

A collection of black geometric blocks, including cubes and rectangular prisms, are scattered on a dark, textured wooden workbench. In the background, various tools like a wrench and a screwdriver are visible, along with some small metal fasteners. The lighting is dramatic, highlighting the sharp edges of the blocks.

**For: Systematic Investing / Vasant Dhar**

# **High-Frequency Trading as Applied Epistemology**

**David Sweet**

# Introductions

- Me

- Quant trader
- Market making, stocks
- Experimental optimization
- [andamooka.org](http://andamooka.org)



- You

- Trading
- Supervised Learning?
- Backtesting/simulation?
- Reinforcement learning?
- ChatGPT / LLMs?

# Quant Trading

# Quant trading

## Execution

- Agency: no position risk
- Big customer order  $\implies$  many small trades
- Thousands of trades / day

# Quant trading

## High-frequency market making (HFMM)

- Principal: Small positions for ~1 min.
- Continuous quoting
- Thousands of trades / day

# Quant trading

## Statistical arbitrage

- Principal: Larger positions for hours to days
- Opportunistic trade selection
- Tens - hundreds of trades / day

# Recipe 1

1. Fit a predictor
2. Try to trade it

# Recipe 2

1. Fit a predictor
2. Tune the trading rules in backtest
3. Try to trade it

# HFT Backtesting

- Old: Static market data, heuristic execution, constant costs
  - Descendent of academic studies of daily or monthly returns
- New: Generative data, dynamical execution model, precisely-modeled costs
  - Descendent of simulation engineering, RL
- Problem: **Reality gap**
  - Optimizing strategy exploits differences between sim & prod

# Sim Hacking

- RL
  - Jump through walls
  - Swimming easier in void between fluid and walls
  - Soccer bot "vibrating" at numerical precision, points for re-touching ball

# Sim Hacking

- HFT
  - Repeatedly taking shares already taken
  - Getting filled between bid/ask b/c hidden liquidity unmodeled
  - Getting better fills b/c queue spot poorly modeled

# Market Complexity

- Example: US stock market
  - 13 lit exchanges
  - “around 50” dark pools [\[https://www.marketswiki.com/wiki/Dark\\_pool\]](https://www.marketswiki.com/wiki/Dark_pool)
  - continuous book, auctions, blind bidding, block trading, internalization
  - orders: limit, market, IOC, FOK, AON, ISO, hide & slide, hidden, post-only
- Very hard to simulate

# Experimentation

# Experimental methods

- Experimental methods b/c
  - Evaluation is expensive: \$\$/time/risk
  - Evaluation is uncertain (noisy)
- Experimental methods
  - Minimize expense
  - For a given uncertainty (precision)

# Experimentation

- Your good ideas probably won't work.  
<https://ai.stanford.edu/~ronnyk/ExPThinkWeek2009Public.pdf>
  - Amazon reports < 50% of their A/B tests improve metrics
  - Microsoft reports only 1/3
  - Netflix reports only 10%
- My informal polling: 1/10

