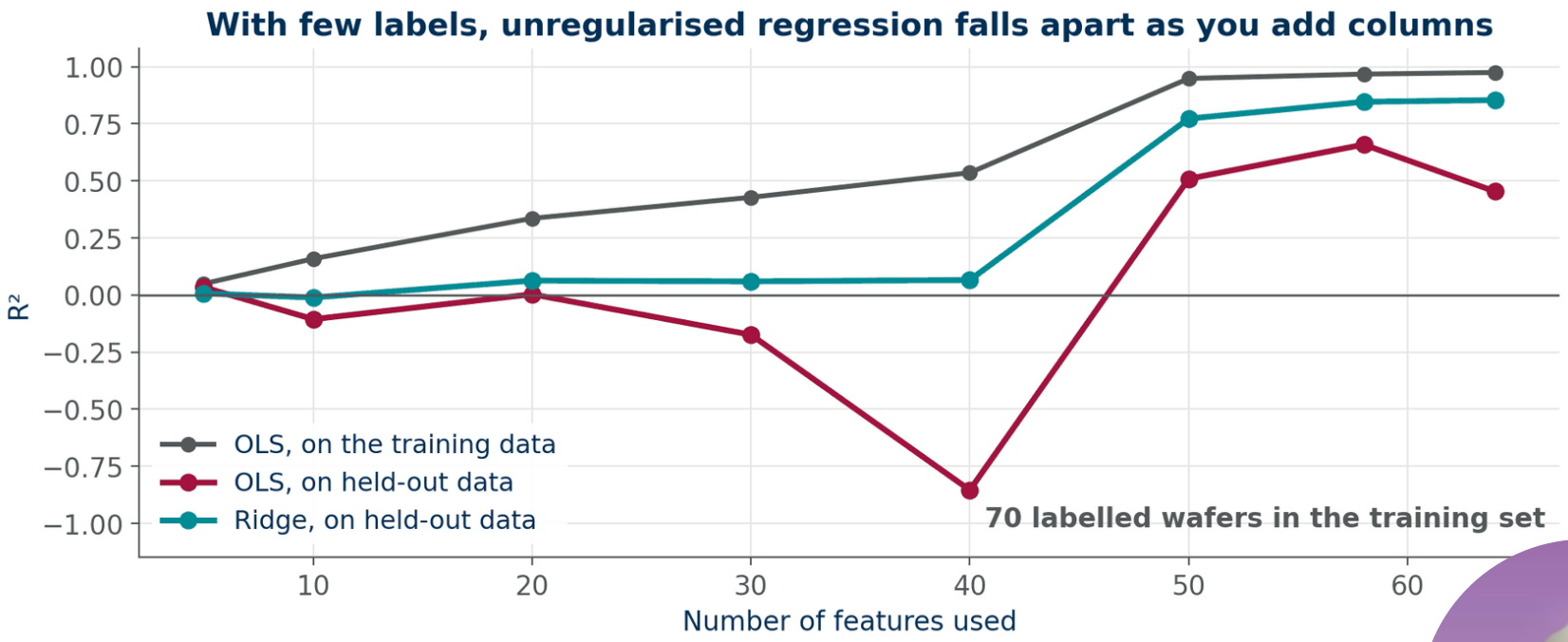
**columns per row**

the wrong way round

**Statisticians write this as p ≫ n. It is the defining constraint of virtual metrology, and it governs every cho**

# What p ≫ n actually breaks

Ordinary least squares, as the number of columns approaches the number of rows




**With more columns than rows there are infinitely many perfect fits, and the one you land on describes yo**

# Housekeeping that changes your answer

**Unglamorous, and each one has cost somebody a project**

## Scale everything

Pressure in millitorr and power in watts differ by orders of magnitude. Any penalised or latent-variable method is scale-sensitive, so standardise inside the cross-validation fold, never before it.

## Handle dead sensors

A sensor that sticks at its last value looks perfectly well behaved and carries zero information. Flag zero-variance columns on a rolling window, not once at the start.

## Remove aborted runs

A run that aborted mid-recipe produces a trace, a feature row, and nonsense. Filter on recipe completion before anything else.

## Keep the identifiers out

Wafer ID, timestamp and run index will predict beautifully and mean nothing. If a column encodes when rather than what it belongs in the split, not the model.

**One stuck sensor in a 9,000-column table is invisible by inspection and obvious to a variance check.**

SEGMENT 4 OF 4

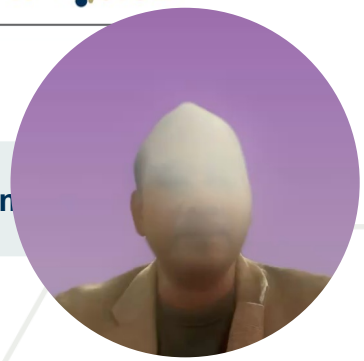
# Models, in the order you should try them

Start with the answer you can defend. Escalate only when the data asks you to.

