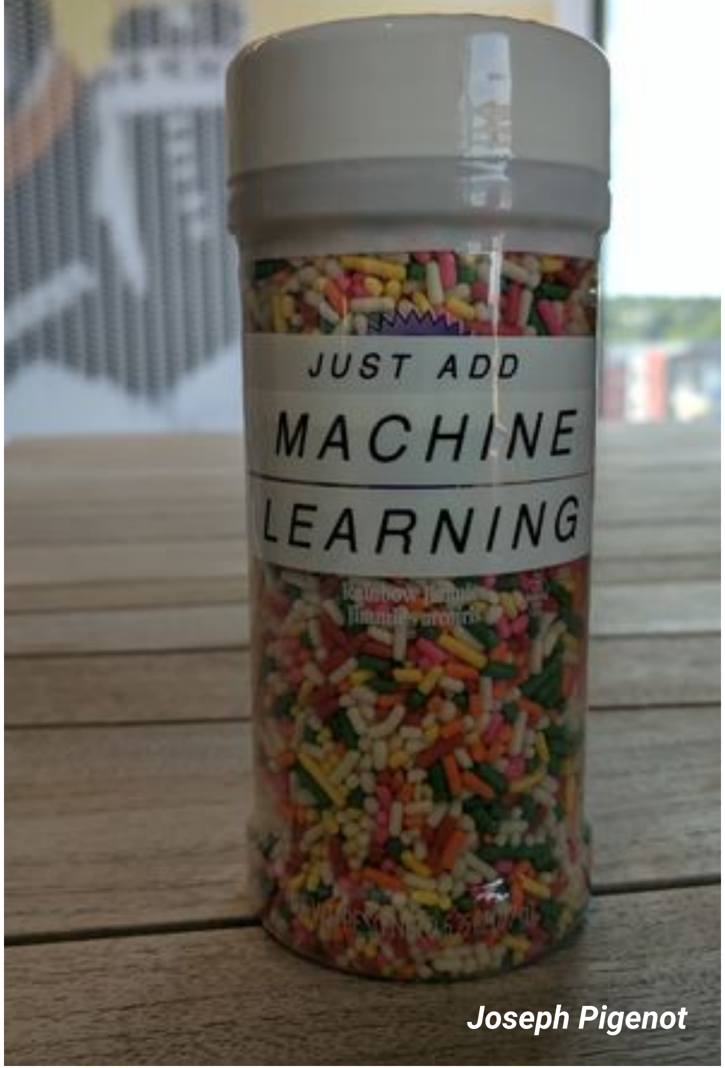
Demystifying ML in Production

Reasoning about a large-scale ML platform

Mary McGlohon
(she/her)
marymc@google.com

Machine learning is treated as magic



Productionizing ML is not magic

The “magic” part of ML is just another limited-observability system. Most of the same principles apply.

Today's talk

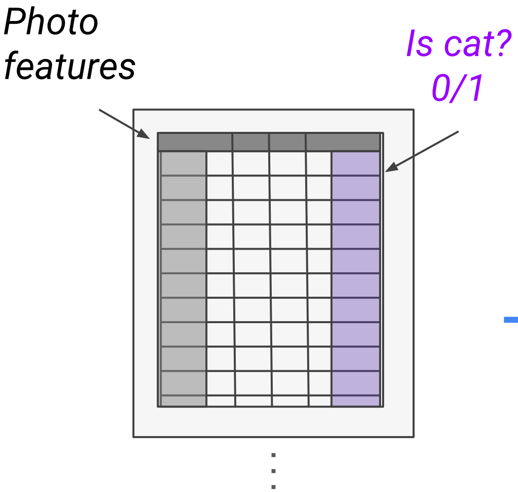
- What (one) ML production platform looks like
- What failure looks like
- 4 things to do to manage risk

4 things you can do for more reliable ML

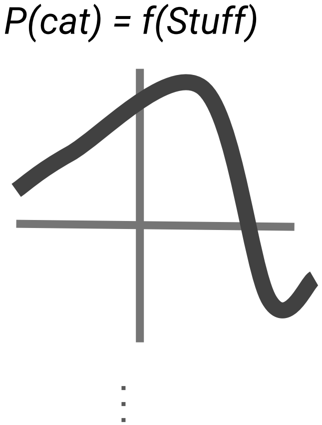
1. Make failure obvious
2. Validate production changes
3. Clarify data integrity requirements
4. Handle pipeline backlogs

Design of (one) ML platform

ML on one machine

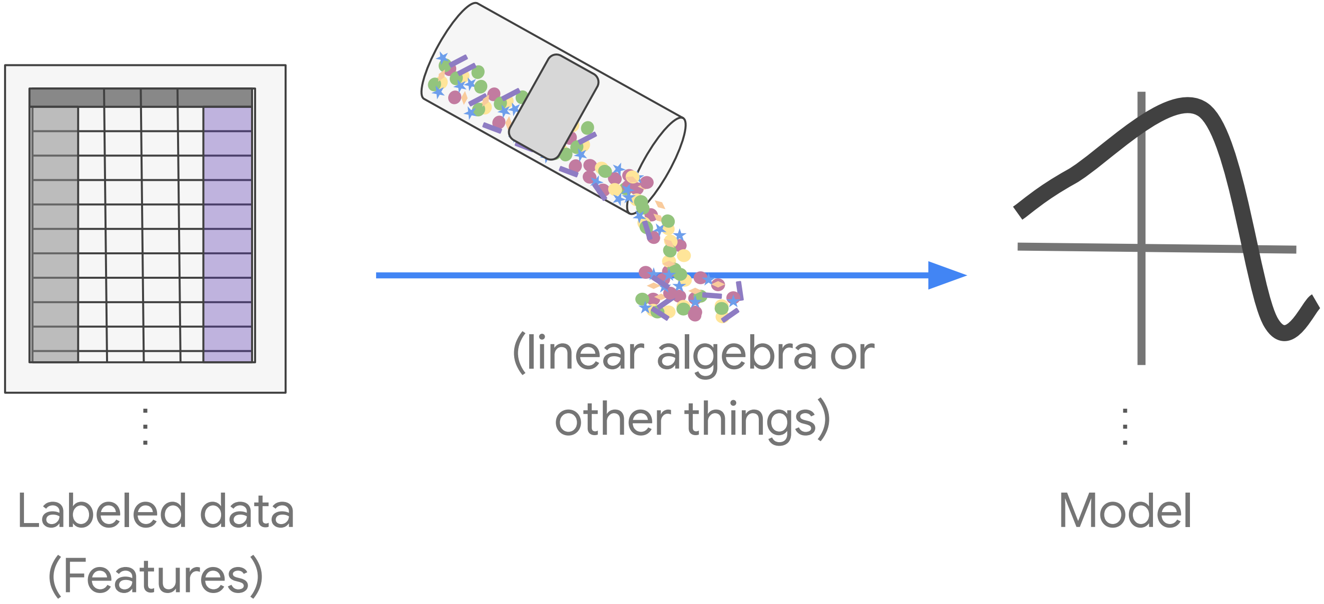


Labeled data
(Features)