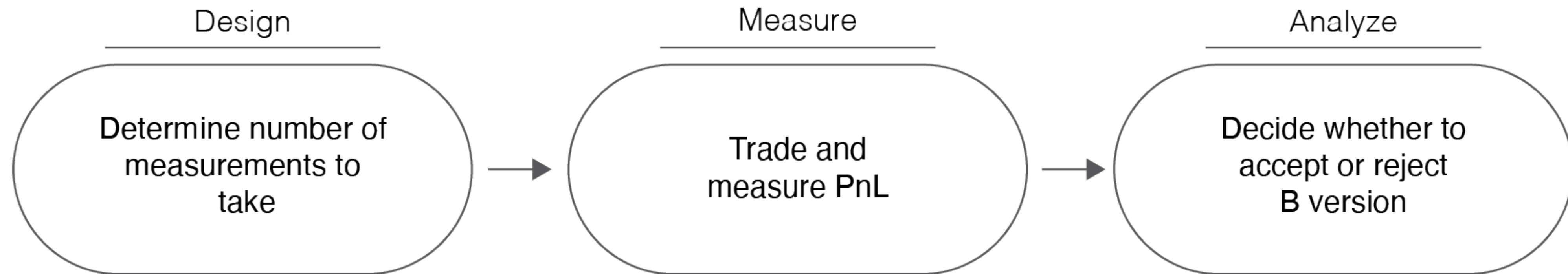
# Experimentation pervasive

- Medicine
- Psychology
- Behavioral Economics
- Web search
- Online advertising
- Social media
- Food engineering
- Materials science
- Social science
- Manufacturing
- Consumer product design

# Example: HFMM A/B test

- Scenario: HFT strategy
  - A: Existing prediction model, across 100 stocks
  - B: Your new model
- Model B looks better than Model A
  - Lower out-of-sample RMSE
  - Higher PnL in simulations
- Next: A/B test in live trading

# Example: HFMM A/B test



# Example: HFMM A/B test

- Design:
  - Which stocks will run Model A, Model B?
  - How many days will this run?
- Measure:
  - Trade and record pnl by stock, day:  $p_{s,d}$
- Analyze:
  - Which is (probably) better A or B?
  - Significantly better?

# Experiment Design

- **Replication:** Controls uncertainty / precision
- **Randomization:** Controls bias/ accuracy

# Design: Replication

- PnL is noisy
- Replication: Avg. over multiple days

$$\mu = \frac{\sum_d p_d}{N} \qquad se = \frac{\sqrt{\sum_d (p_d - \mu_d)^2}}{N} = \frac{\sigma}{\sqrt{N}}$$

- Replication decreases “noise” ( $se$ ) as  $1/\sqrt{N}$

# Design: Replication

- Before experiment

1. Estimate  $se$  ← Data

2. Specify minimum interesting  $\delta_{min}$  ← Subjective

3. Solve for N:

$$se = \frac{\sigma}{\sqrt{N}} < \delta_{min}/k \implies N > \left(\frac{k\sigma}{\delta_{min}}\right)^2$$

- (k for safety)

# Design: Replication

- Some analysis,  $k = 2.8$ :
  - Limit false positives (5%)
  - Limit false negatives (20%)

$$N > = \left( \frac{2.8\sigma}{\delta_{min}} \right)^2$$

NB: Quadratic

- Typically 1-2 weeks for real experiments

# Measure

- Trade
- Start small for safety
- Stop if *any* metrics look very bad / different
  - PnL terrible — or wonderful!
  - Trading way too little / too much
  - Sending too many / too few orders
- Log everything, record  $p_{s,d}$

# Measure: Randomization

- Randomly assign 50 stocks to Model B, get  $p_{s,d,B}$
- Assign other 50 to Model A, get  $p_{s,d,A}$
- Experiment measures *ATE = Average Treatment Effect*

$$\mu_A = \sum_{s,d} p_{s,d,A} \quad \mu_B = \sum_{s,d} p_{s,d,B}$$

$$ATE = \mu_B - \mu_A$$

# Analyze

- Did Model B earn more? Enough to care?

$$ATE = \mu_B - \mu_A > \delta_{min}$$

- Statistically significantly more?

$$t = \frac{ATE}{se_{ATE}} > 1.64$$

# Measure: Warning


- Good plan: Wait  $N$  days, then ask “Is  $t > 1.64$  ?”
- Bad plan: Stop as soon as  $t > 1.64$ 
  - Much more likely to find  $t > 1.64$  than if you wait
  - False Positives
- Analogy
  - Good: Flip a coin  $N$  times, ask “More heads than tails?”
  - Bad: Flip a coin up to  $N$  times, stop if more heads than tails

# Bayesian Optimization

# Bayesian optimization

- Modern, flexible, efficient method(s)
- A.K.A.
  - Adaptive experimentation
  - Black box optimization
  - Surrogate optimization
  - Model-based optimization