# Start with the baseline you have to beat

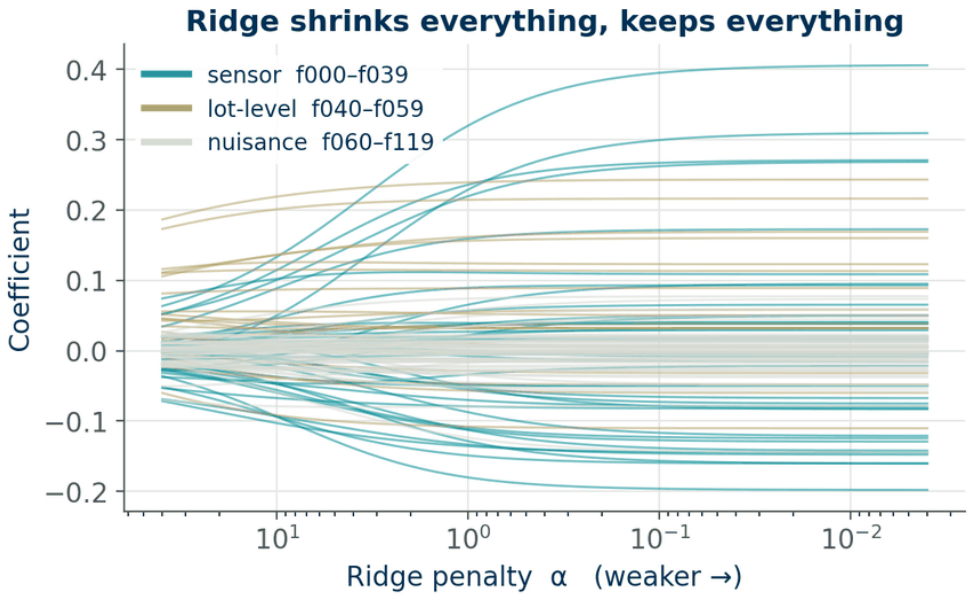
**Predict this wafer's CD with the previous measured wafer's CD**