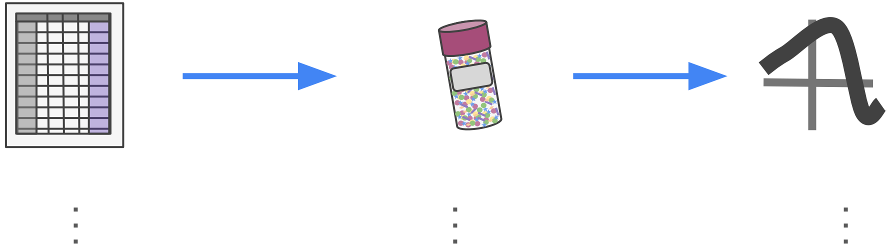


Model

ML on one machine



ML in production

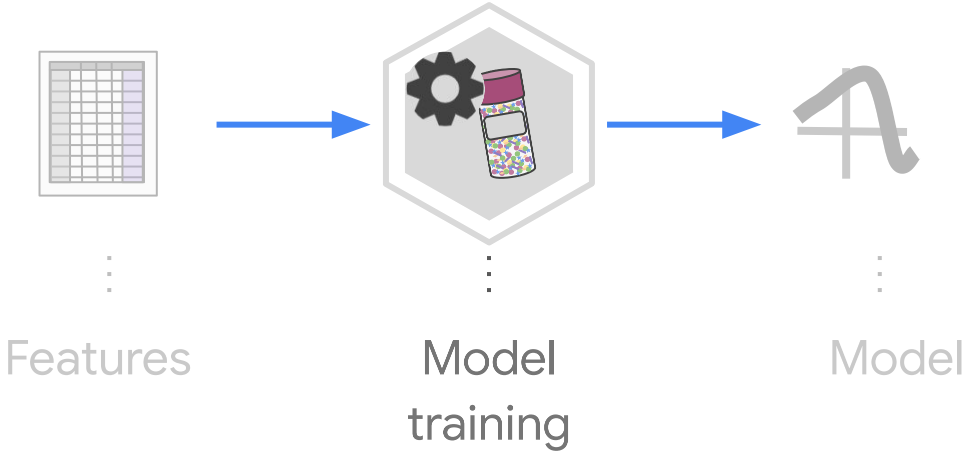


Features

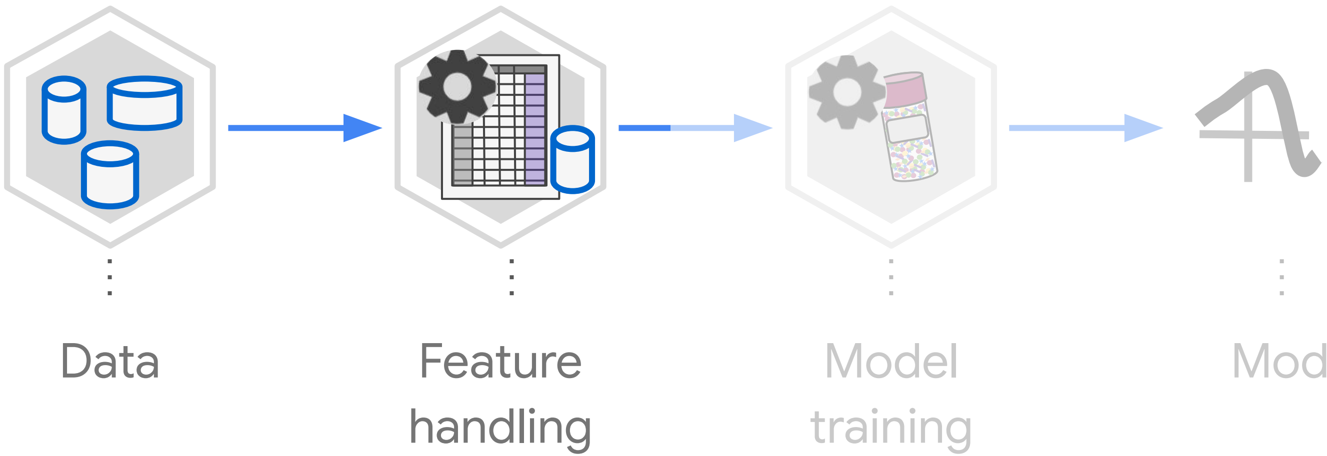
Model training

Model

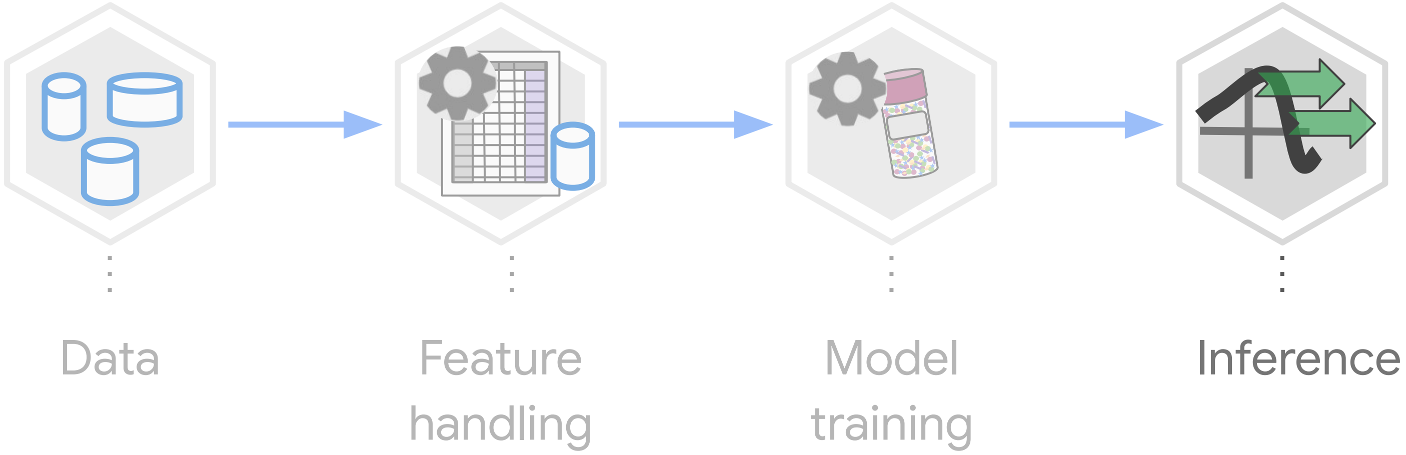
ML in production



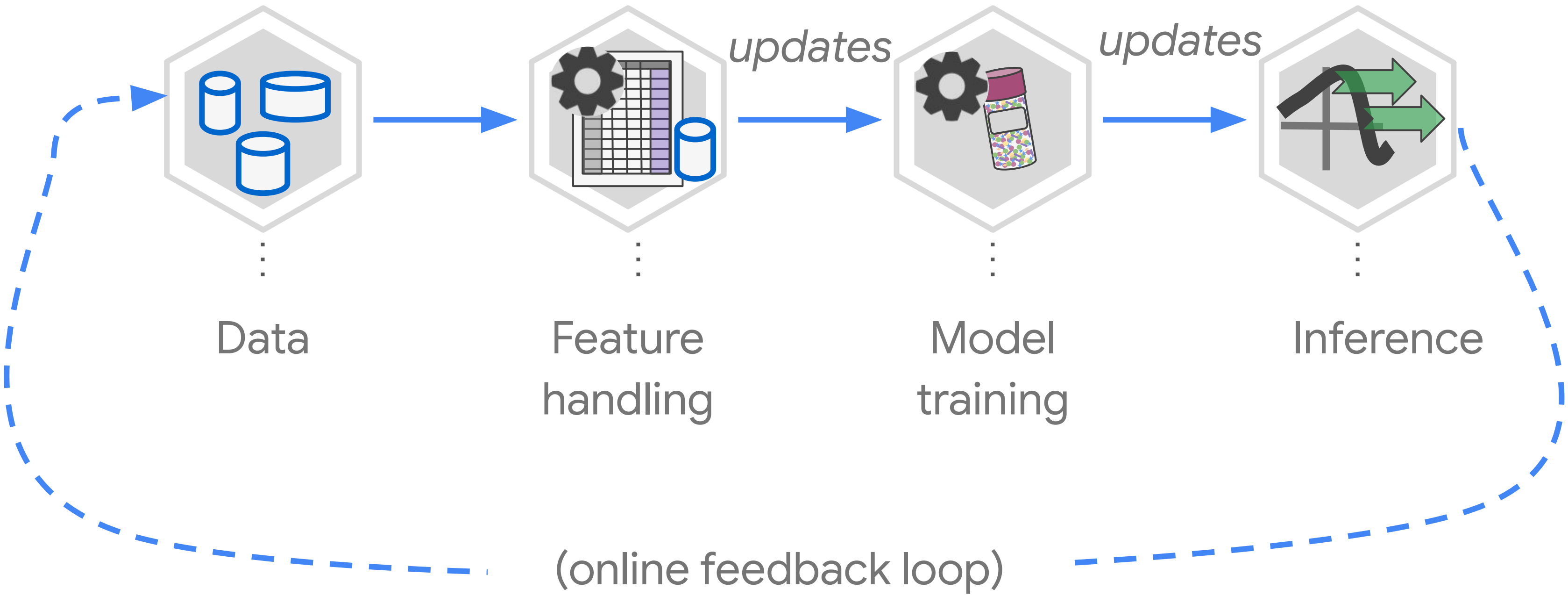
ML in production



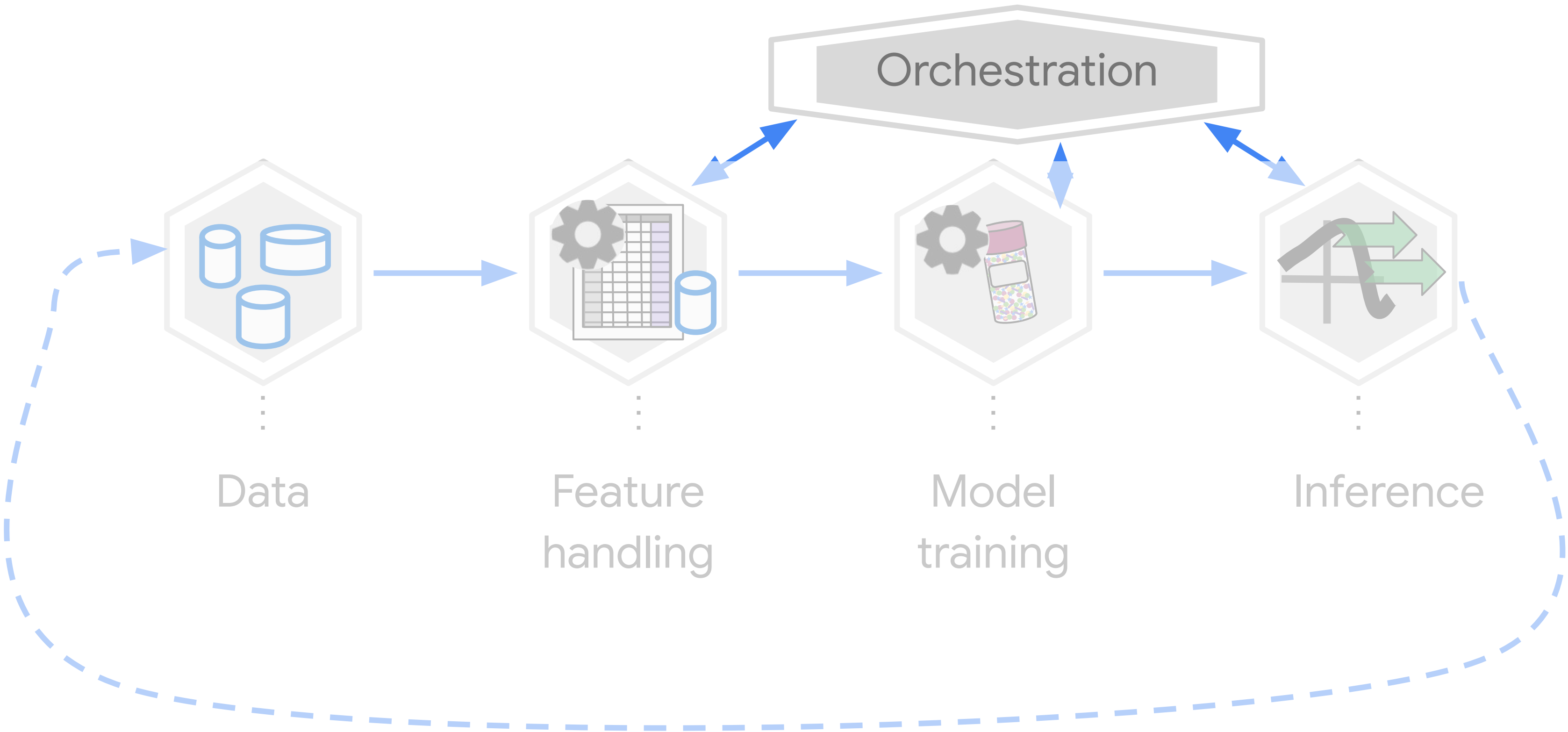
ML in production



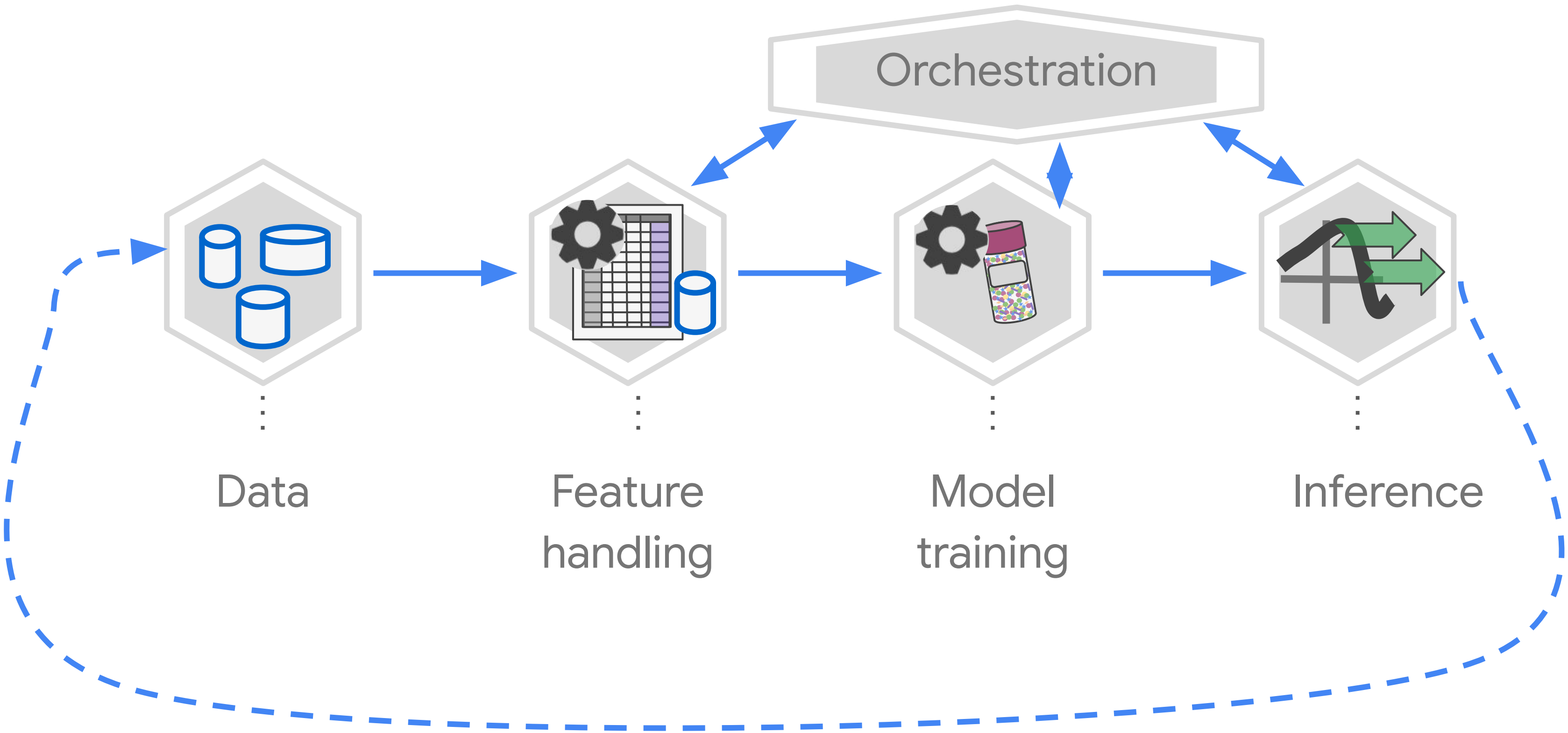
ML in production



ML in production



ML in production



What makes ML in prod interesting

A lot of data dependencies

That's, like, the whole point. But, it's kind of messy

Pipeline of pipelines

ML pipelines are sometimes dependent on each other, may share feature data

Atypical workloads, at scale

Feature processing can be very I/O heavy; training can be very compute-heavy. This gets interesting with many models and a large amount of data.