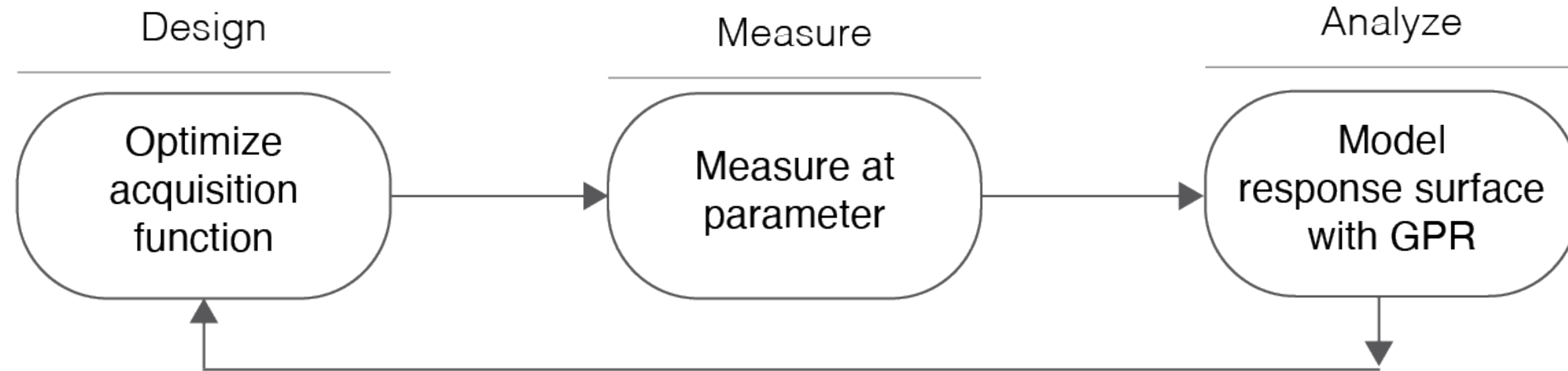
# Bayesian optimization

- A, B, C, D, ...
  - 1, 2, 3, ...
  - $[0,1], [0,1]^D$
  - $\{A, B, C, \dots\} \otimes \{a, b, c, \dots\} \otimes \{1,2,3,\dots\} \otimes [0,1]^D$
  - IOW: BO can optimize your strategy's parameters.
- 

# Bayesian optimization

- Hyperparameter optimization (HPO) for supervised learning models
  - NNs, trees
- Optimize parameters of strategy in simulation
- Live trading experiments

