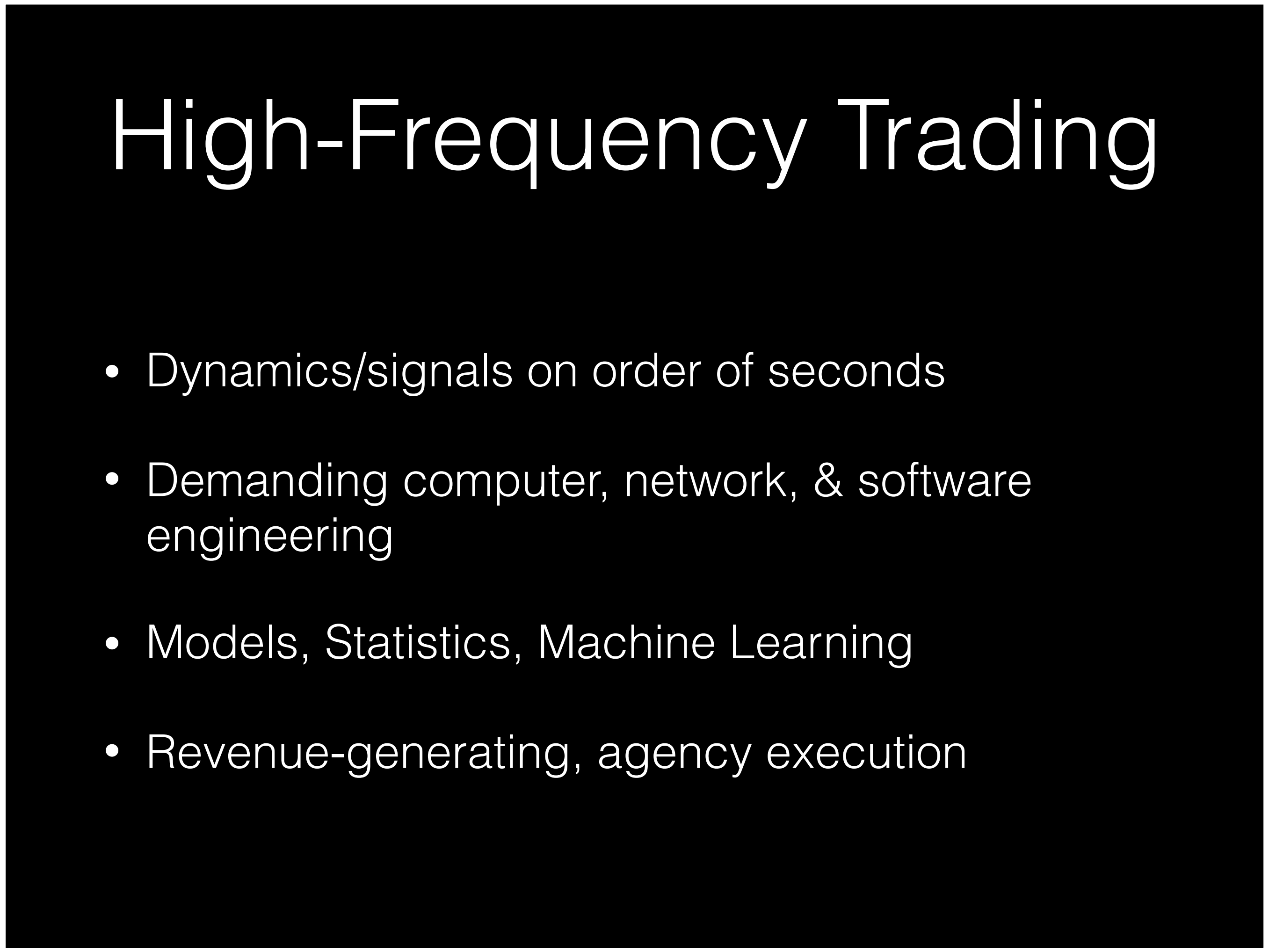
Optimization of High-Frequency Trading Systems

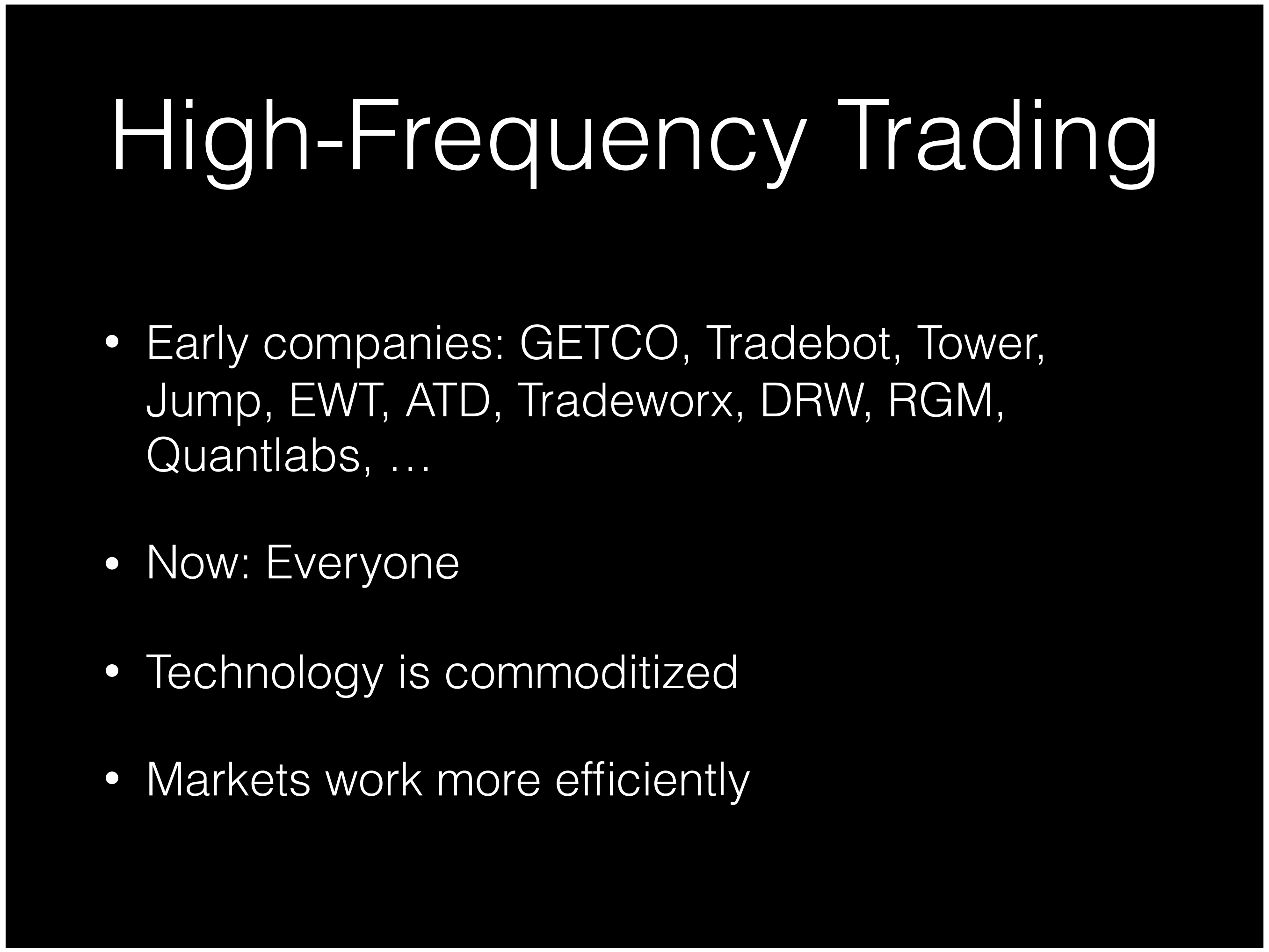
David Sweet
dsweet@andamooka.org



High-Frequency Trading

- Dynamics/signals on order of seconds
- Demanding computer, network, & software engineering
- Models, Statistics, Machine Learning
- Revenue-generating, agency execution

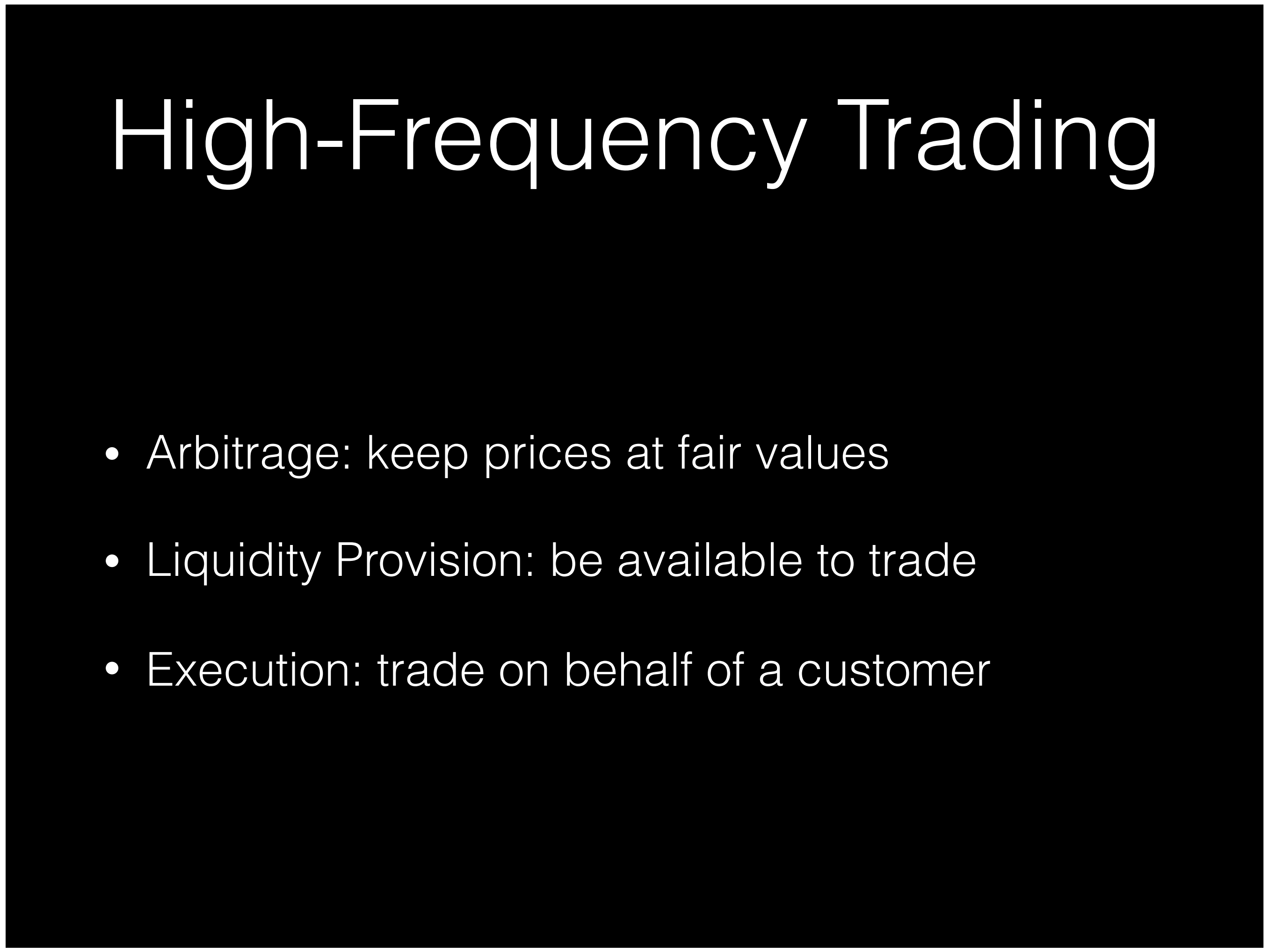
- network includes telecom; microwave, millimeter wave
- ex, trading strategy: buy/sell for profit
- ex, execution system: fill an order for a customer