When something goes wrong

What goes wrong?

- Usually not ML-specific things:
  - incorrectly validated rollout
  - load balancing / overload issues
  - unexpected / error-prone interactions between systems

- See: Papasian and Underwood, “How ML Breaks: Fifteen years of ML production pipeline outages and insight”, OpML 2020
  - Root cause analysis of 15 years of ML system postmortems:
    - 30% were “inherent to ML”
    - 40% were “inherent to distributed systems”

What goes wrong?

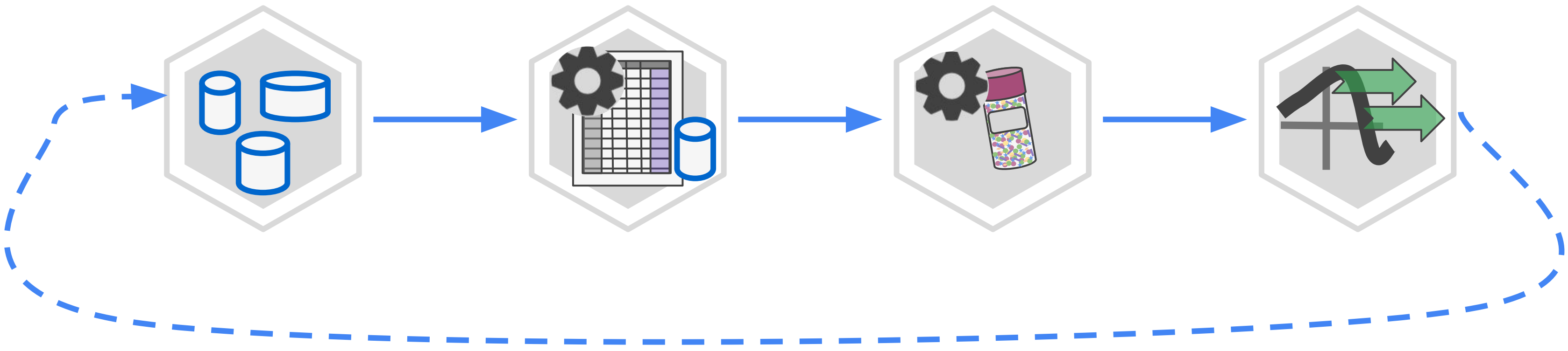
- Successful risk mitigation considers both **general best practices**, and **contextual best practices** based on what you know about your systems.

- Our ML platform is a set of distributed, **data-intensive**, **pipeline** systems.

4 things for more reliable ML

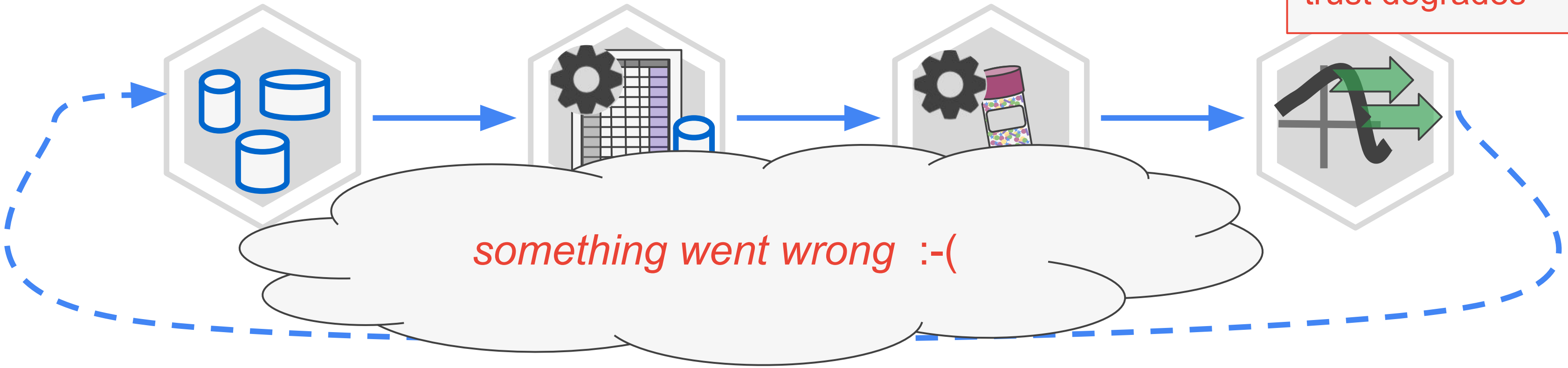
1. **Make failure obvious**
2. Validate production changes (binaries + data)
3. Clarify data integrity requirements
4. Handle pipeline backlogs

ML outages from the outside

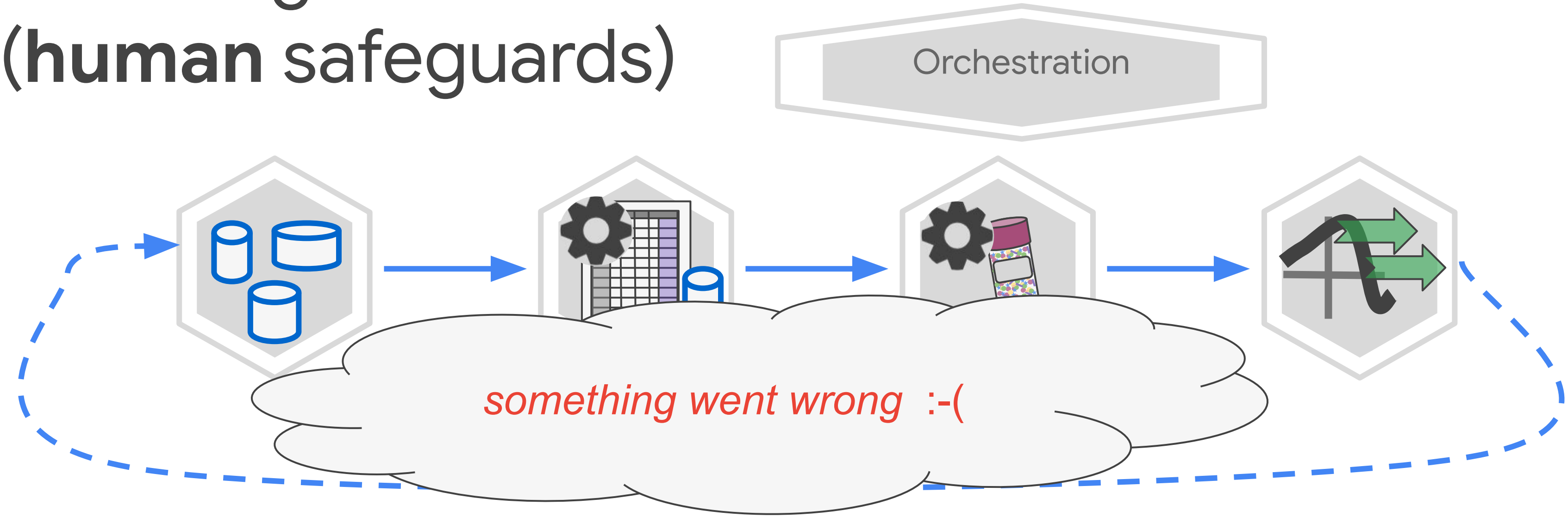


ML outages from the outside (no safeguards)