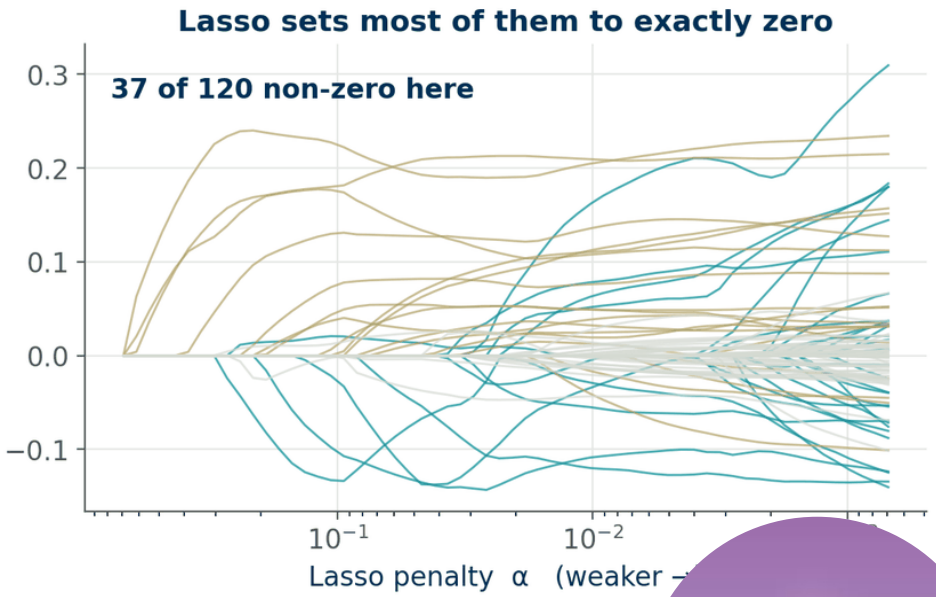

**RMSE 0.600 nm, R² 0.61, from one line of code and no features. Any model that cannot beat this is not a n**

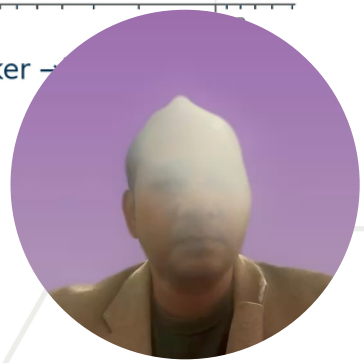


# Two ways to survive too many columns

Ridge shrinks coefficients toward zero; lasso sets most of them to exactly zero

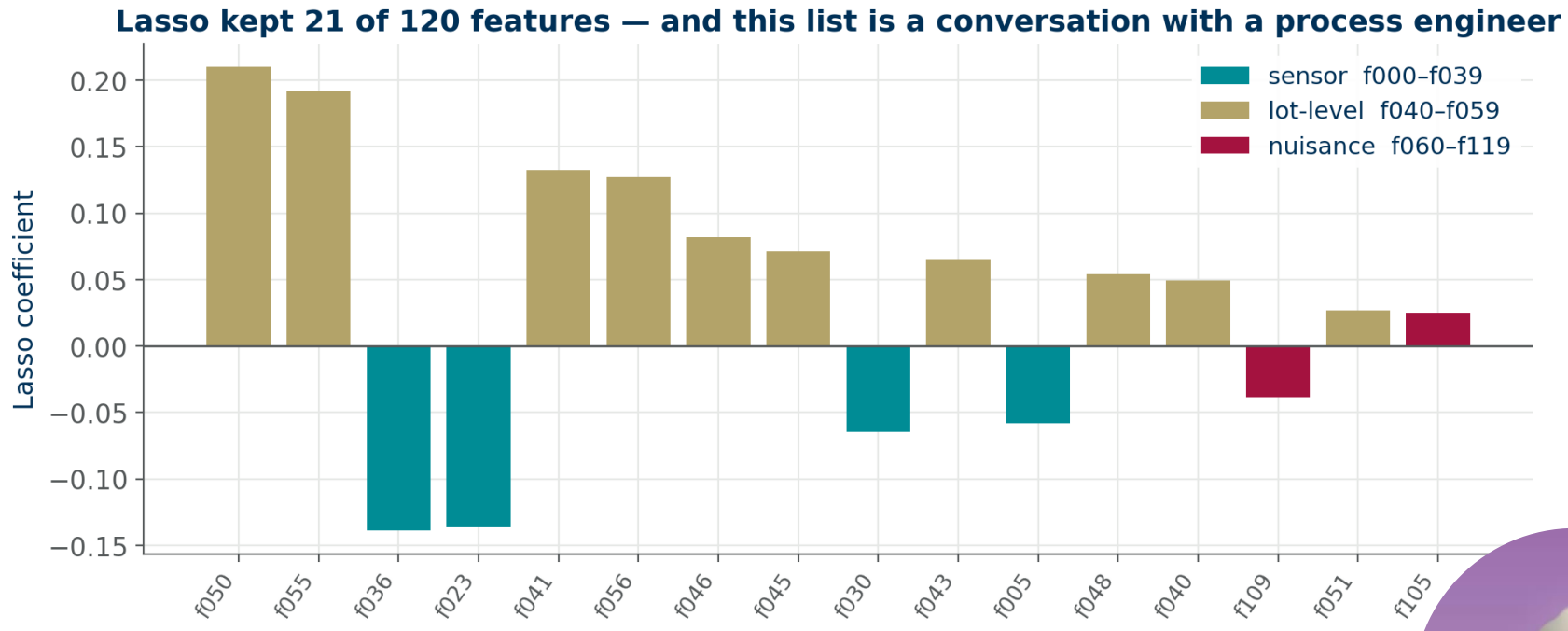


# Why lasso earns its keep in a fab

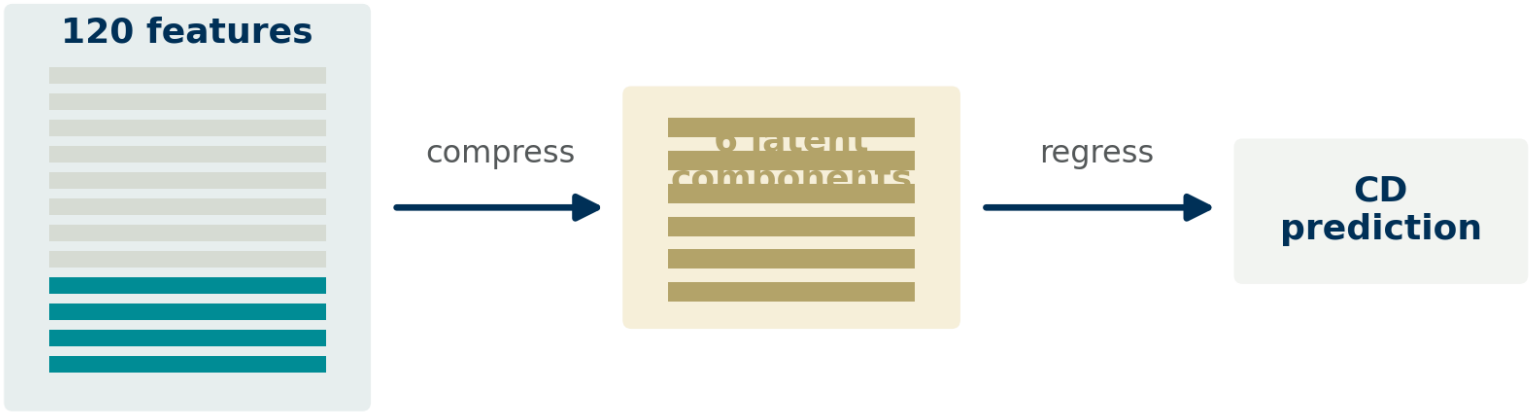
The selected features are a list you can take to a process engineer