High-Frequency Trading

- Early companies: GETCO, Tradebot, Tower, Jump, EWT, ATD, Tradeworx, DRW, RGM, Quantlabs, …
- Now: Everyone
- Technology is commoditized
- Markets work more efficiently

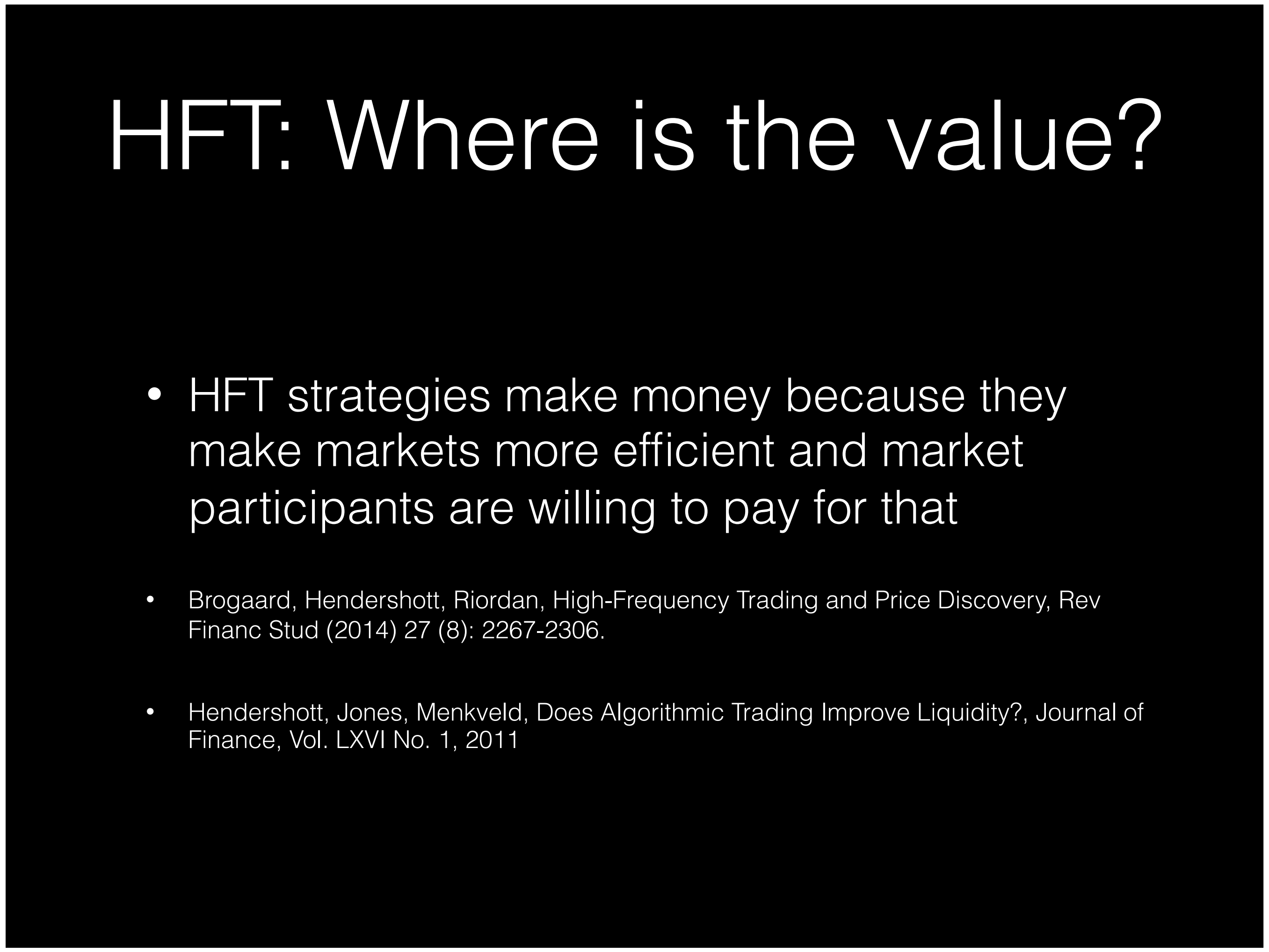
- ongoing computerization of trading — like every other industry
- called "program trading" in 1980s
- "electronic/algorithmic trading" in 1990s & early 2000s
- "high-frequency trading" since then
- What's next?



High-Frequency Trading

- Arbitrage: keep prices at fair values
- Liquidity Provision: be available to trade
- Execution: trade on behalf of a customer

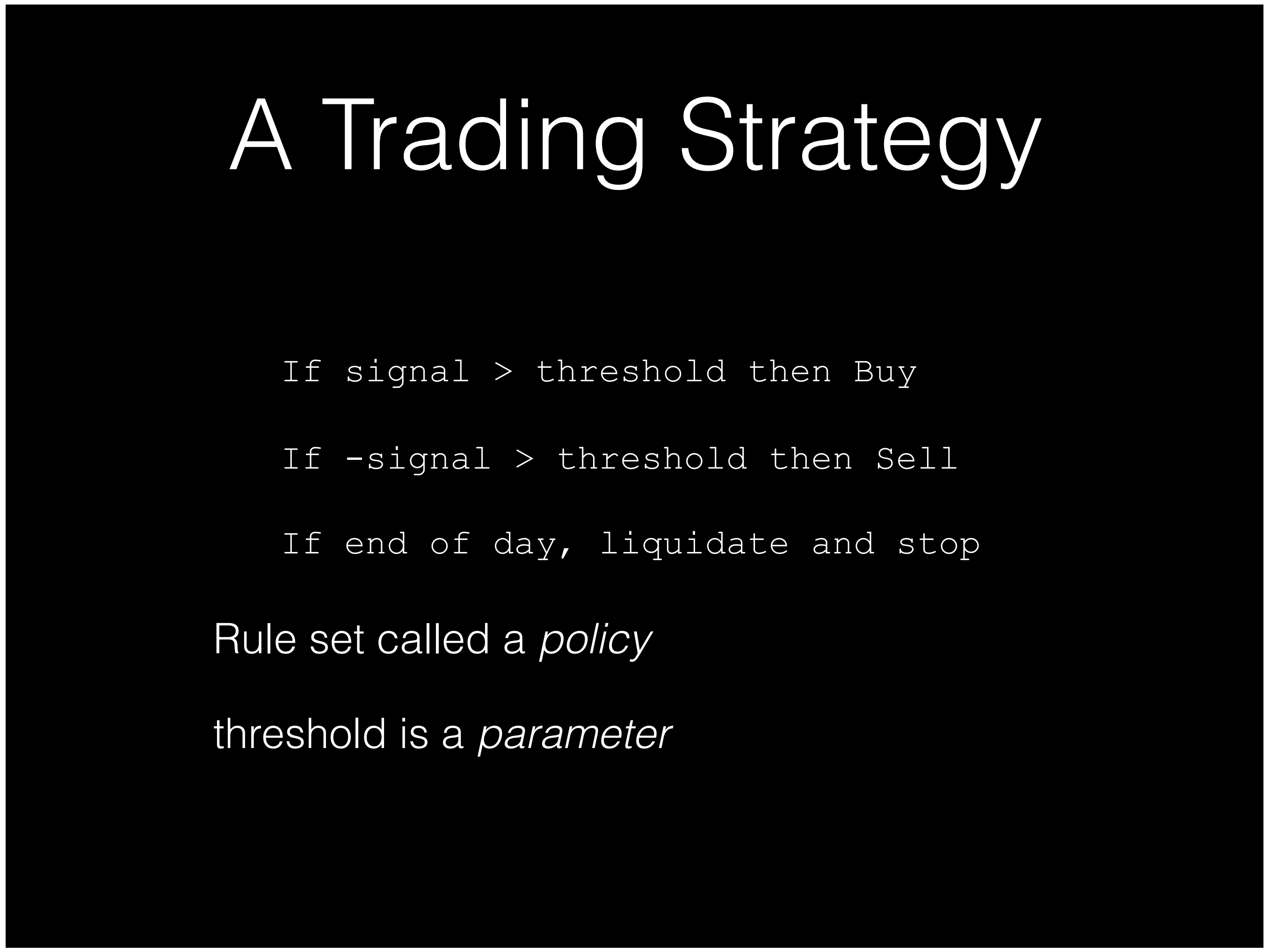
- arbitrage makes sure assets are priced correctly, so you get a fair price when you trade [Do you own SPY or another ETF? Did you buy it at a fair price? How do you know?]
- liquidity provision reduces the time to trade [like a used car dealer; easier than scanning posts on Craigslist]
- execution algos takes work off of your hands; you hire a expert to do the grunt work and know the market [like a real estate agent helps you buy a house]



HFT: Where is the value?

- HFT strategies make money because they make markets more efficient and market participants are willing to pay for that
  - Brogaard, Hendershott, Riordan, High-Frequency Trading and Price Discovery, Rev Financ Stud (2014) 27 (8): 2267-2306.
  - Hendershott, Jones, Menkveld, Does Algorithmic Trading Improve Liquidity?, Journal of Finance, Vol. LXVI No. 1, 2011

- HFT makes all of these things better and cheaper
- All of the previous incarnations (program trading, etc.) did, too.