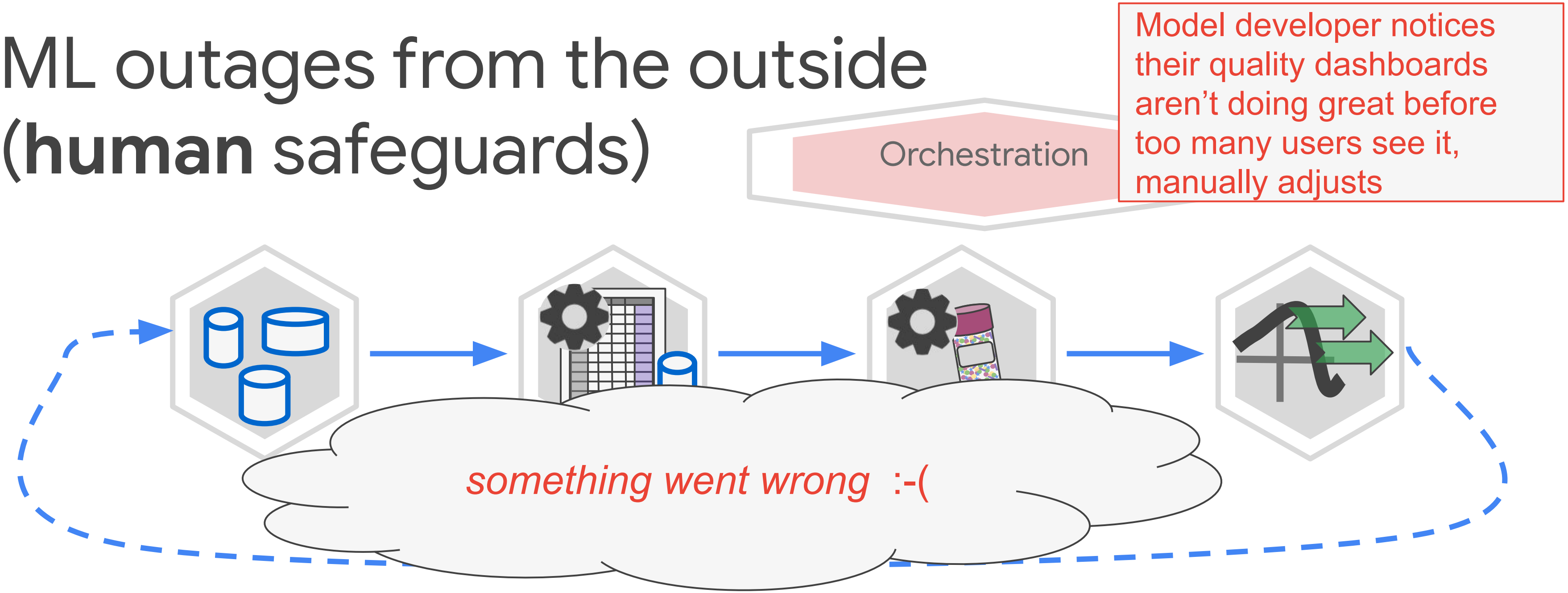
No new models (or bad models) go to serving, inferences get worse, eventually user trust degrades



ML outages from the outside (**human** safeguards)



ML outages from the outside (**human** safeguards)



4 things for more reliable ML

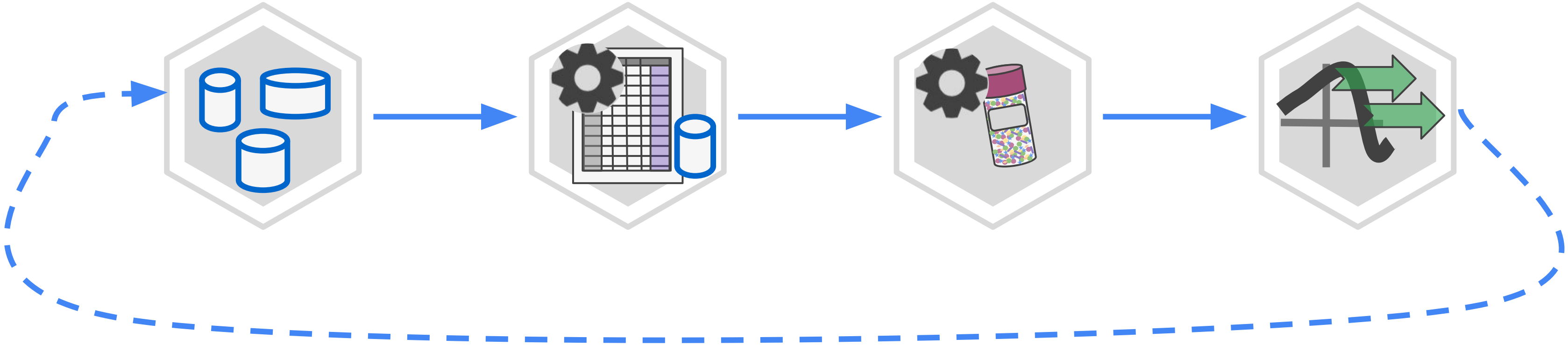
1. Make failure obvious
2. **Validate production changes (binaries + data)**
3. Clarify data integrity requirements
4. Handle pipeline backlogs

Where changes happen: binaries

Feature processing binaries

Training pipeline binaries

Serving binaries



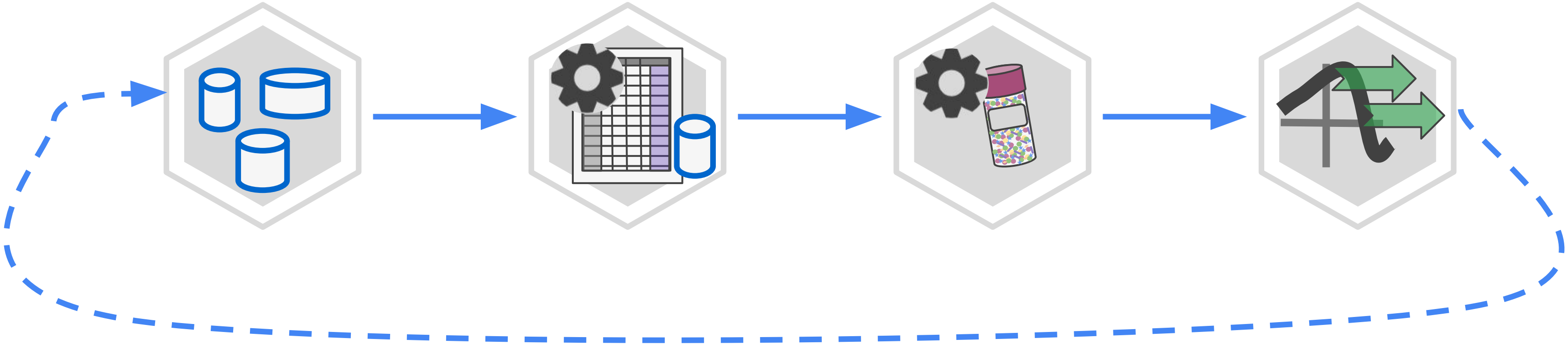
Where changes happen: configuration

Config data schema

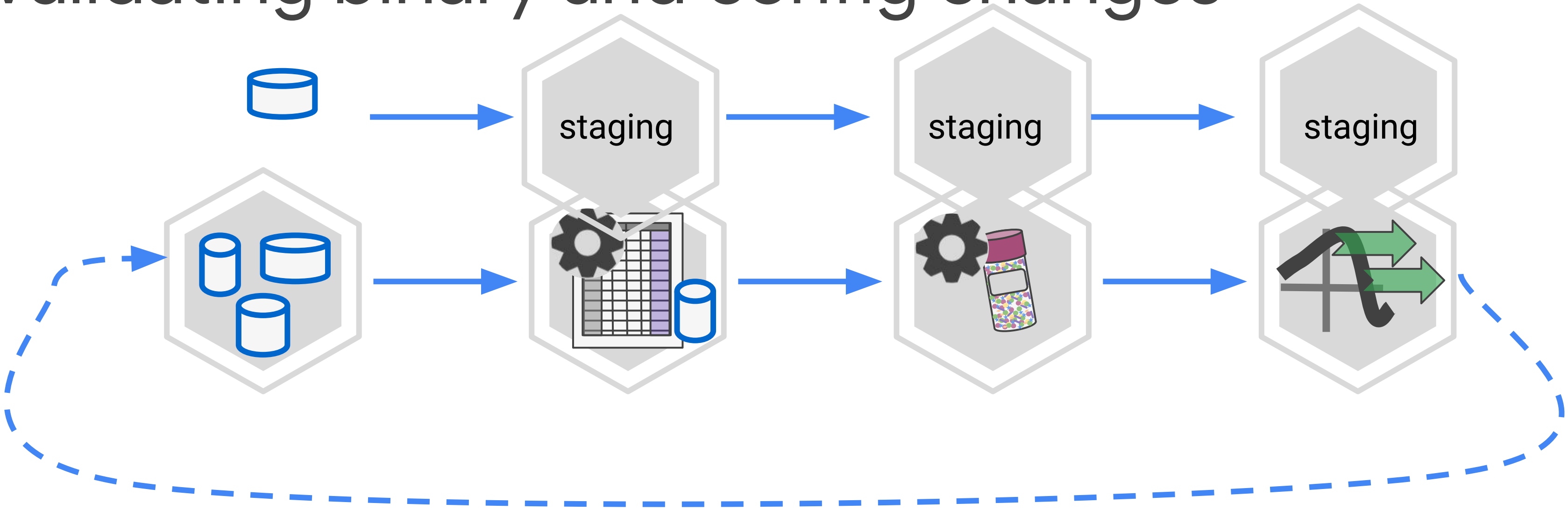
Feature processing binaries
Data schema/configurations