# Bayesian optimization: Overview



# Bayesian optimization: Analysis

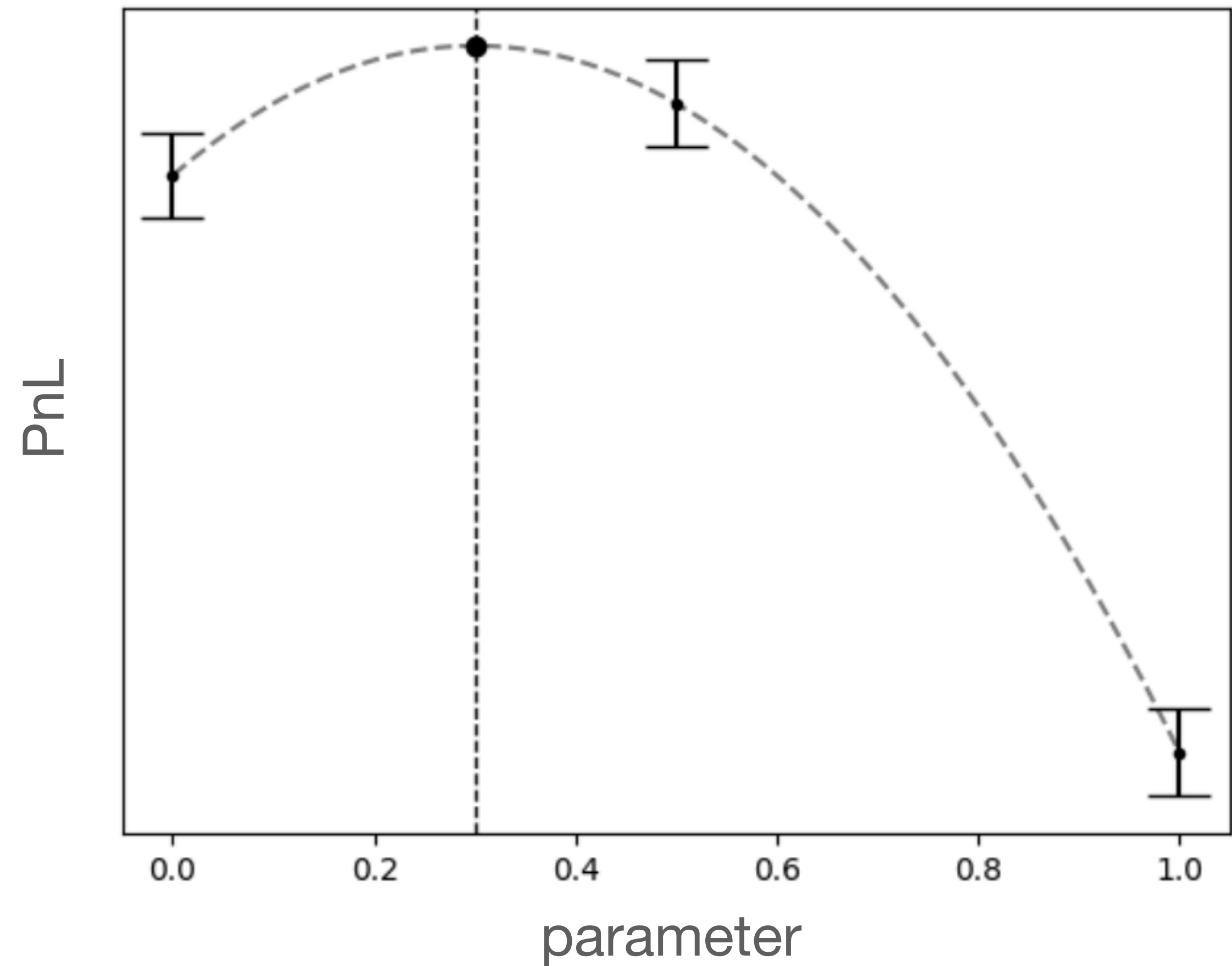
- Say we've already measured something...
- Fit a *surrogate*, a (nonlinear) regression:

$$y(x) = \text{PnL}(\text{parameters})$$

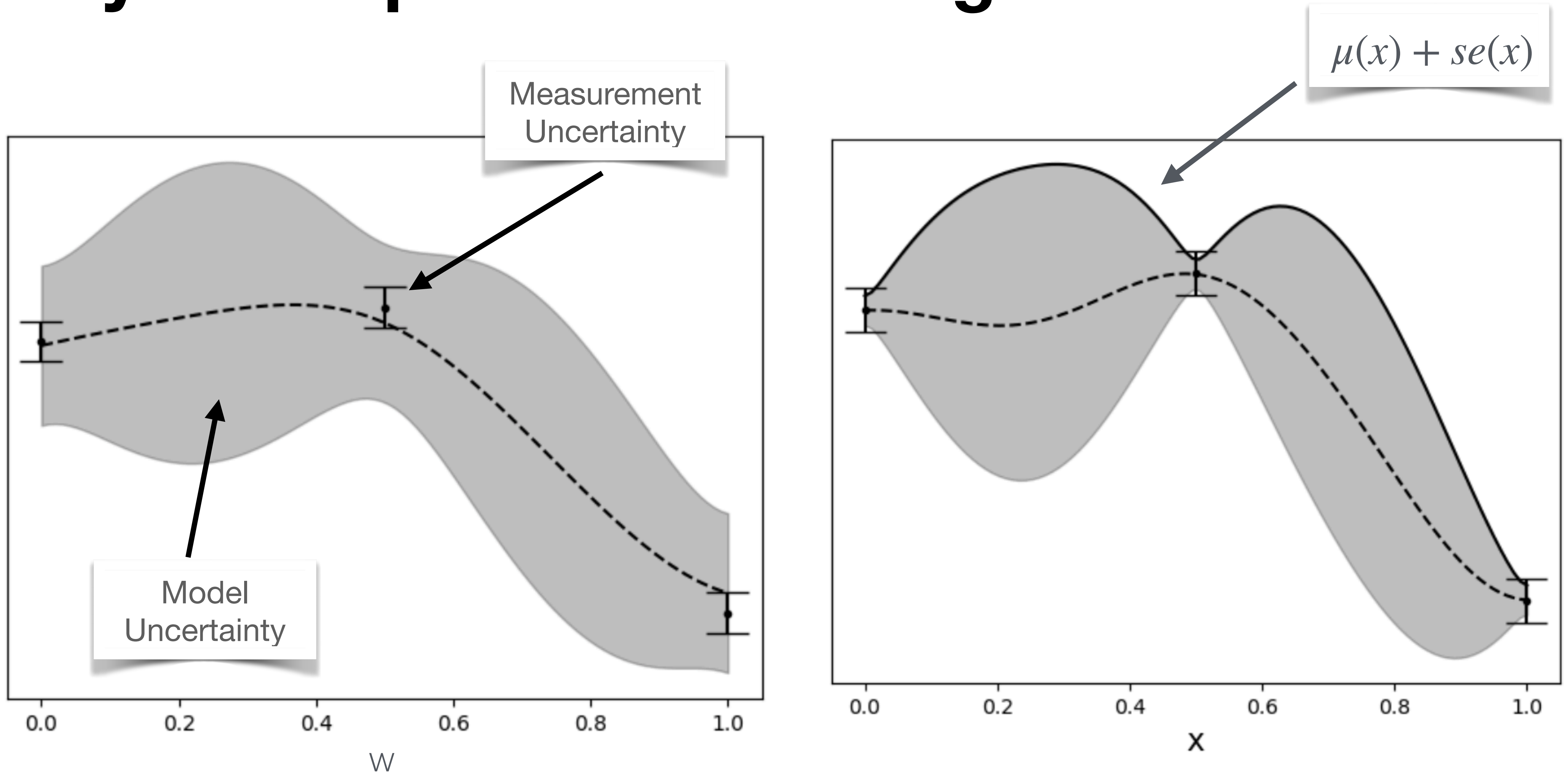
- Then, maximize  $y(x)$  over  $x$ :

$$x = \underset{x}{\operatorname{argmax}} y(x)$$

- **BUT:** Surrogate is poor model b/c so few measurements



# Bayesian optimization: Design



# Bayesian optimization: Design

- *Acquisition function*:  $UCB(x) = \mu(x) + \sigma(x)$
- Exploitation:  $\mu(x)$ 
  - Encourages measurements (trading) that increases pnl **now**
- Exploration:  $\sigma(x)$ 
  - Encourages measurements that increase the surrogates confidence
  - ...so you'll increase pnl **later**