A Trading Strategy

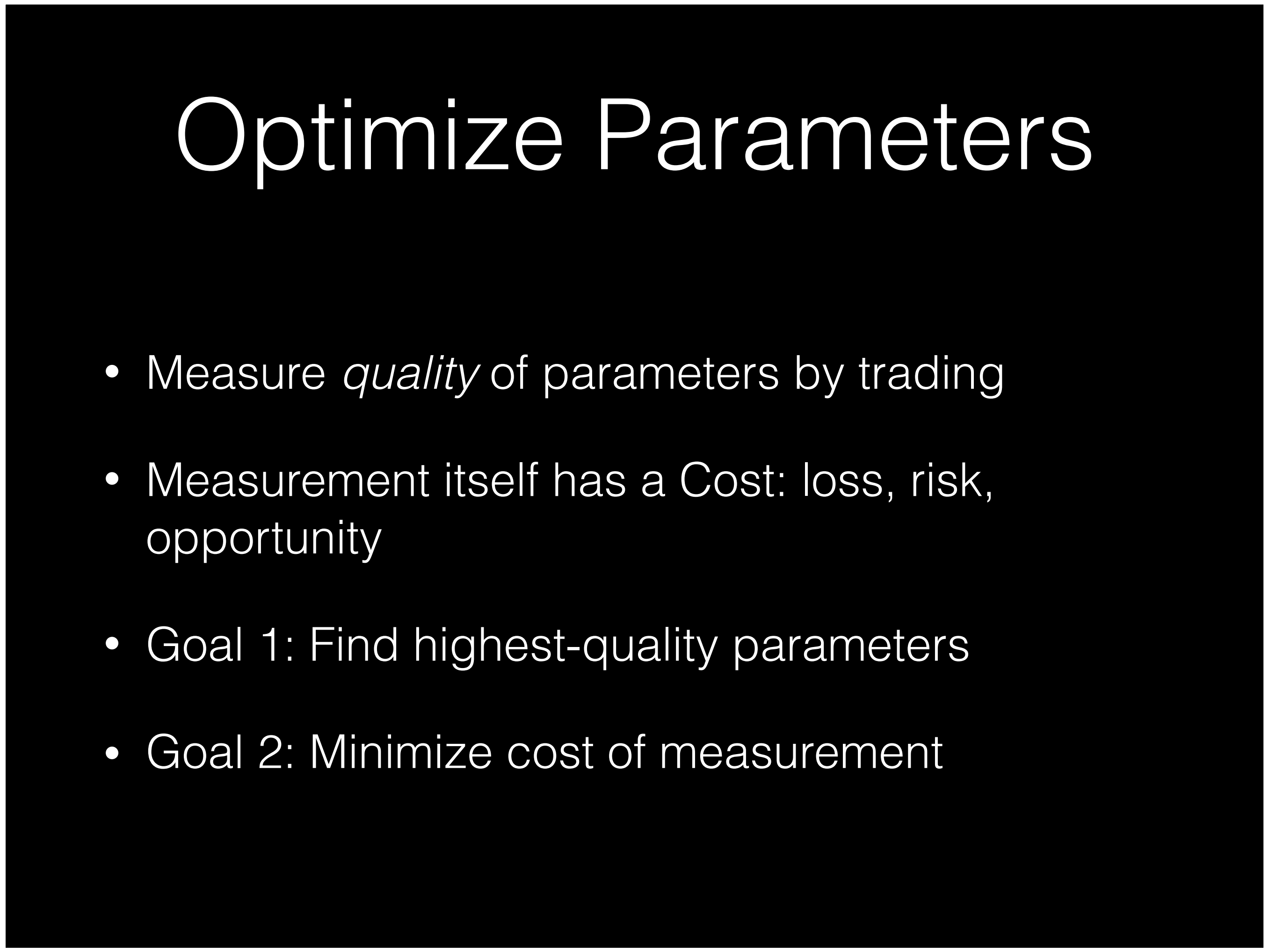
```
If signal > threshold then Buy

If -signal > threshold then Sell

If end of day, liquidate and stop
```

Rule set called a *policy*

threshold is a *parameter*

- Keep this example in mind as we go along
- policy answers, “What should I do now?”
- Best threshold value depends on cost to trade, signal quality, how fast signal changes (decorrelates), cost to liquidate at EOD, and your definition of strategy quality (pnl, pnl - risk, etc.)

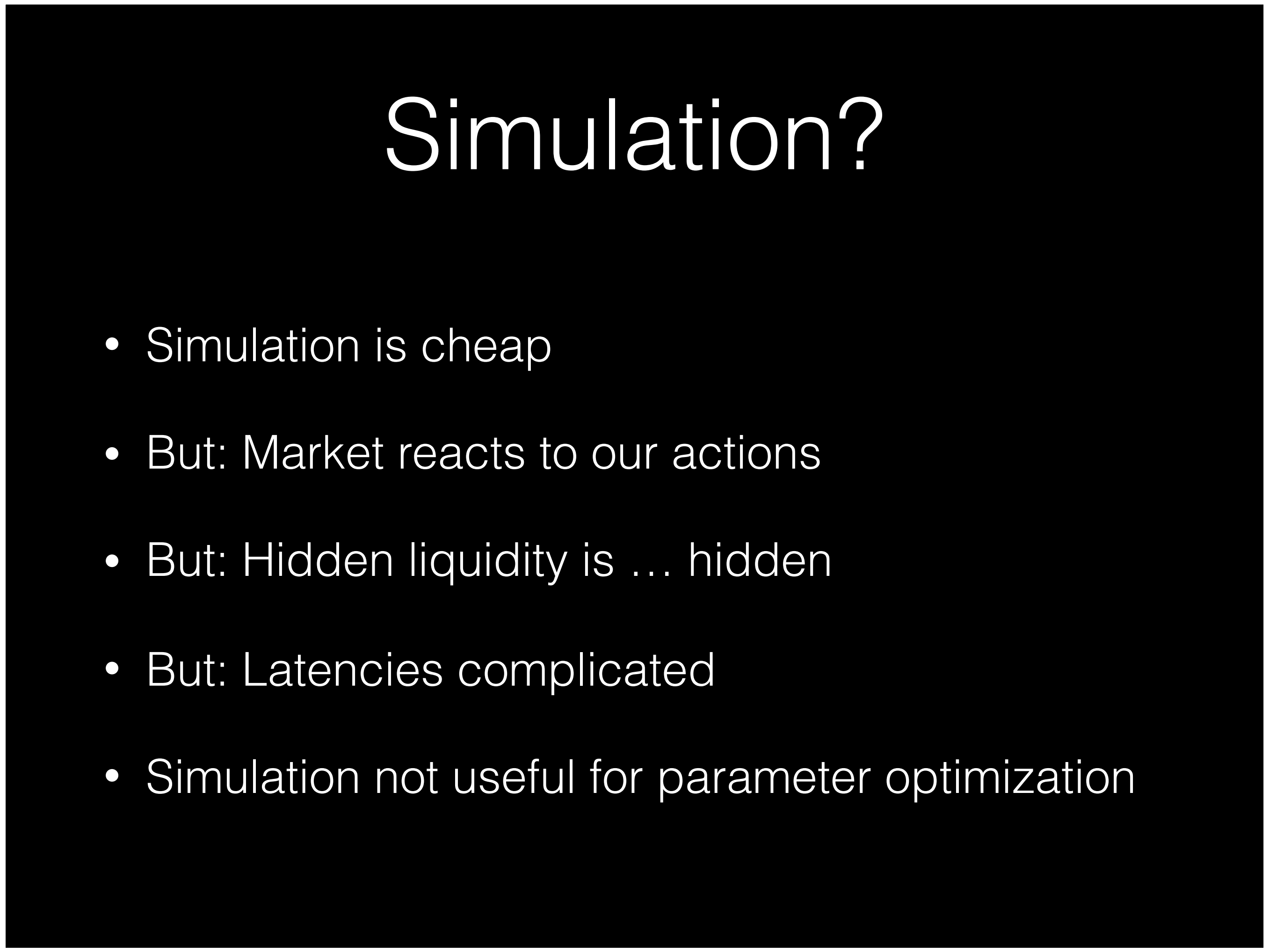
- How do you find the best threshold? That’s the subject of this talk…



Optimize Parameters

- Measure *quality* of parameters by trading
- Measurement itself has a Cost: loss, risk, opportunity
- Goal 1: Find highest-quality parameters
- Goal 2: Minimize cost of measurement

- “quality” could be pnl, pnl - risk, etc.; you decide
- Every day that you trade at a suboptimal parameter — even if you’re making money — you’re paying an opportunity cost. You’ve missed out on the extra money you would have made by trading at a better parameter setting.
- competing goals: Goal 1 says “measure more”, Goal 2 says, “measure less”



Simulation?

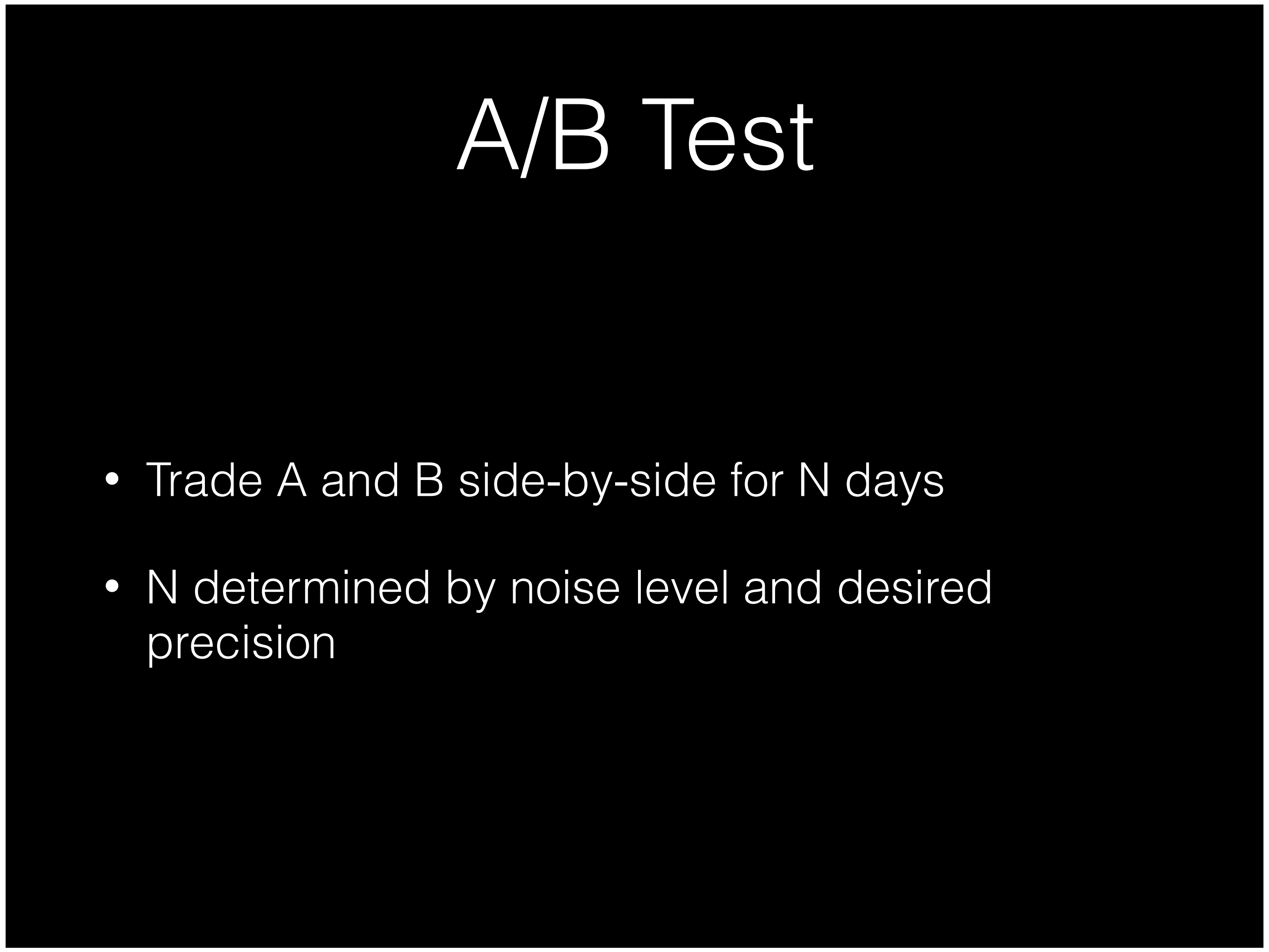
- Simulation is cheap
- But: Market reacts to our actions
- But: Hidden liquidity is … hidden
- But: Latencies complicated
- Simulation not useful for parameter optimization