Training pipeline binaries
Training configurations

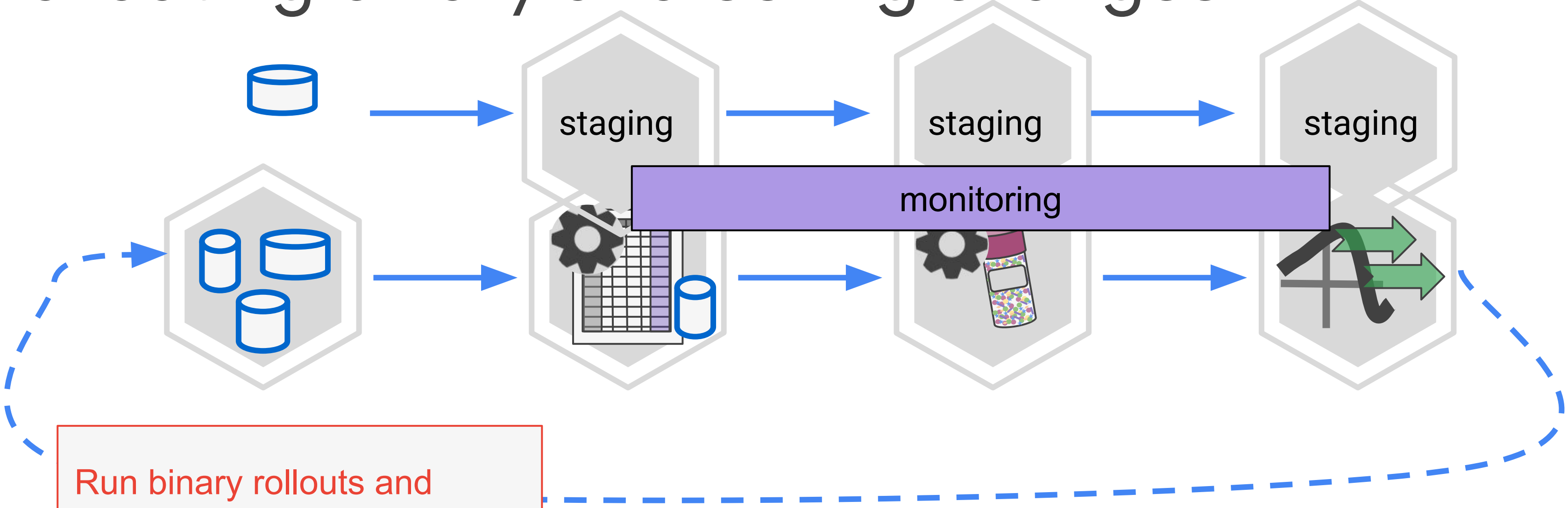
Serving binaries
Serving configurations



Validating binary and config changes



Validating binary and config changes



Run binary rollouts and config changes in staging environments. In particular, monitor for performance degradations.

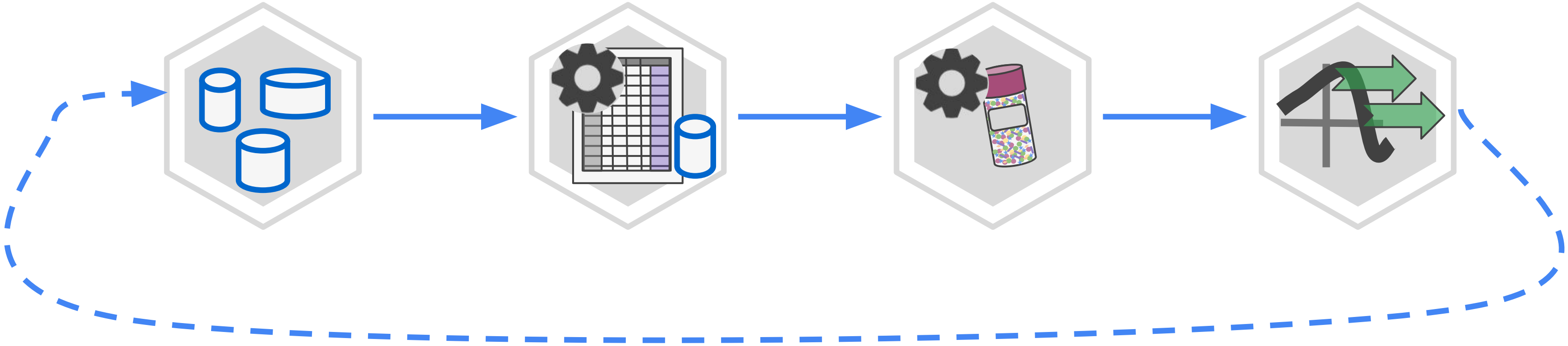
Where changes happen: data

Raw data schema
Raw data updates

Feature processing binaries
Data schema/configurations
Generated feature data updates

Training pipeline binaries
Training configurations
Model representations

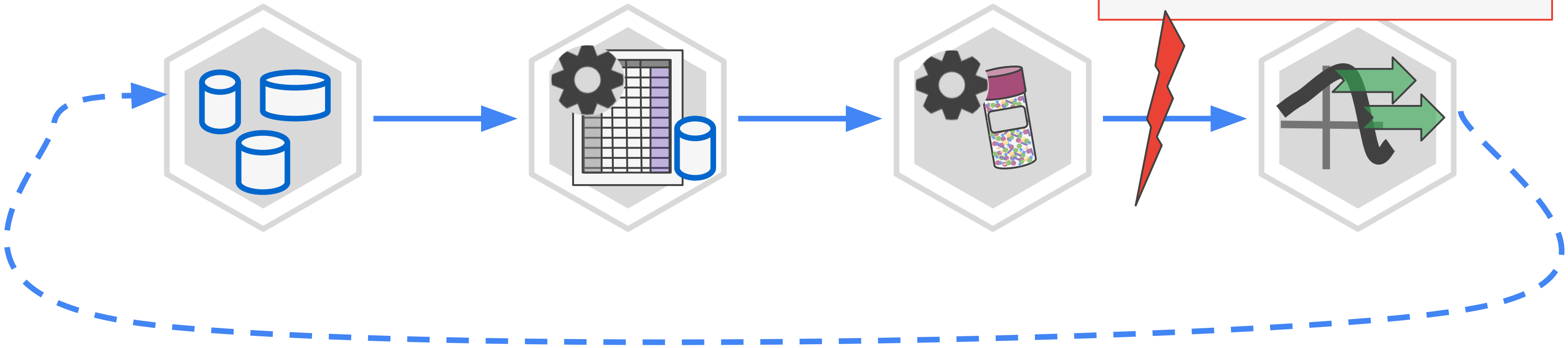
Serving binaries
Serving configurations
Inferences