**A model that names twenty sensors gets acted on. A model that uses nine thousand gets ignored.**

# PLS: compress first, then regress

The workhorse of virtual metrology, and the reason is p ≫ n

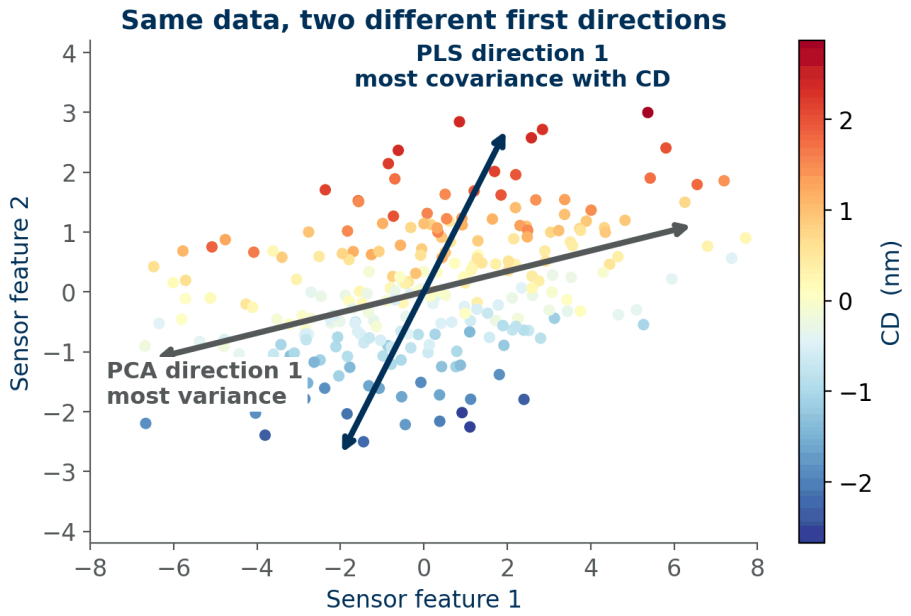



**PLS chooses the directions that vary with CD — not merely the directions that vary most.**

Instead of estimating 9,000 coefficients from 465 rows, estimate 8 — one per latent component.

# PLS against the PCA you already know

Same machinery, different objective




## PCA looks at X alone

It finds the directions of greatest variance in the sensors. Useful for monitoring — that is exactly the $T^2$ and SPE charts from Lecture 3 — because you do not need a target.

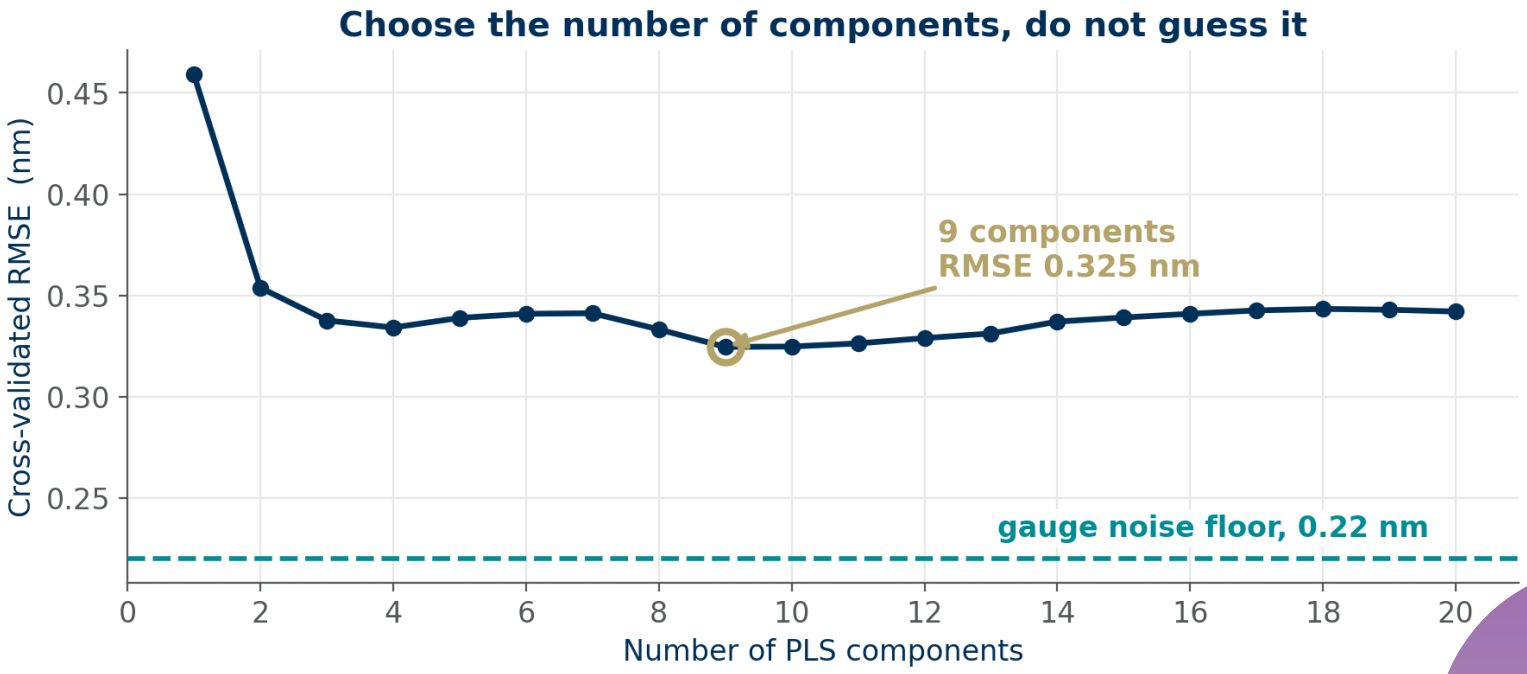
## PLS looks at X and y together

It finds the directions of greatest covariance with CD. A direction that dominates the sensor variance but says nothing about CD is discarded.