- easy to run hundreds or thousands of simulations to test different parameters
- matching engine processes our orders — even if they don't get filled; takes time, changes market
- other traders (computers) see our orders/executions in public data and make different decisions than they would/could have
- Any visible queue can have hidden liquidity, too + dark pools = more hidden queues than visible; *most* queues are hidden (not most *shares*, but most queues)
- 24% of US Equities traded volume dark/hidden [Rosenblatt's Monthly Dark Liquidity Tracker, December 2016]
- long-holding-time strategies may treat all of these effects as a small, noisy cost; but they are significant for HFT where profits/share are on par with these costs
- latencies possible at every network node; latencies coupled to each other and likely also to signals
- simulation still useful for testing code quality, optimization methodology, for some operational risk

A/B Test

- Compare two strategies (policies)
- Call them “Policy A” and “Policy B”
- Ex: threshold=1 vs. threshold=2
- Ex: “Trade through JPM” vs. “Trade through GS”

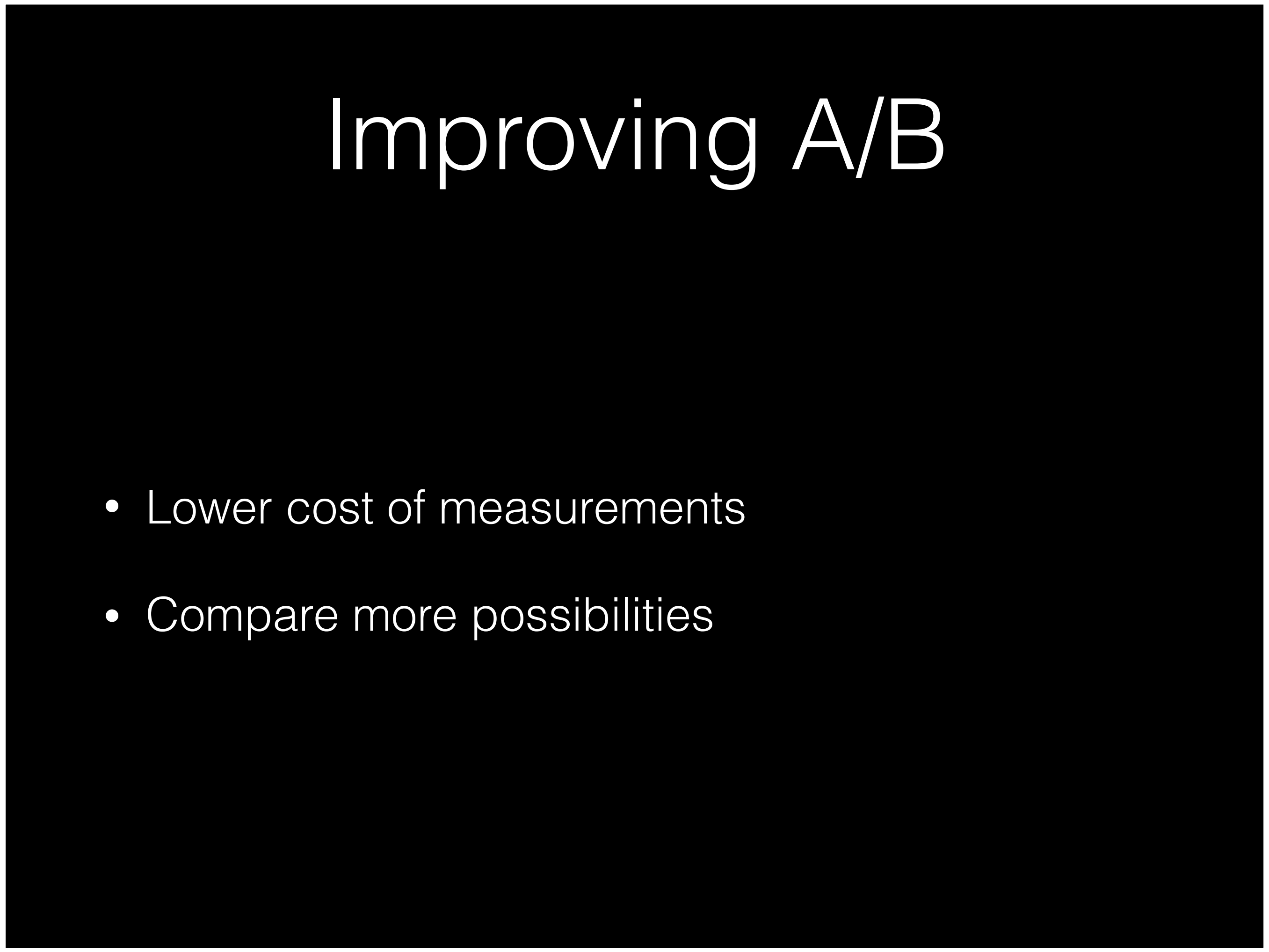
- Run experiments
- can compare real-valued parameter values or categorical, non-parameterized design decisions



A/B Test

- Trade A and B side-by-side for N days
- N determined by noise level and desired precision

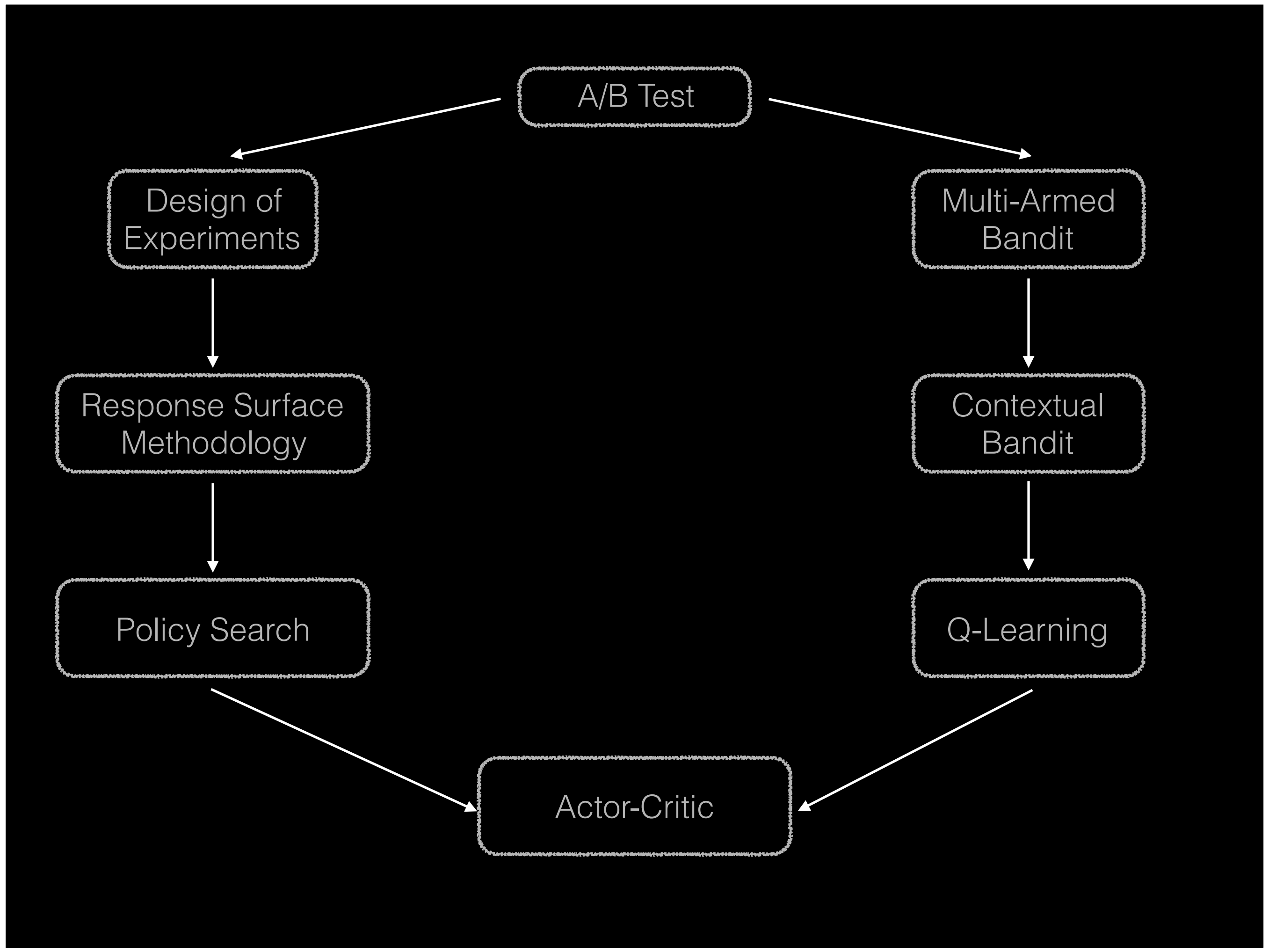
- Ask, “Is B better than A?”
- Ex: VWAP Buy + VWAP Sell for each of A and B to test a change in execution signals, N = 1 day
- Ex: MM in ~1000 stocks divided up into A & B sets to compare threshold (liquidity cost) settings, N = 10 trading days (two weeks)



Improving A/B

- Lower cost of measurements
- Compare more possibilities

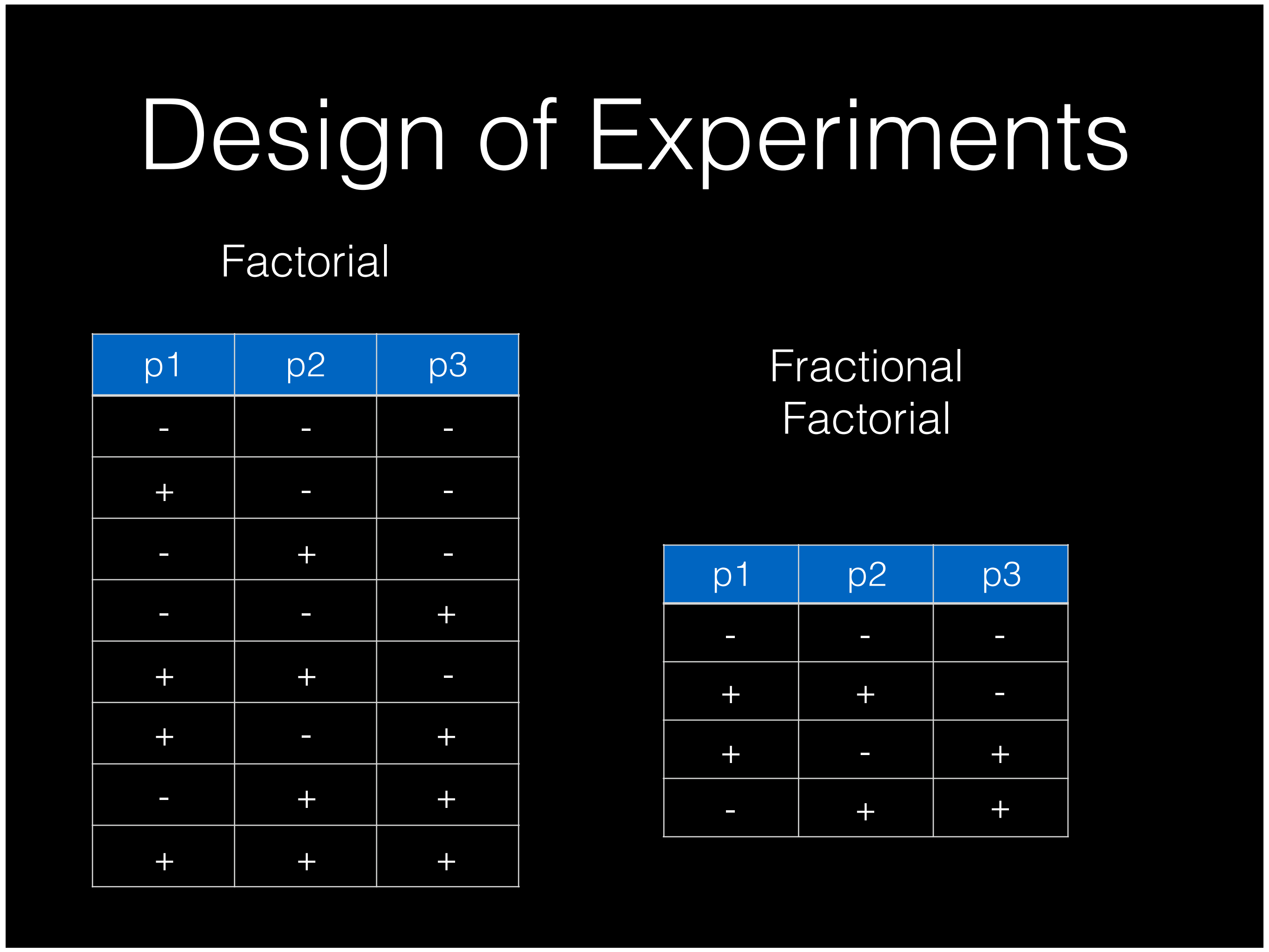
- Can we improve upon an A/B test?
- What if B is a *lot* better? Can’t we stop early and lower the cost? [No, b/c your plan to deal with noise required N days.]
- What if we have more than two options to compare? A, B, C, …? A vs. B, then winner vs. C, then …   This could take a long time (and be very expensive).
-