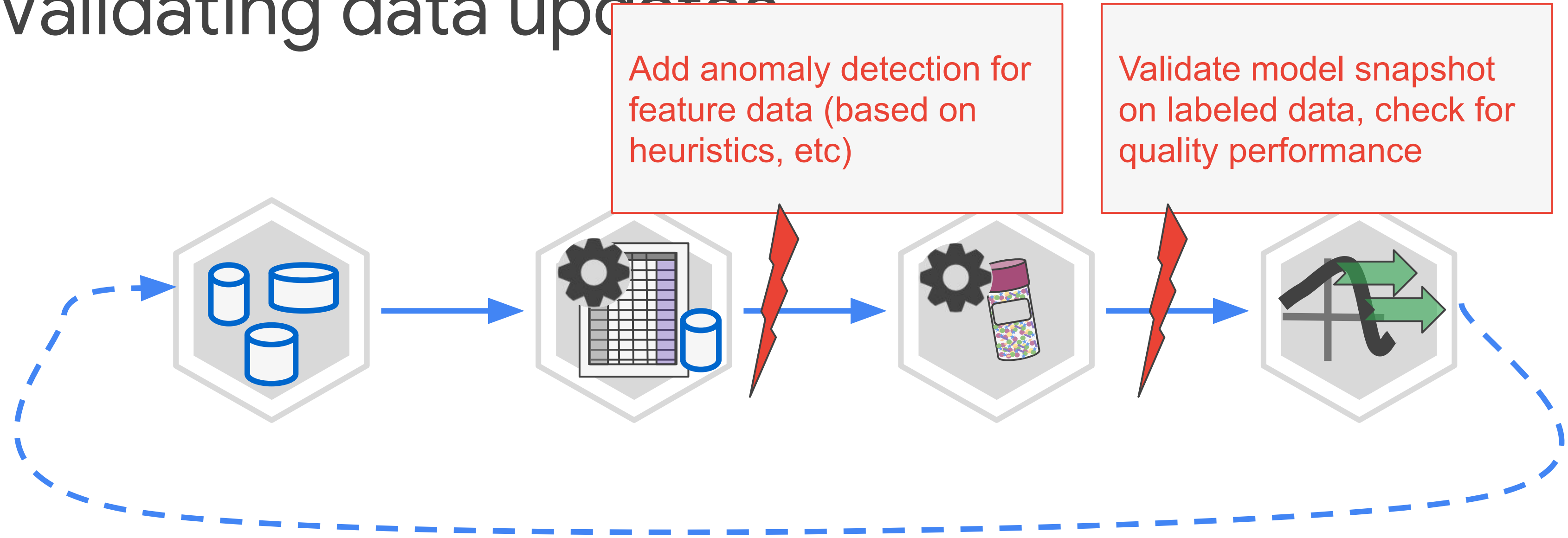
(Potentially other changes here)

Validating data updates

Validate model snapshot on labeled data, check for quality performance



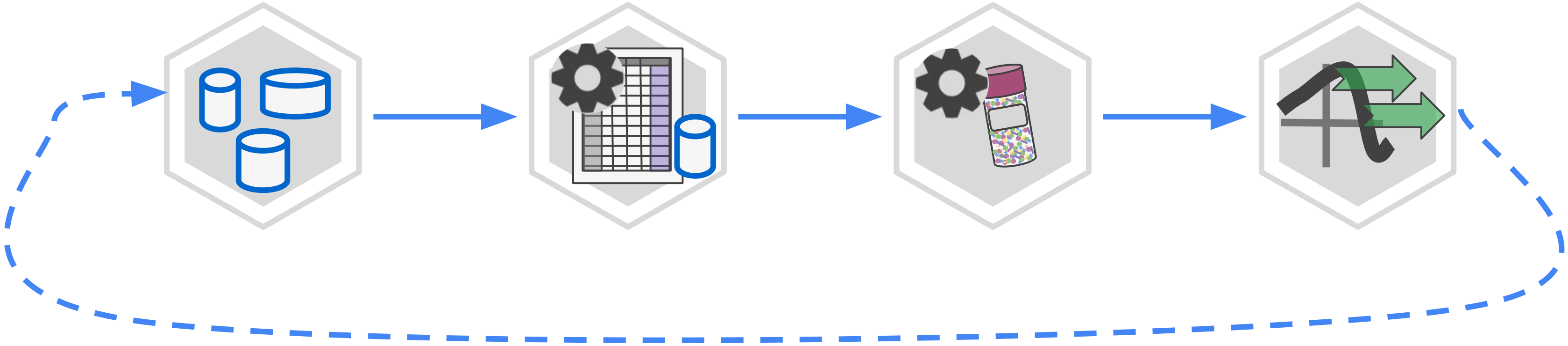
Validating data updates



4 things for more reliable ML

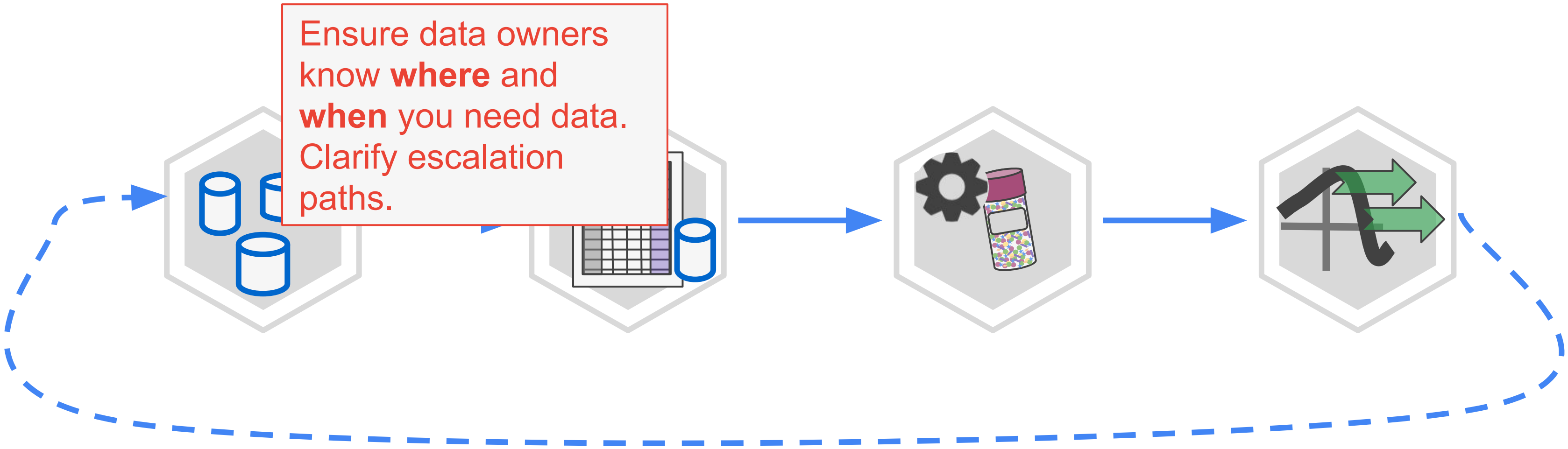
1. Make failure obvious
2. Validate production changes (binaries + data)
3. **Clarify data integrity requirements**
4. Handle pipeline backlogs