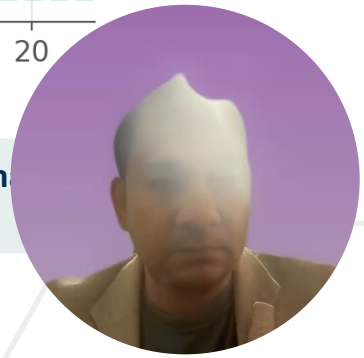
**For monitoring, use PCA. For prediction, use PLS.**

# Choose the number of components; do not guess it

Cross-validated RMSE against the number of latent components

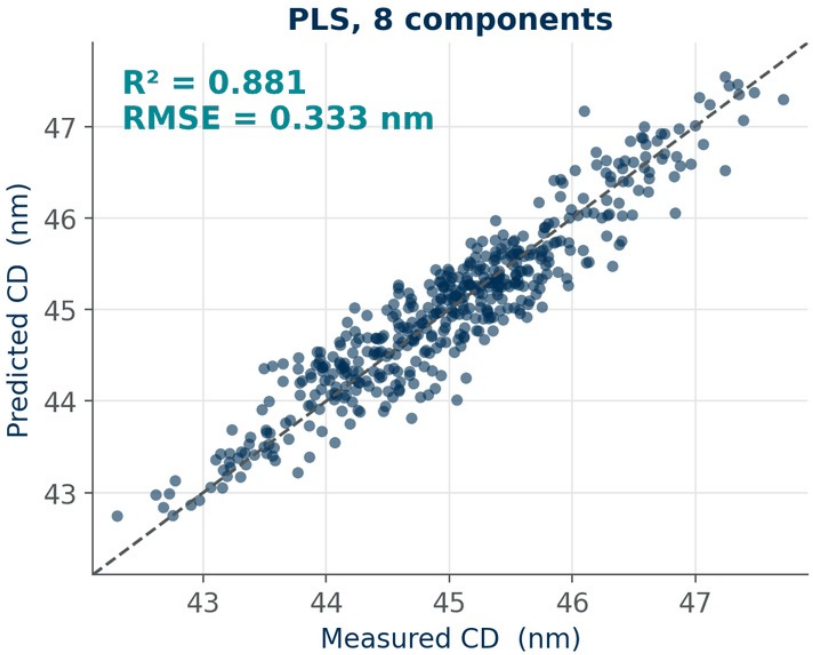


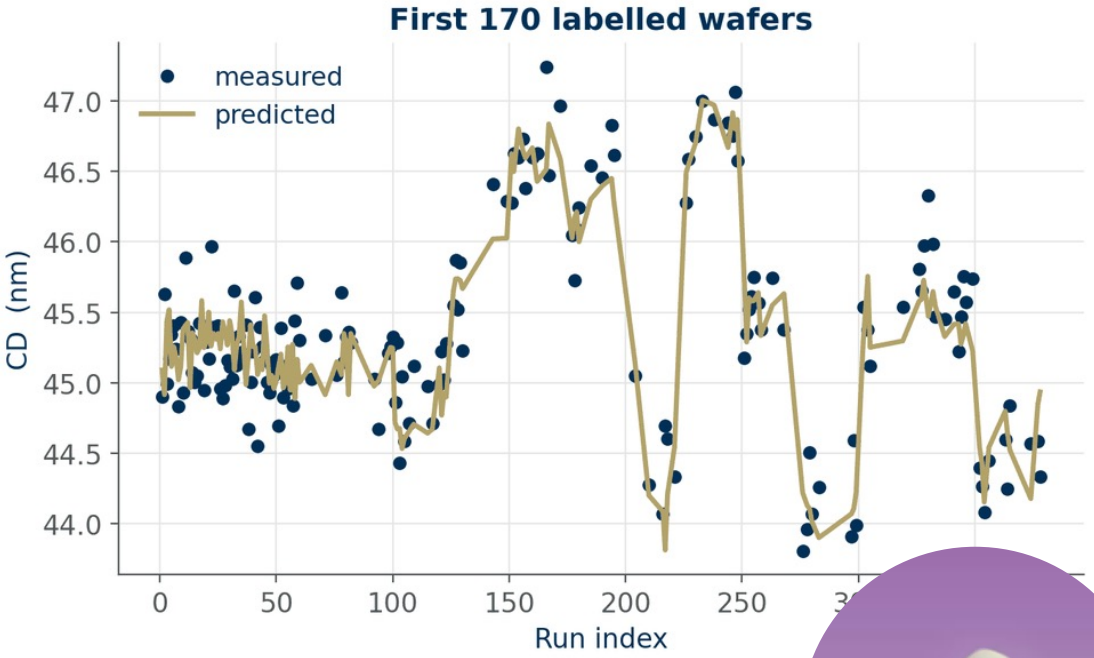