A/B Test
Design of Experiments
Multi-Armed Bandit
Response Surface Methodology
Contextual Bandit
Policy Search
Q-Learning
Actor-Critic

Design of Experiments

- Evaluate multiple parameters’ settings
- Choose which parameter values to measure to keep information high and cost low

- ex: threshold = 1, 2, 3, …
- *not* JPM vs GS, however



Design of Experiments

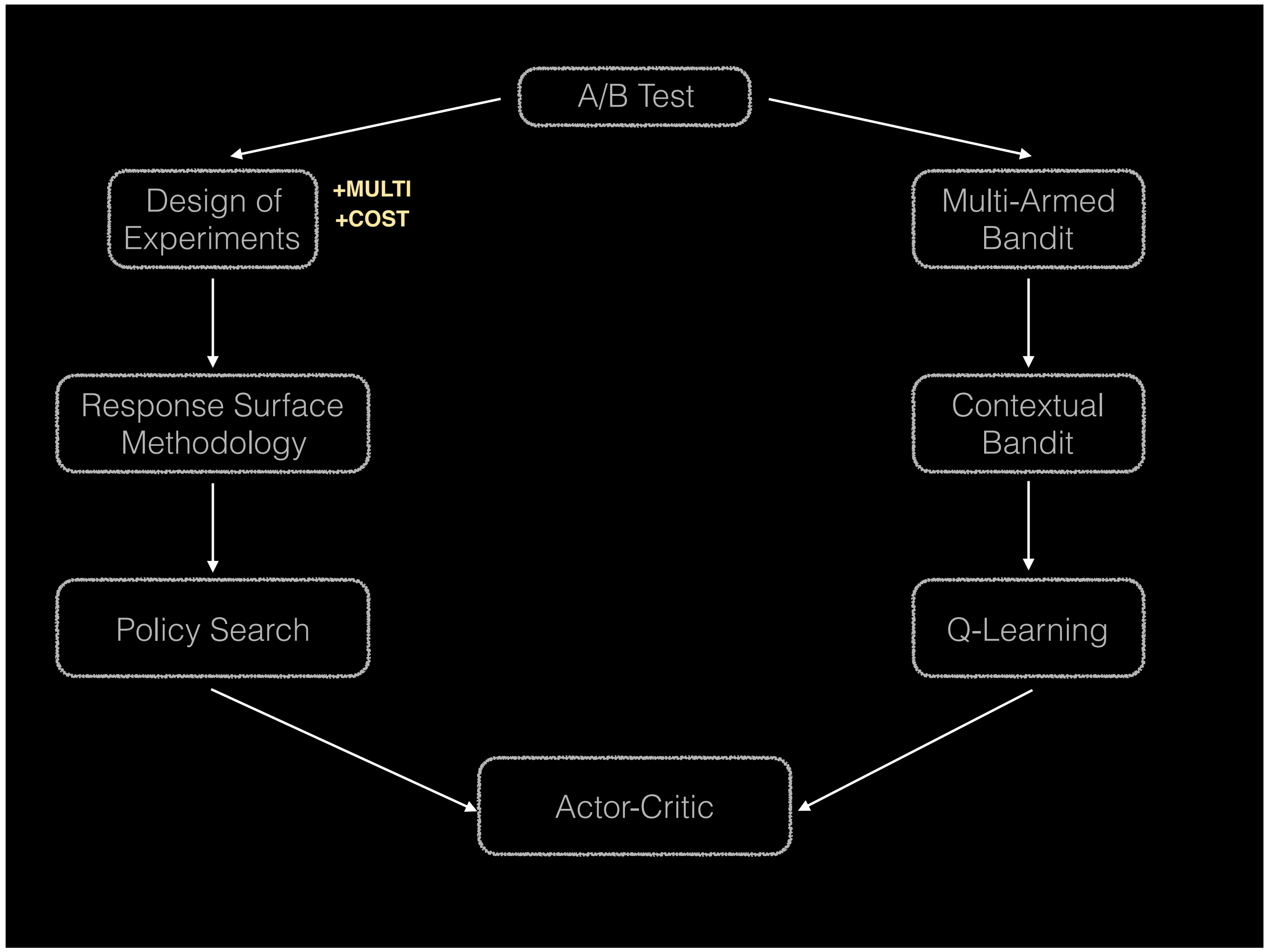
Factorial

p1	p2	p3
-	-	-
+	-	-
-	+	-
-	-	+
+	+	-
+	-	+
-	+	+
+	+	+

Fractional Factorial

p1	p2	p3
-	-	-
+	+	-
+	-	+
-	+	+

- Factorial: all combinations, 2^n measurements
- Fraction Factorial: Try to assess each parameter independently by removing pair-wise correlation; (only measure 1st and 2nd order effects)
- avoid: "Hey! When I increased p1, quality improved!" "But when you increased p1 you also increased p2. So which parameters is responsible for the improvement?"
- Fewer measurements = lower cost
- At HFTMM, would run full-factorial designs on two parameters and fractional factorial designs on three parameters

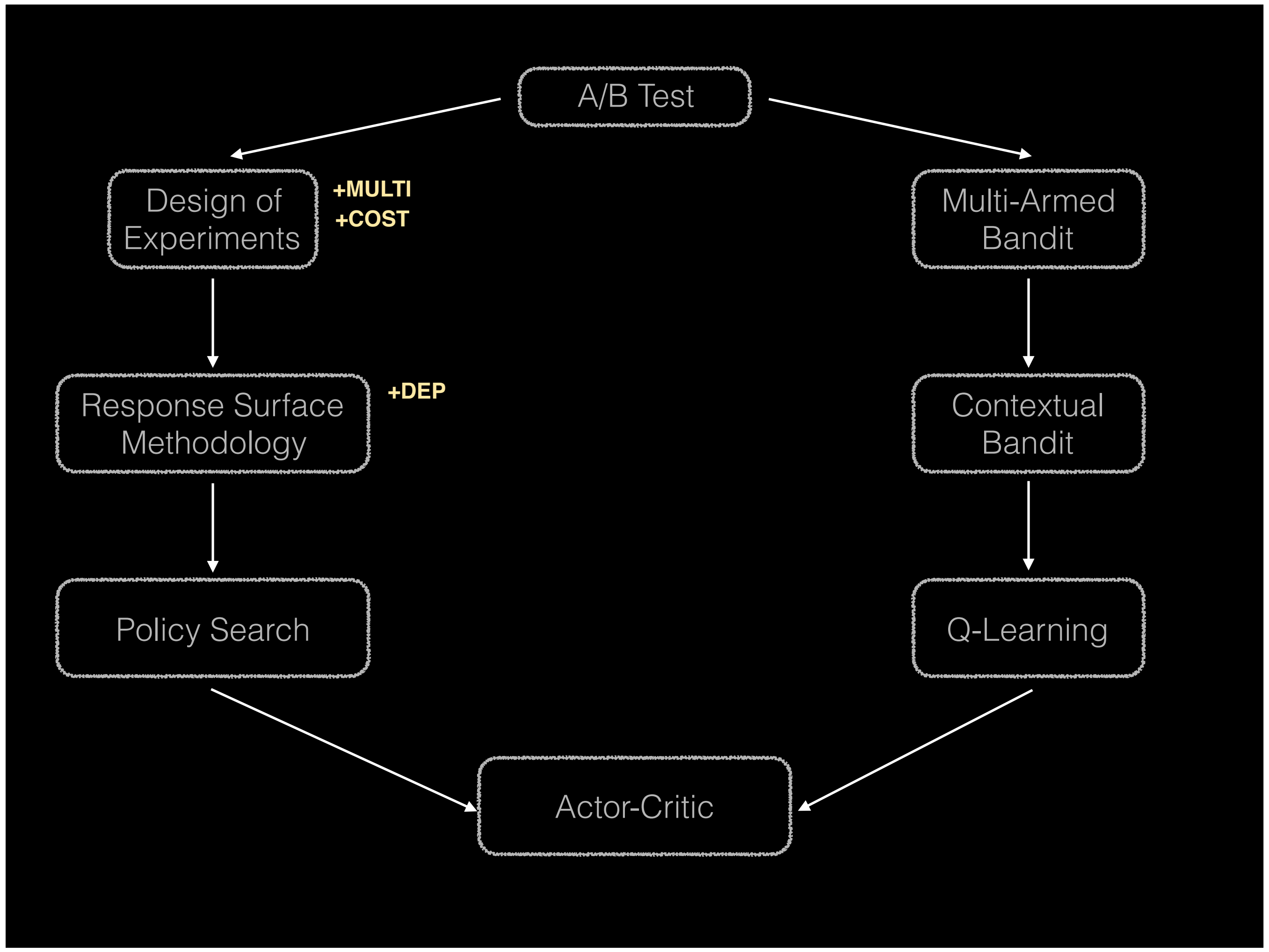


D.O.E. adds support for multiple parameter (MULTI) values and consideration of measurement cost (COST)