Where changes happen: other

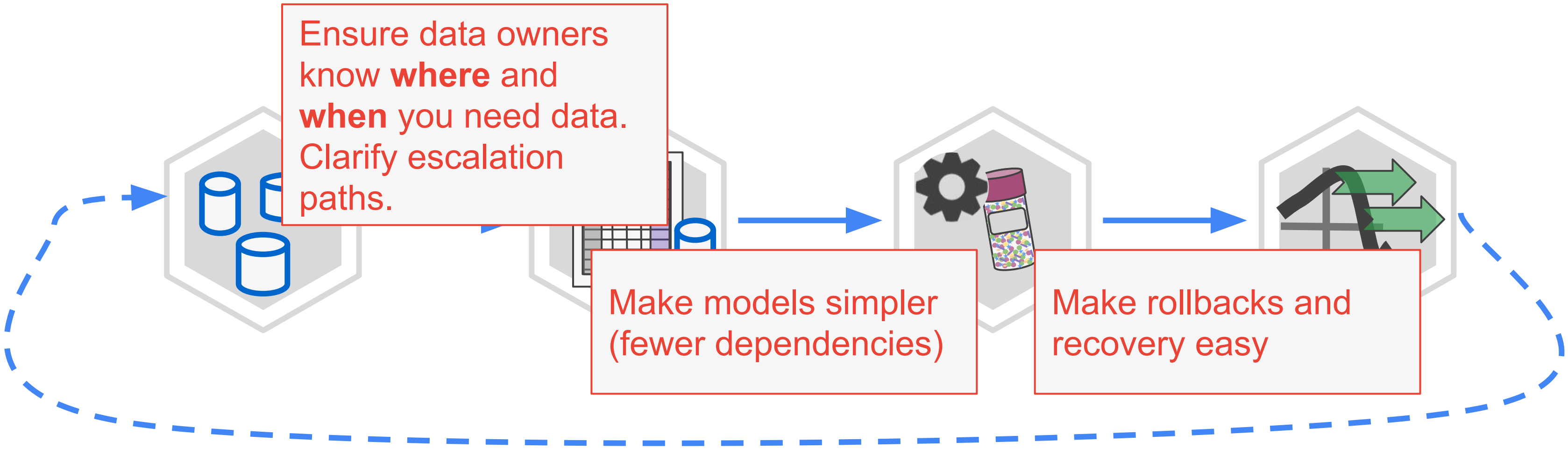


Here be dragons

Improving data integrity



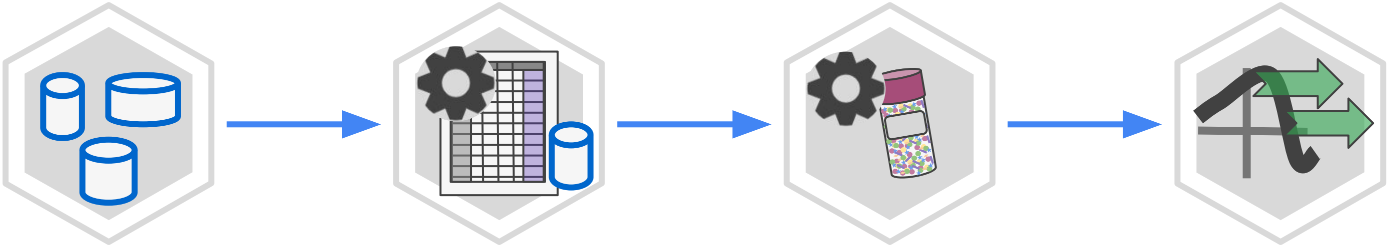
Improving data integrity



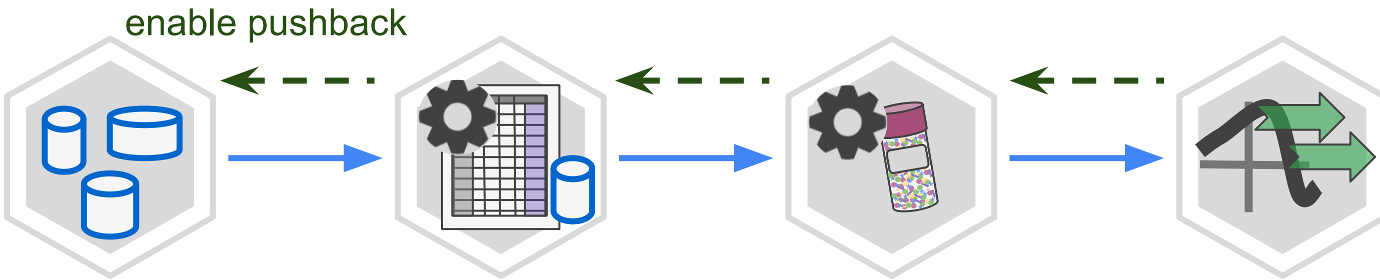
4 things for more reliable ML

1. Make failure obvious
2. Validate production changes (binaries + data)
3. Clarify data integrity requirements
4. **Handle pipeline backlogs**

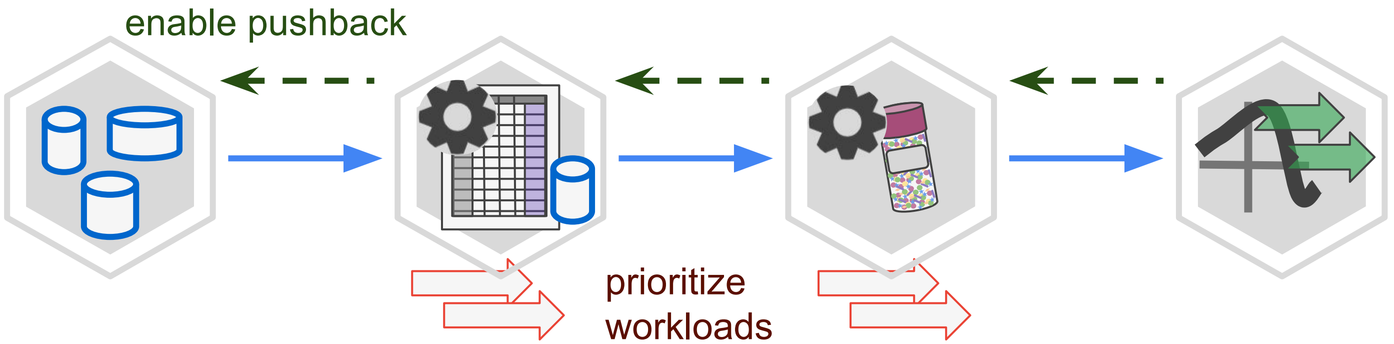
Handling pipeline backlogs



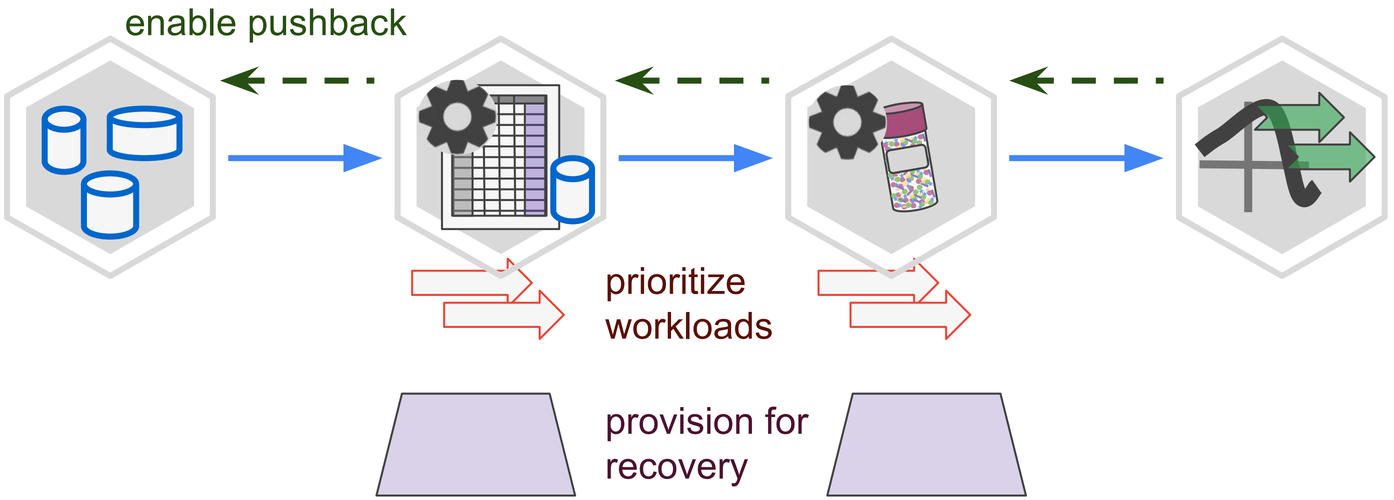
Handling pipeline backlogs



Handling pipeline backlogs



Handling pipeline backlogs



Conclusion

ML systems are weird, but no more than other systems we deal with

Successful risk management considers general best practices and knowledge about the specific system's characteristics

A few things to consider for ML:

1. Make failure obvious
2. Validate production changes (binaries + data)
3. Clarify data integrity requirements
4. Handle pipeline backlogs

Thank you!

Special thanks: Julian Grady, Gráinne Sheerin, Todd Underwood, Darinka Zečević, SRECon+OpML organizers