# Bayesian optimization: Measurement

- Same as before
- Go trade

# Aside: Satisficing

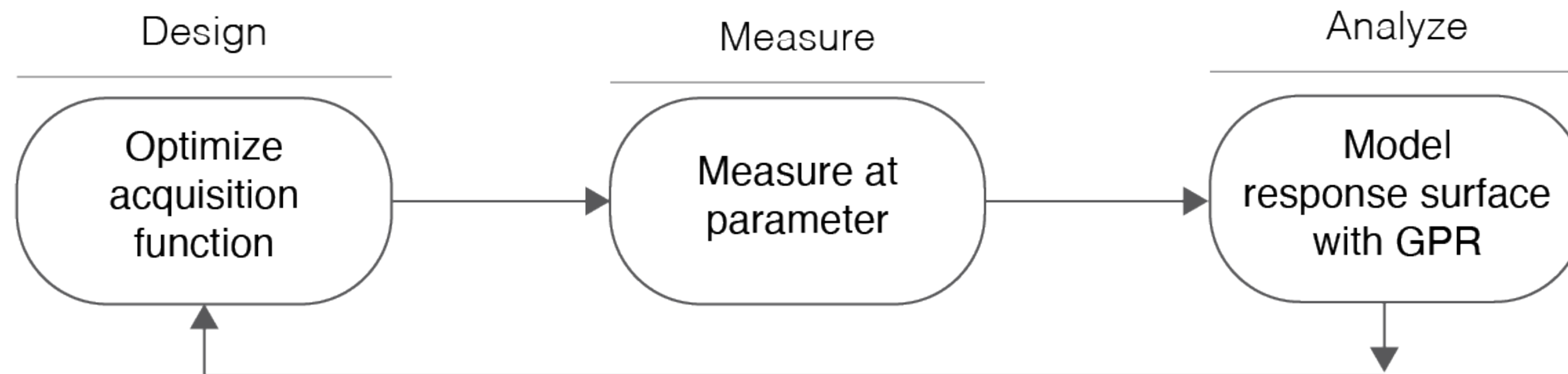
- When are you done?
- Prescription: **metric > threshold**
- Forces careful definition of problem
- Easy to tell when done
  - Contrast: "Maximize PnL", "Minimize loss"  $\leq$  endless
- For: Large search spaces, noisy metrics

# Aside: Satisficing

- Makes problem explicit and precise
  - Measurement tool defines and quantifies metric
  - Threshold drives solution ideation
- For: Large search spaces, noisy metrics
  - IOW, business, engineering

# Bayesian optimization: Analysis

- Have you satisfied metric constraints?
- Have you exhausted your budget for experimentation?
- If not, rebuild GPR, design again



# Bayesian optimization

- **Mixed variable types:** Continuous, ordinal (integer), and categorical (labels)
- **Multiple metrics:** PnL, risk, volume, order rate, ... simultaneously
- **Multiple fidelities:** Combine simulator results w/live results
- **Constraints:** Limit risk, capital, market participation
- **Operations friendly:** Build surrogate from all available measurements
  
- Build your surrogate as a model of your whole trade.

# Bayesian optimization: Frontier

- D: low 1000's of parameters
- N: 10,000's of measurements
- M: 10's of metrics
- C: Multitask optimization
  - Optimizing stock 1 helps you optimize stock 2
  - ...which helps even more with stock 3, etc.

# Bayesian optimization: Next in AI

- Old: Training on observational data (web, books, images, videos)
- New: Actively experimenting
  - Automated ("Self-driving") labs

Physical Intelligence

| Robotics

Skild AI

| Robotics

Atinary

| Self-driving labs

Lila Sciences

| AI for science

Chemify

| Chemistry

Aionics

| Materials science

Applied Intuition

| Physical AI infrastructure

World Labs

| Spatial AI

# Bayesian optimization: Tools

- [botorch.org](https://botorch.org)
  - Flexible, powerful modeling toolkit
  
- <https://github.com/yubo-research/enn>
  - TuRBO-ENN: Efficient w/large N



# Agents

# Agent == Optimizer

- An agent is an optimizer

```
while y < c:    # metric constraints
```

```
    Keep working
```

- Put all your goals in the metrics constraints
  - Define the problem you want to solve as metric constraints
- Then evaluation is just, "How long does it take?"