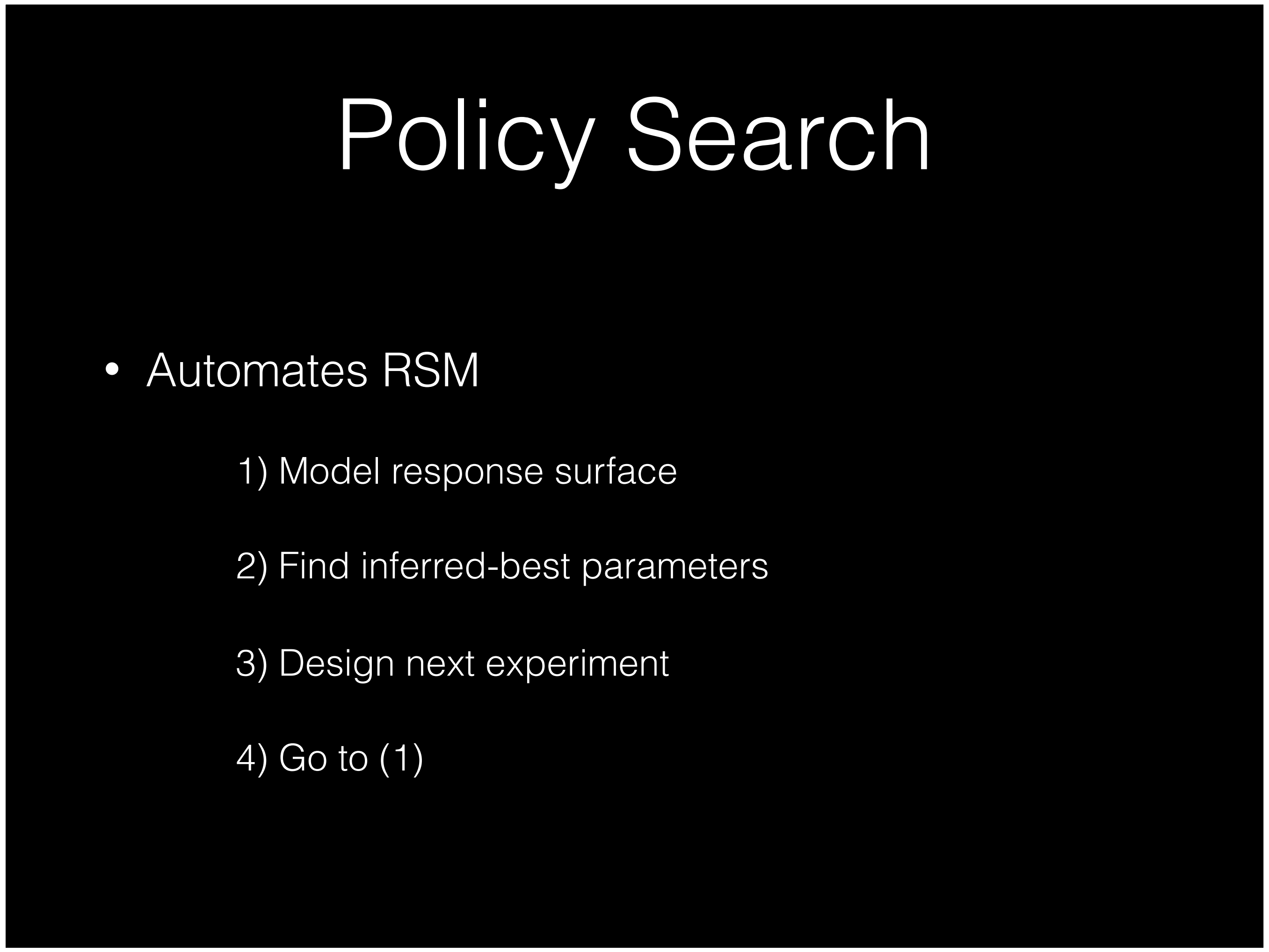
Response Surface Methodology

- Model (regress) quality vs. parameters from D.O.E data
- *Infer* the best parameters from model!
- Verify/Improve: D.O.E. around inferred-best

- Model (regress) quality vs. parameters
- The “best” parameters likely won’t be in the data set.
- Re-center the measurements around the inferred-best. Then take measurements to verify your inference.
- Repeat if desired until your inferred-best stops changing.
- This is an iterative (manual) optimization routine
- At a bank: Designed intraday strategy using simulation costs. Ran with various values of a parameter, modeled quality vs. parameter, and set to inferred-best value. Strategy ran successfully. Did not iterate, however.



RSM adds data efficiency by modeling quality vs. parameters (DEP).



Policy Search

- Automates RSM

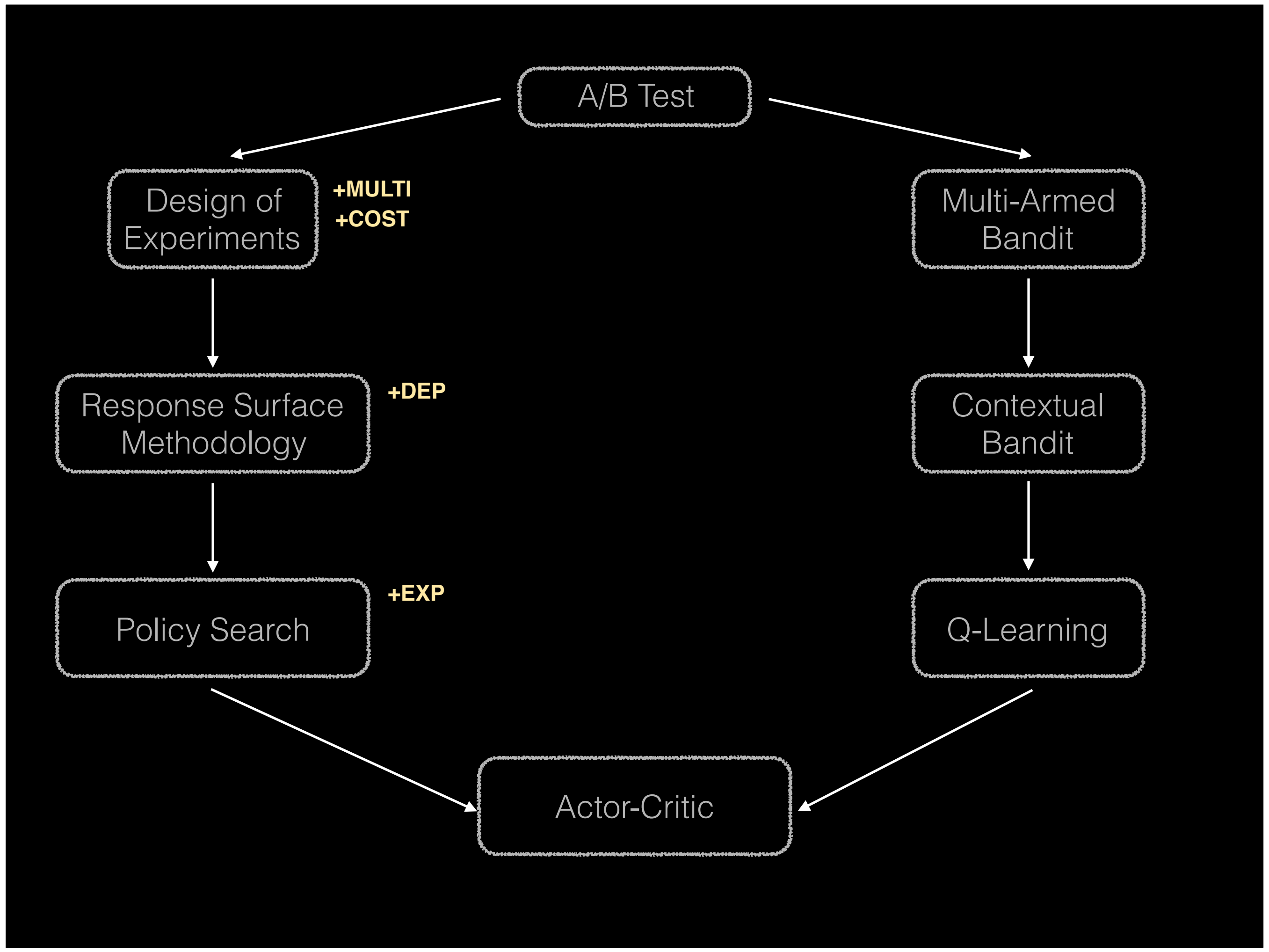
  1) Model response surface

  2) Find inferred-best parameters

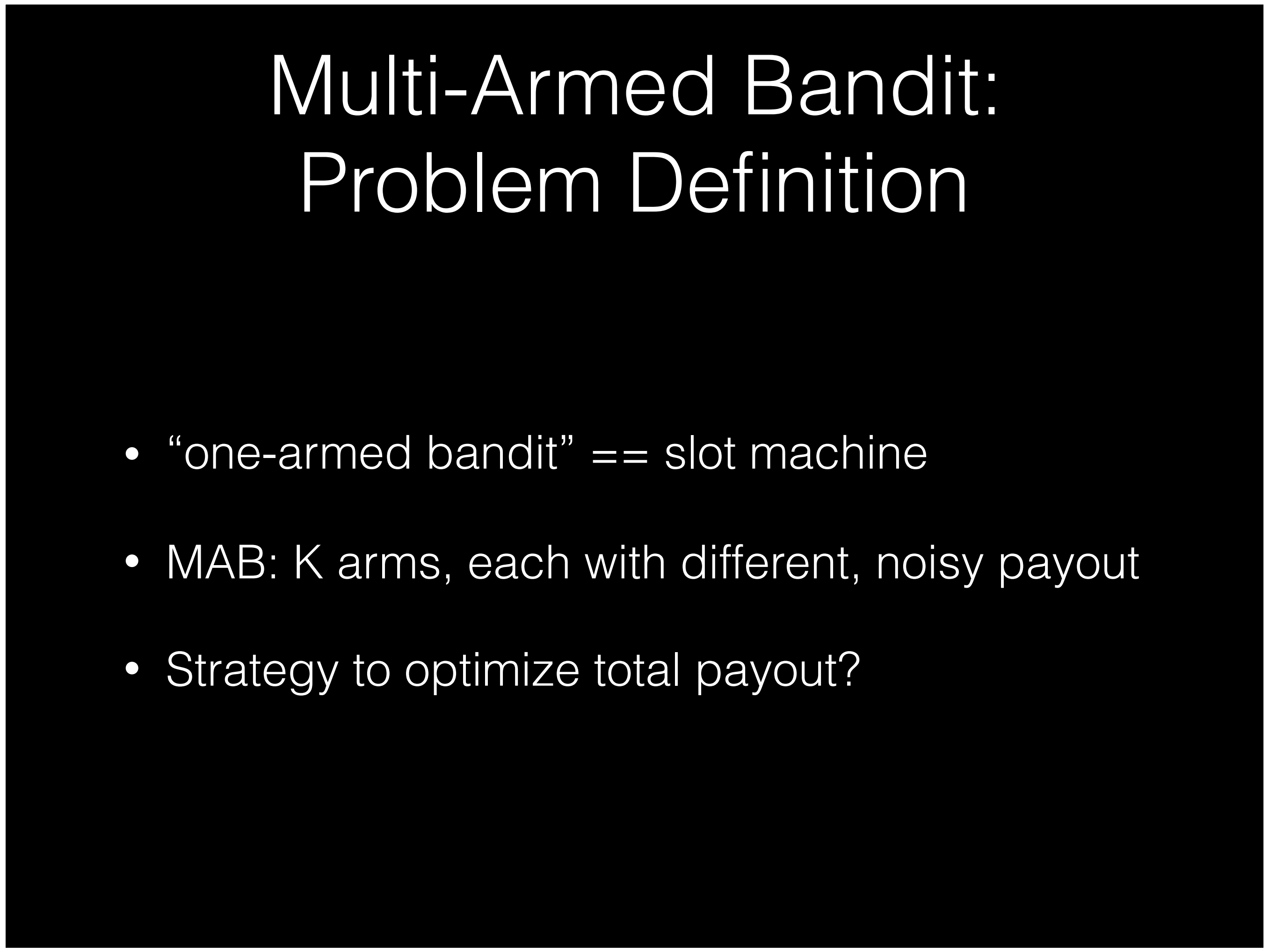
  3) Design next experiment

  4) Go to (1)