**The curve is flat from about 4 components onward, so take the simplest model in the flat region rather th minimum.**



# A first virtual metrology model

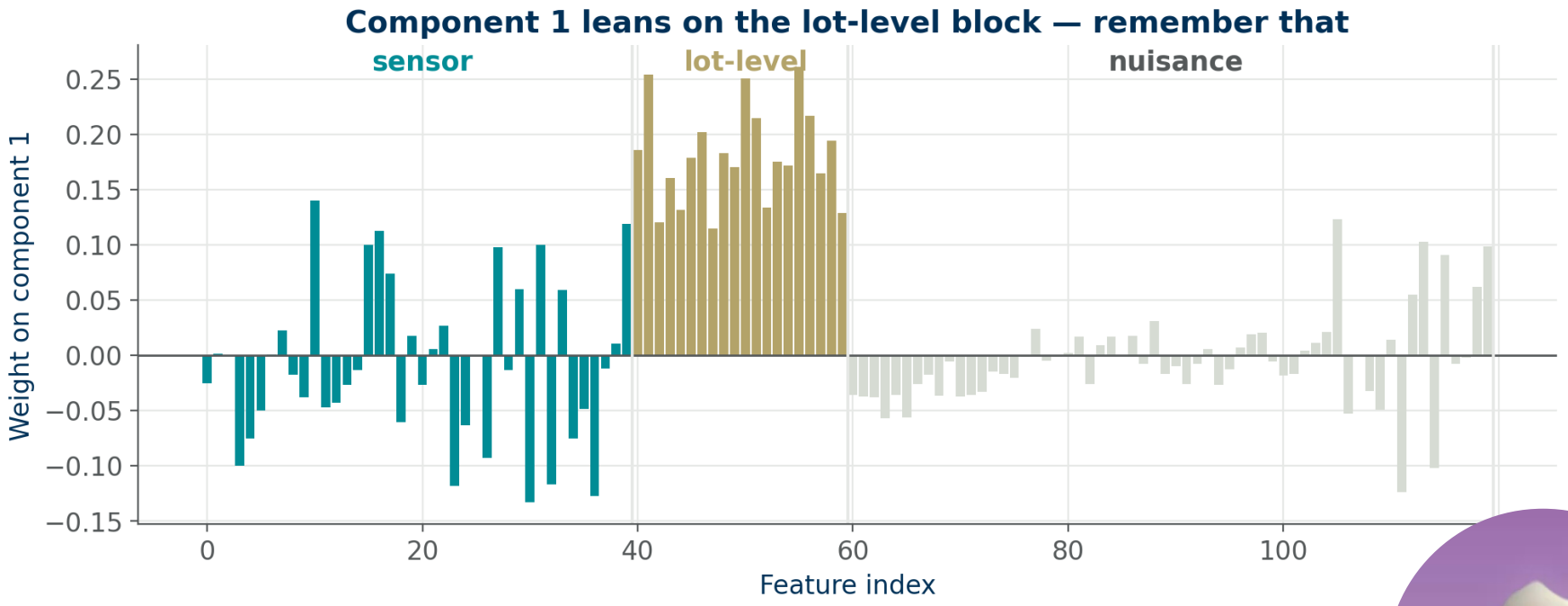
PLS with 8 components, random 5-fold cross-validation, 465 labelled wafers