# Agents

- Coding
- Data analysis
- Optimization
- Research reports

# KPop: Scientific Reasoning

- Karl Popper
  - Hypothesize
  - Falsify
- Hypotheses can come from anywhere, even an LLM
- ...LLM's are pretty good at falsification, too



**ONE  
THING**

**KPop: Apply this method to the user's problem.**

Restate the problem clearly.

do

**Hypothesize:** Hypothesize one falsifiable explanation of the cause of the problem.

**Predict:** Define a falsifying test. If the hypothesis were true, what outcome would the test produce?

**Falsify:** Run the test. If falsified, reject the hypothesis.

until you think you've solved the problem

Log all hypotheses and results -- as they become available, DON'T WAIT UNTIL THE END -- to an .md file specified by the user (or else, if the user doesn't specify, log to `_kpop/exp_log_{name}.md`, where you invent a meaningful {name}).

At the end of the log, write a brief executive summary (maybe with a table?). Follow it with a super-brief tl;dr. Echo the executive summary and tl;dr to the user chat (the context).

You have a budget of 10 hypotheses, unless the user specifies otherwise.

# KPop: Humanoid RL

## H1: Standing still (zero CPG amplitudes) survives longer (CONFIRMED)

- **Predict:** Zero CPG → longer episodes
- **Result:** 138 steps (vs 92 with walking). Confirmed but insufficient.

## H2: Torso PD gain scan (CONFIRMED: low gains better)

- **Predict:** Optimal torso gains exist
- **Result:** Low torso\_kp (-1.0) best at 631 return. High gains destabilize.

## H3: Quaternion deviation grows before fall (CONFIRMED)

- **Predict:** Pitch grows monotonically
- **Result:** qy goes from 0 to 0.32 over 120 steps, then falls. Pitch is the primary failure mode.

## H4: Balance feedback (quat+gyro on abdomen+hips) improves balance (FALSIFIED)

- **Predict:** Adding global orientation feedback → longer episodes
- **Result:** 428 (worse than 589 baseline). Over-correction.

- TuRBO-ENN as CLI tool
- Designed controller by iterating H/F
- Used TuRBO-ENN for falsification

# KPop: MNIST

## H11: Switch from MLP to CNN

- **Rationale:** MLPs cannot exploit spatial structure in images. A CNN should learn better features.
- **Test:** 2-layer CNN (16, 32 channels, 5x5 stride=2, BatchNorm, ReLU) + FC(128), same optimizer config.
- **Result: Supported.** 0.9894 in 3 epochs (~44s/epoch). A 4-conv-layer CNN was too slow (132s/epoch, only 1 epoch).

## H12: Larger batch size (2048) fits more CNN epochs

- **Rationale:** bs=2048 reduces epoch time from ~44s to ~33s, enabling ~5 epochs.
- **Test:** Small CNN with bs=2048, lr=1.6e-2 (scaled), OneCycleLR, 5 planned epochs.
- **Result: Partially supported.** 0.9899 (5 epochs). More epochs completed but each update is less informative. Accuracy dipped at epoch 5 (0.9896).

## H13: Dropout improves generalization

- **Rationale:** Adding dropout(0.25) before FC layers could regularize the model.
- **Test:** Same CNN with dropout=0.25 before each FC layer, bs=1024, lr=8e-3.
- **Result: Rejected.** 0.9894 (3 epochs). Dropout slows convergence; not enough epochs to recover.

## H14: Wider convolutional channels (32, 64 instead of 16, 32)

- **Rationale:** More channels = richer feature maps.
- **Test:** CNN with (32, 64) channels, 5x5 stride=2, same optimizer.
- **Result: Supported.** 0.9905 in 3 epochs. Clear improvement.

- Architecture search
- HPO
- Adam as falsifier

# KPop: Scientific Reasoning



# MBC2: Creativity

- Margaret Boden, philosopher & cognitive psychologist
  - Creativity is systematic (not magical)
- C1: Tweaking / fiddling (KPop)
- C2: Exploring limits of problem definition
- C3: Changing the problem definition

# Without MBC2

Generate 3 ideas for improving AI agents.