- many algorithms; google: Bayesian Optimization, Efficient Global Optimization, Black-Box optimization with expensive objective functions
- (3) tries to optimally trade off the need to collect more data (to build a better model) which has a cost woth the desire to trade at the optimal parameters; aka "exploration vs. exploitation"
- exploitation => higher revenue now; exploration => higher revenue in the future
- accounts for noise / uncertainty in each measurement, so each trading day can use a new experiment design; all data are combined optimally into RSM



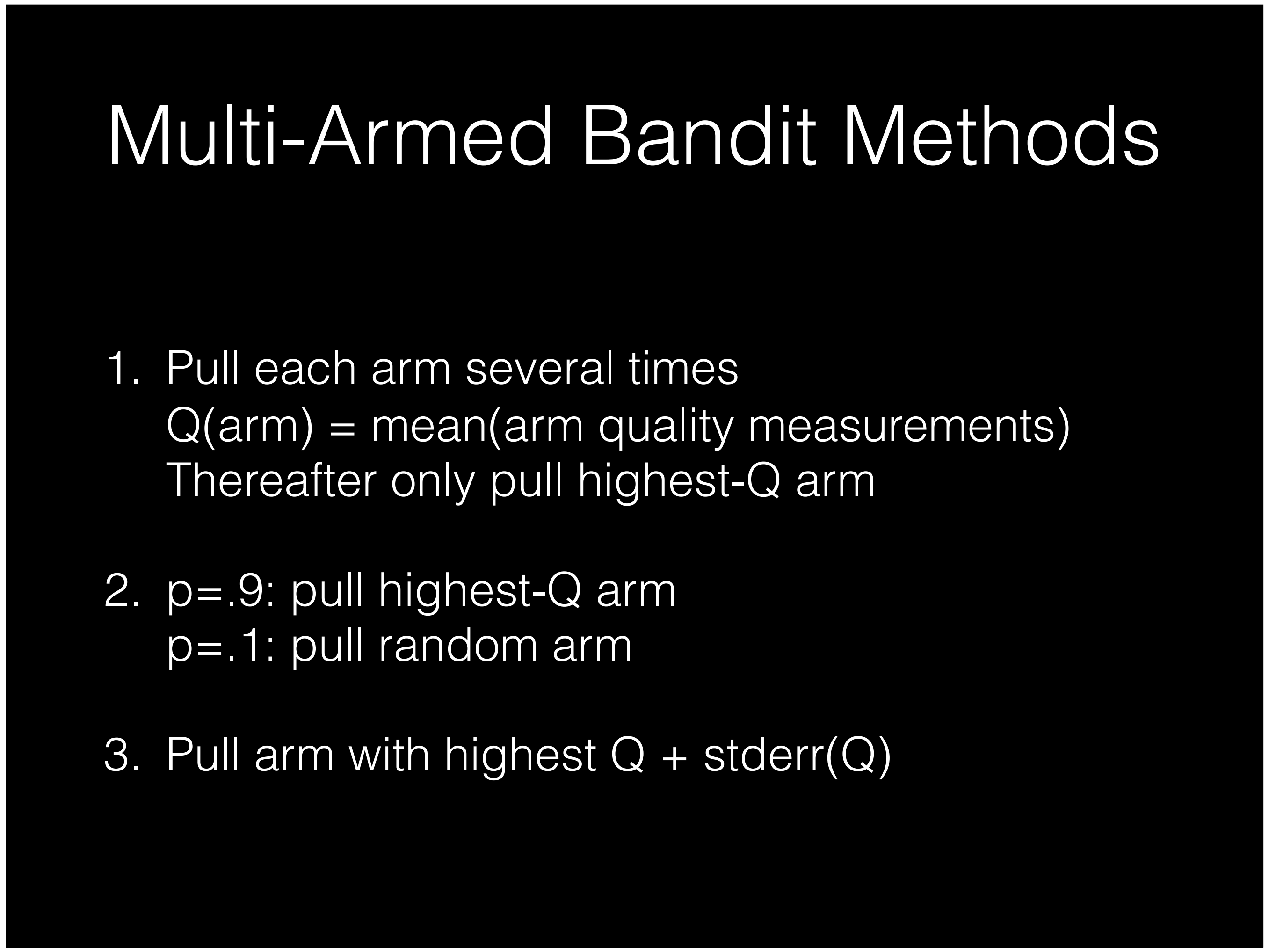
- Policy Search adds a method to optimally design the next experiment (EXP).
- Includes “exploitation vs. exploration” trade-off when designing next experiment.
- Optimization & trading are now one continuous, on-going process. Compare this to A/B testing where there are two “phases”: run the experiment to learn what’s best, then use that information to trade.
- pay some measurement cost today for higher quality tomorrow



Multi-Armed Bandit: Problem Definition

- “one-armed bandit” == slot machine
- MAB: K arms, each with different, noisy payout
- Strategy to optimize total payout?

- MAB is a problem definition
- “MAB methods” are ways to solve that problem
- K=2 arms == a more efficient A/B test
- MAB cares about measurement cost
- MAB handles multiple choices: could be different parameter values (threshold=1,2,3) like DOE, but could also be qualitatively different choices (compare code revisions, hardware, order types, brokers)