# Read the weights before you celebrate

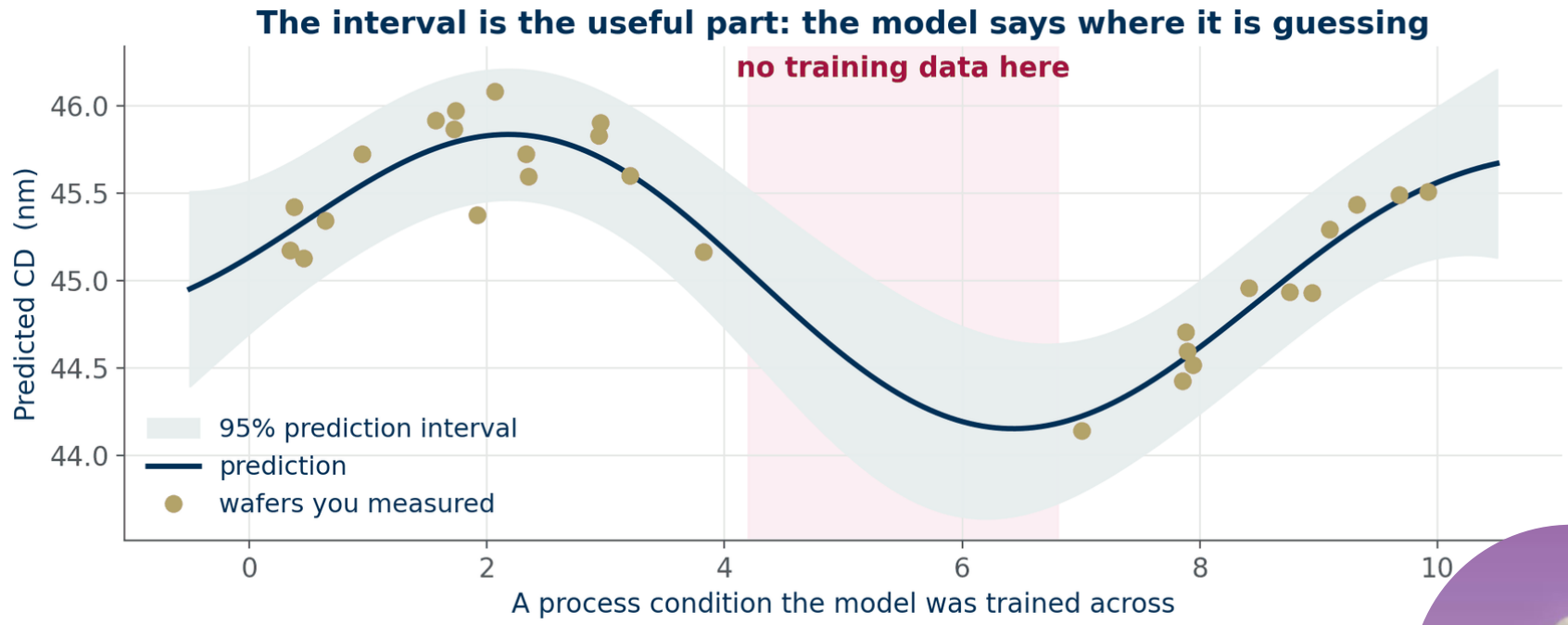
Which blocks of features the first component is leaning on




**The lot-level block dominates. Hold on to that observation — it is the first thing next lecture takes apart.**

# A prediction is more useful with an interval on it

Gaussian process regression, and the reason to consider it




**A model that knows where it is guessing can be told to abstain. That becomes an operational feature nex**

# The rest of the shortlist

**Two more options, and honest guidance on both**

## Gradient-boosted trees

Usually the strongest performer on a wafer-level feature table. Handles interactions and monotone-but-not-linear effects without being told.

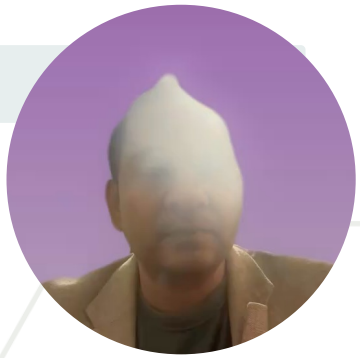
The cost: it cannot extrapolate. Outside the range it trained on it returns the edge value, confidently. In a drifting fab that happens more than you would like.

## Neural networks on raw traces

A 1-D convolutional network learns shape features — overshoots, settling, ramps — instead of you specifying window statistics.

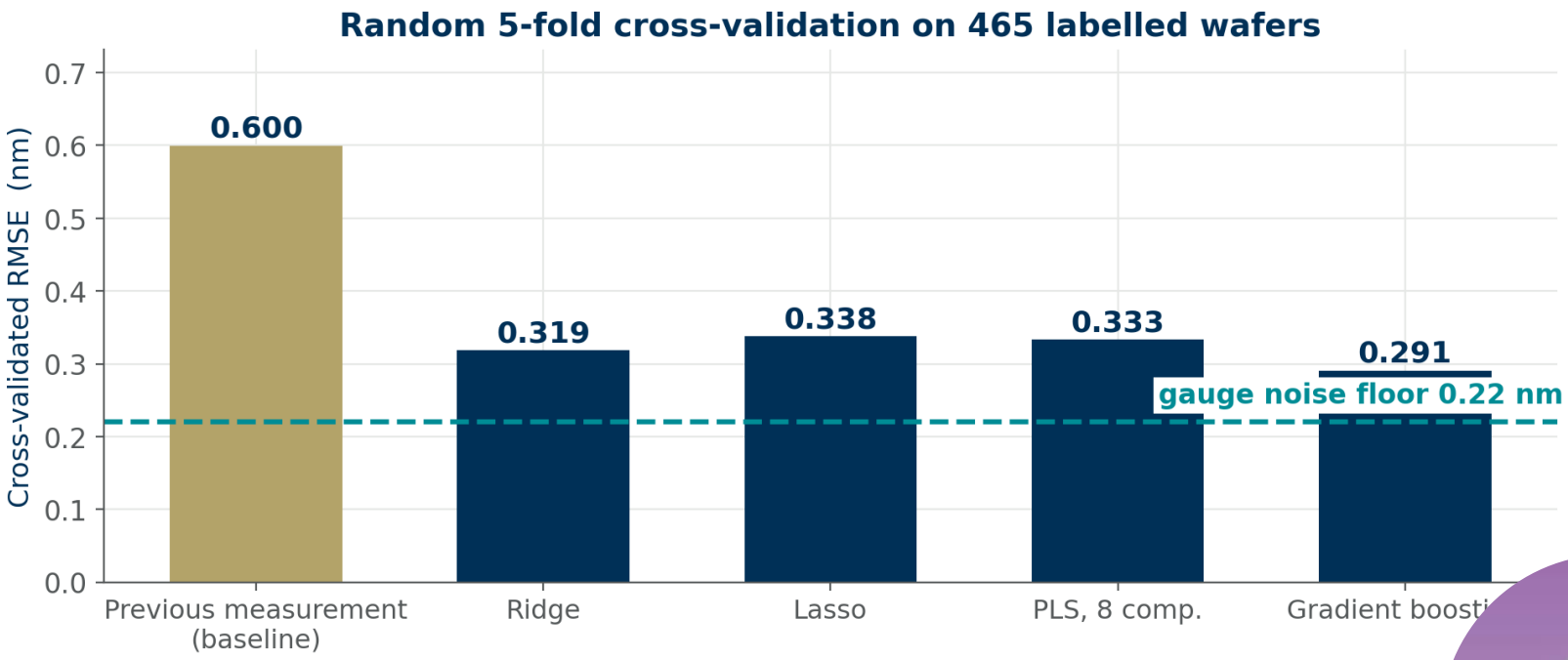
The cost: labels. With 465 measured wafers a CNN on raw traces will lose to PLS. Revisit it at tens of thousands of labels, or with self-supervised pretraining on the unlabelled runs.