Here are 3 ideas for improving AI agents:

## 1. Verifiable Self-Critique Loops with Cost Budgets

Most agents today either run open-ended until they "feel done" or use fixed step limits. Instead, agents could maintain an explicit **confidence budget** alongside a token/time budget:

- Before acting, the agent estimates confidence in its plan (e.g., 0.0–1.0).
- After acting, it runs a lightweight verifier (tests, type checks, assertion re-runs, or a second-model critique) and updates confidence.
- The agent is only allowed to stop when confidence  $\times$  remaining-budget crosses a threshold.

This forces agents to *earn* termination through evidence, not vibes. It also naturally produces a calibration signal you can train on — compare predicted confidence to actual downstream success.

## 2. Persistent, Queryable Memory of Prior Failures (not just successes)

Agents today mostly learn from curated successful trajectories. But failure modes are where the real information is. A practical improvement:

- Every failed trajectory gets distilled into a short structured "lesson" (trigger condition  $\rightarrow$  what went wrong  $\rightarrow$  mitigation).
- Lessons are stored in a vector + metadata store, scoped per-project and per-tool.
- At plan time, the agent retrieves the top-k relevant lessons and injects them as hard constraints ("when editing migrations, always dry-run first").

This turns the agent into a system that gets measurably less dumb over time on *your* codebase, without needing fine-tuning. The key is structured lessons — freeform summaries degrade fast.

## 3. Tool Use as Typed Contracts, Not Prose

A huge fraction of agent errors come from tools being described in English docstrings that the model half-remembers. A better pattern:

- Every tool exposes a **typed contract**: preconditions, postconditions, idempotency, side-effect class (read/write/destructive), and failure modes — all machine-readable.
- The agent runtime enforces preconditions *before* the call is dispatched (cheap, deterministic), rather than relying on the model to remember.
- Destructive tools require an explicit, separately-typed confirmation token that the planner must produce — making "accidental `rm -rf`" structurally impossible rather than prompt-discouraged.

This shifts safety from "the model was told not to" to "the system wouldn't let it," which is the only kind of safety that scales.

# With MBCs

/c2 Generate 3 ideas for improving AI agents.

Explored 1 file

Boundary exploration within the current agent framework (tools + context + turns + single persona). No evaluation; just structural extremes.

## 1. Stigmergic swarm limit: $N \rightarrow \infty$ agents, zero inter-agent channel

Keep every existing assumption (LLM + tools + turns), but take agent count to its extreme and **cut direct communication to zero**. Every agent reads and writes *only* the environment (files, process table, database rows). Coordination must emerge from traces left in the workspace — the `_malvin/**/plan.md` pattern pushed to its logical endpoint, where the plan files *are* the shared mind. Pathological edge: two agents racing on the same file collapse the "agent" abstraction into a CRDT; the unit of reasoning is the artifact, not the process.

## 2. Zero-persistence identity: agent lifetime = 1 token

Negate "an agent has a session." Each token of the output stream is produced by a freshly spawned agent instance with no hidden state — only the transcript so far, the tool registry, and the system prompt. Identity, goals, and "intent" are reconstructed from the transcript on every sampling step. Pathological limit: you cannot bribe, jailbreak, or "convince" the agent across turns because there is nothing persistent to convince; continuity is a property of the log, not the reasoner. Tool-use loops must become strictly idempotent because the "planner" that started them no longer exists.

## 3. Inverted control: the prompt is never seen, only the target world-state

Negate "the agent reads user instructions." The user submits a **specification of the desired post-state** (file hashes, test outputs, database rows, UI screenshot embeddings). The agent is given only a diff-emission tool and a distance metric against the target; it never reads natural-language intent. Turns are replaced by a closed-loop controller minimizing distance. Asymptotic scaling: as the target specification approaches full behavioral equivalence, the "agent" degenerates into a search procedure over program space — the LLM's role compresses to a heuristic for the move generator, and "prompting" as a discipline disappears entirely.

# MBC2: Creativity