**Model capacity is rarely the binding constraint. The label count is.**

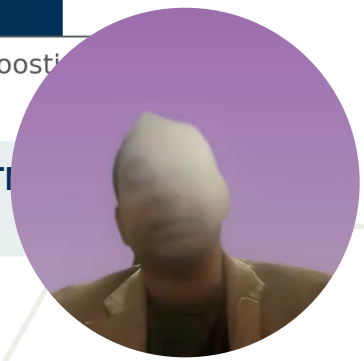


# Everything, side by side

Random 5-fold cross-validated RMSE on the same 465 wafers




**Every model beats the baseline. Gradient boosting reaches 0.291 nm, about 1.3 times the gauge noise. Th result.**



# So we have a working model

And one number on this slide is about to cause trouble.

**PLS, 8 components · $R^2$ = 0.881 · RMSE = 0.333 nm**

## What we would report

$R^2$ of 0.88 against a gauge floor of 0.22 nm, on 465 wafers, cross-validated. By the standards of most published virtual metrology work, this is a good model.

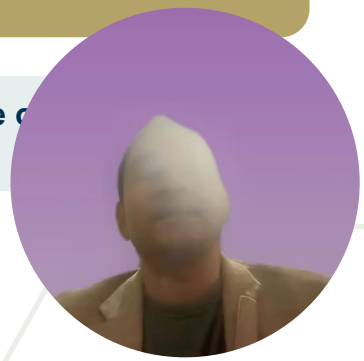
## What we did

Shuffled the 465 labelled wafers, split them into five folds, trained on four and tested on the fifth. Repeated five times. Standard practice.

## What is wrong with it

The shuffle. Wafers from the same lot ended up in training and in test, and wafers from next year's process ended up in training for wafers from last year's.

**Next lecture we evaluate this same model honestly. $R^2$ falls from 0.88 to 0.59, and the advantage over the c baseline falls from 42% to 3%.**



# Homework

**Datasets on the course site · due as posted**

1 **Build the features.** From vm_traces_subset.csv, segment each run by recipe step and compute mean, standard deviation, slope and range per step per sensor. Compare your columns against the supplied vm_features.csv.

2 **Beat the baseline.** Compute the previous-measurement baseline RMSE. Then fit ridge, lasso and PLS, and report random 5-fold RMSE for each.

3 **Choose the components.** Select the number of PLS components by cross-validation. Justify choosing the simplest model in the flat region.

4 **Read the model.** List the ten features with the largest lasso coefficients. Which block do they come from, and what does that tell you?

5 **Find the error floor.** Given a gauge σ of 0.22 nm, what is the smallest RMSE any model could achieve on these labels? How close did you get?

6 **One dead sensor.** Find the column that stops varying, and identify the run at which it stops.

**Keep your notebook. Next lecture you will re-evaluate the same model and the numbers will change.**

# References and pre-reading

- **Virtual metrology, surveys.** Start with a recent review of virtual metrology in semiconductor manufacturing; the field's vocabulary is settled and the surveys are a fast way in.
- **Partial least squares.** Wold, Sjöström and Eriksson, *PLS-regression: a basic tool of chemometrics* (2001). Short, readable, and the source of the framing used today.
- **Regularisation.** Hastie, Tibshirani and Friedman, *The Elements of Statistical Learning*, chapter 3 — ridge, lasso and why $p \gg n$ changes the problem.
- **Gaussian processes.** Rasmussen and Williams, *Gaussian Processes for Machine Learning*, chapters 1 and 2, for the predictive variance argument.
- **Fault detection and classification data.** Any of the public FDC or SECOM datasets, for practice on a table with the same shape as today's.
- **Before next lecture:** re-read the gauge R&R segment of SPC Lecture 3. Next lecture leans on $\sigma^2_{observed} = \sigma^2_{process} + \sigma^2_{measurement}$ applied to the training labels.

Georgia Tech