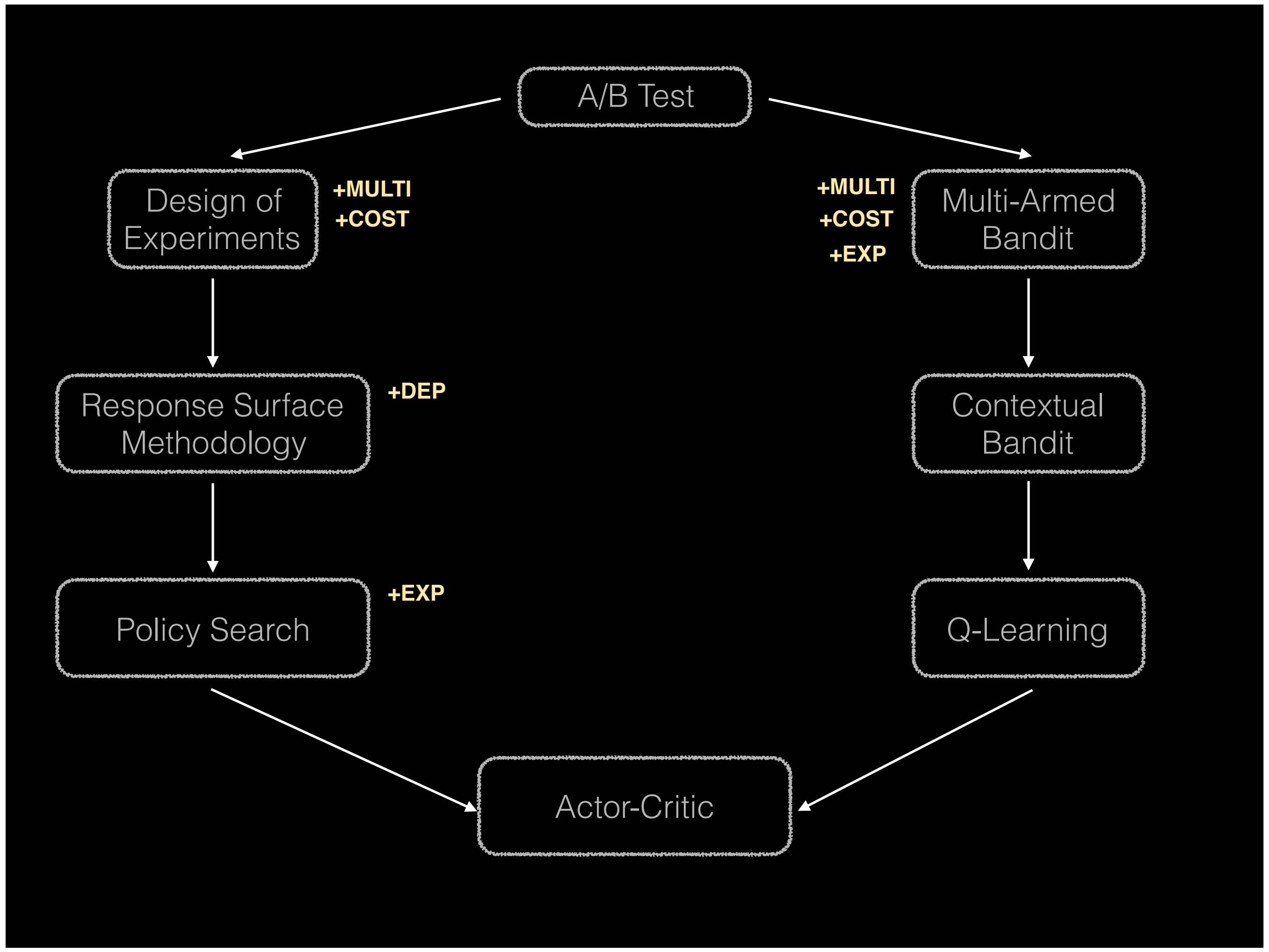
Multi-Armed Bandit Methods

1. Pull each arm several times
   Q(arm) = mean(arm quality measurements)
   Thereafter only pull highest-Q arm

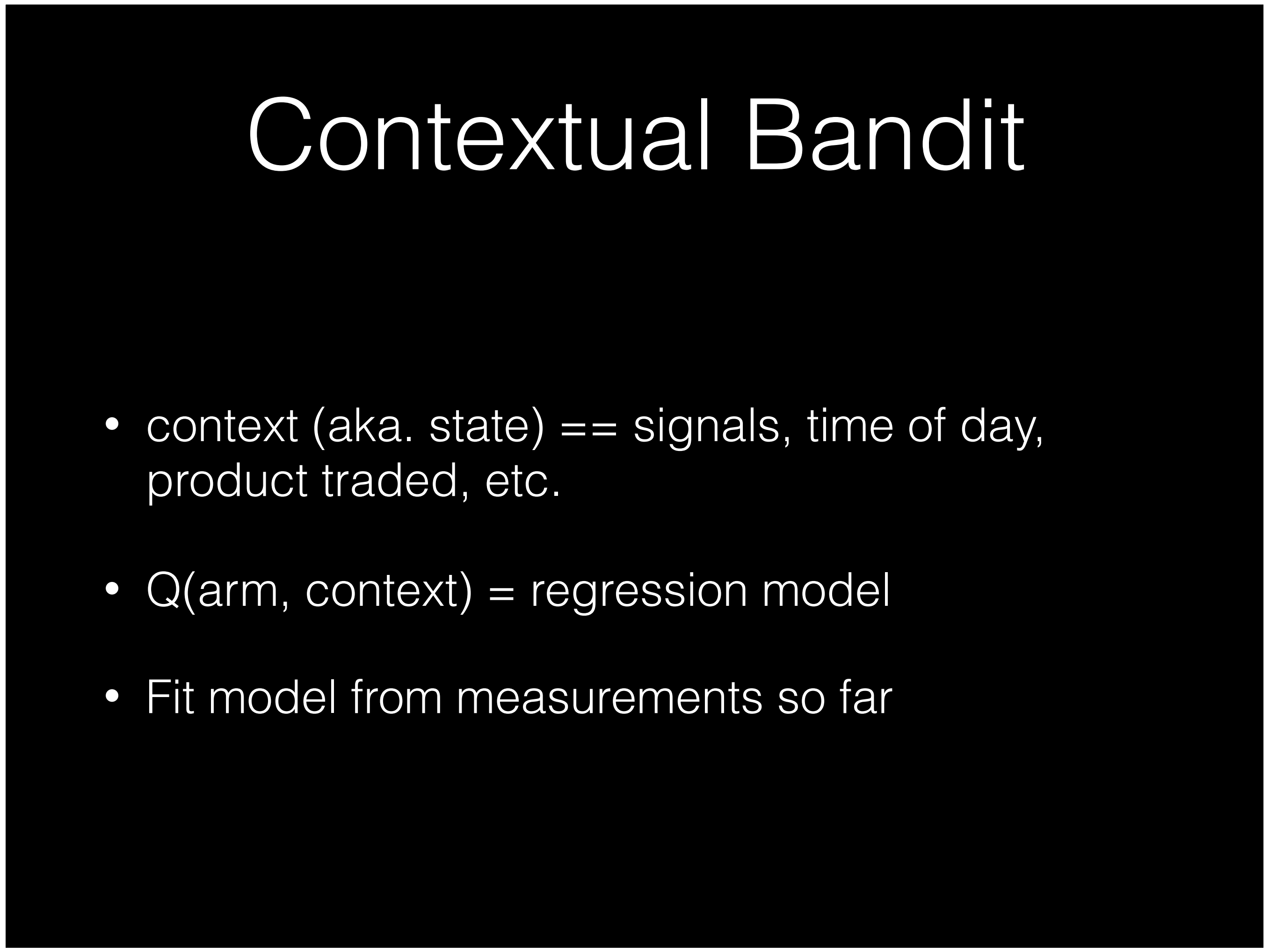
2. p=.9: pull highest-Q arm
   p=.1: pull random arm

3. Pull arm with highest Q + stderr(Q)

- (1) spends a lot of time measuring, but ultimately pulls the best
- (2) “explores” 10% of time to improve estimates, but usually (90% of time) pulls the one we think is best; but never stops exploring
- (3) expression makes exploration vs. exploitation explicit; adds more samples to the noisier estimates (more efficient exploration); eventually stops exploring (more efficient exploitation)
- HFTMM: each “arm” was a small-risk strategy
- HFTMM: would run ~10,000 arms each day dropping worst arms each night and adding new arms each morning



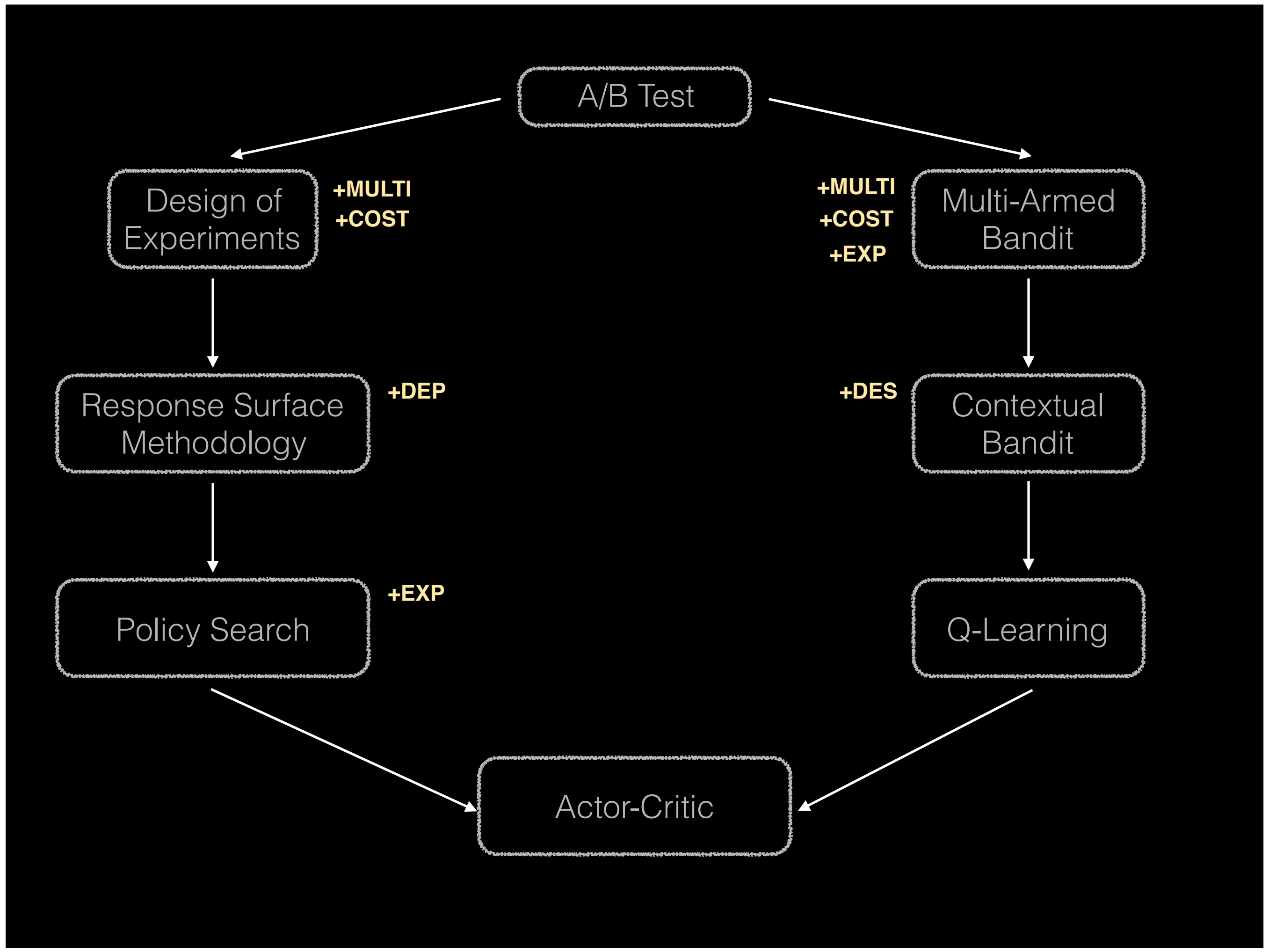
- MAB measures multiple options (MULTI)
- MAB is sensitive to cost of measurements (COST)
- MAB "designs" series of experiments (EXP)
- Compare to DOE:
    - DOE compares real-valued parameter values, designs one big, low-noise measurement
    - MAB compares arbitrarily-defined options, "designs" a series of small, noisy measurements



Contextual Bandit

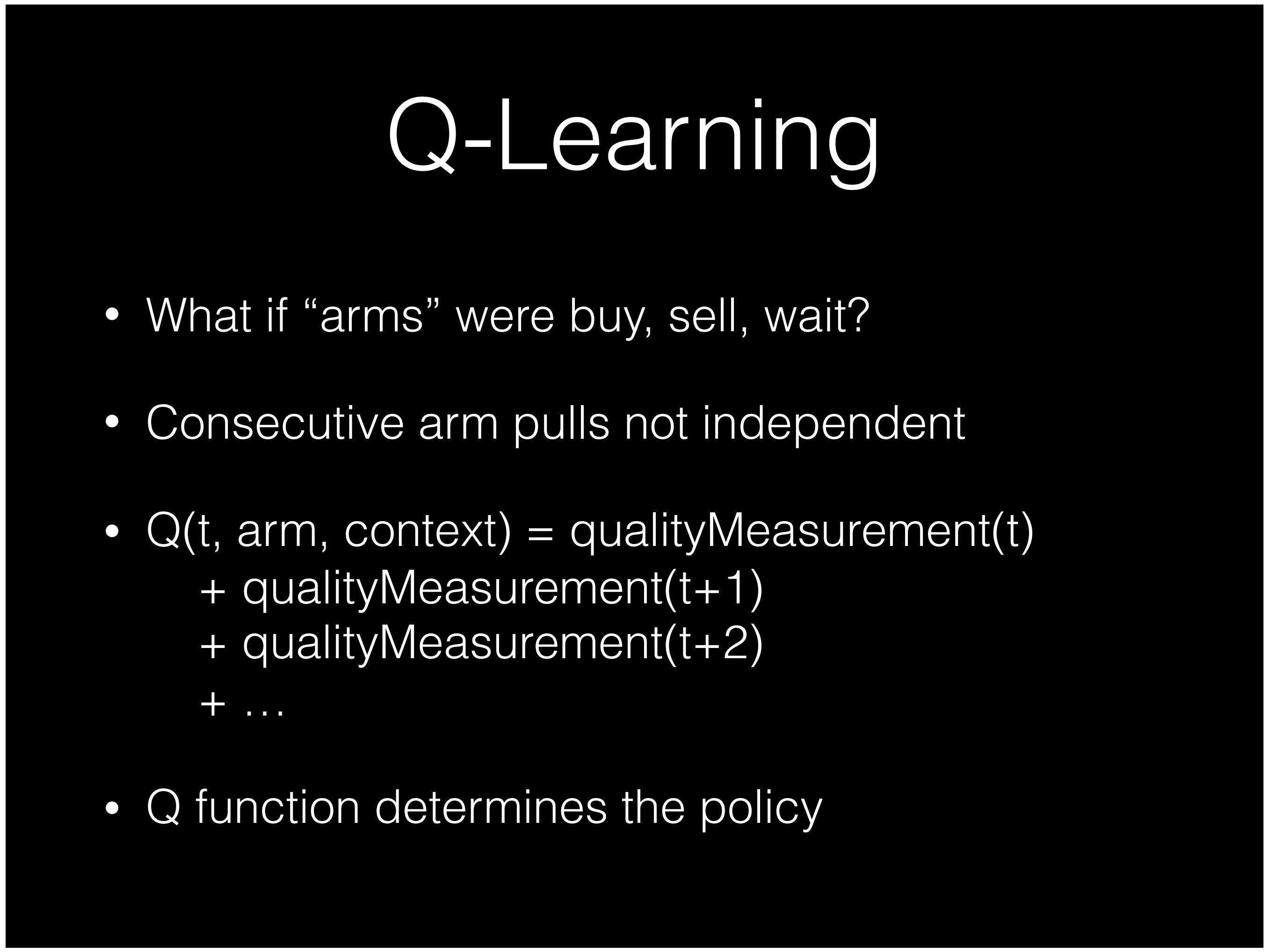
- context (aka. state) == signals, time of day, product traded, etc.
- Q(arm, context) = regression model
- Fit model from measurements so far

- Follow same rules as MAB — 90%/10% or maximal mean+se, except means are replaced by conditional means, i.e. model's prediction of arm quality
- Execution Router: four brokers to route orders to; model slippage of parent order based on broker, time of day, product, other signals; rebuild model every night to “learn” from the day's activity
- ad-hoc in HFTMM: choice of strategies to run was conditioned on time of day, market volume/volatility
-



- Contextual Bandit increases data efficiency by modeling arm quality vs. state (DES) and quality vs. arm (DEP)

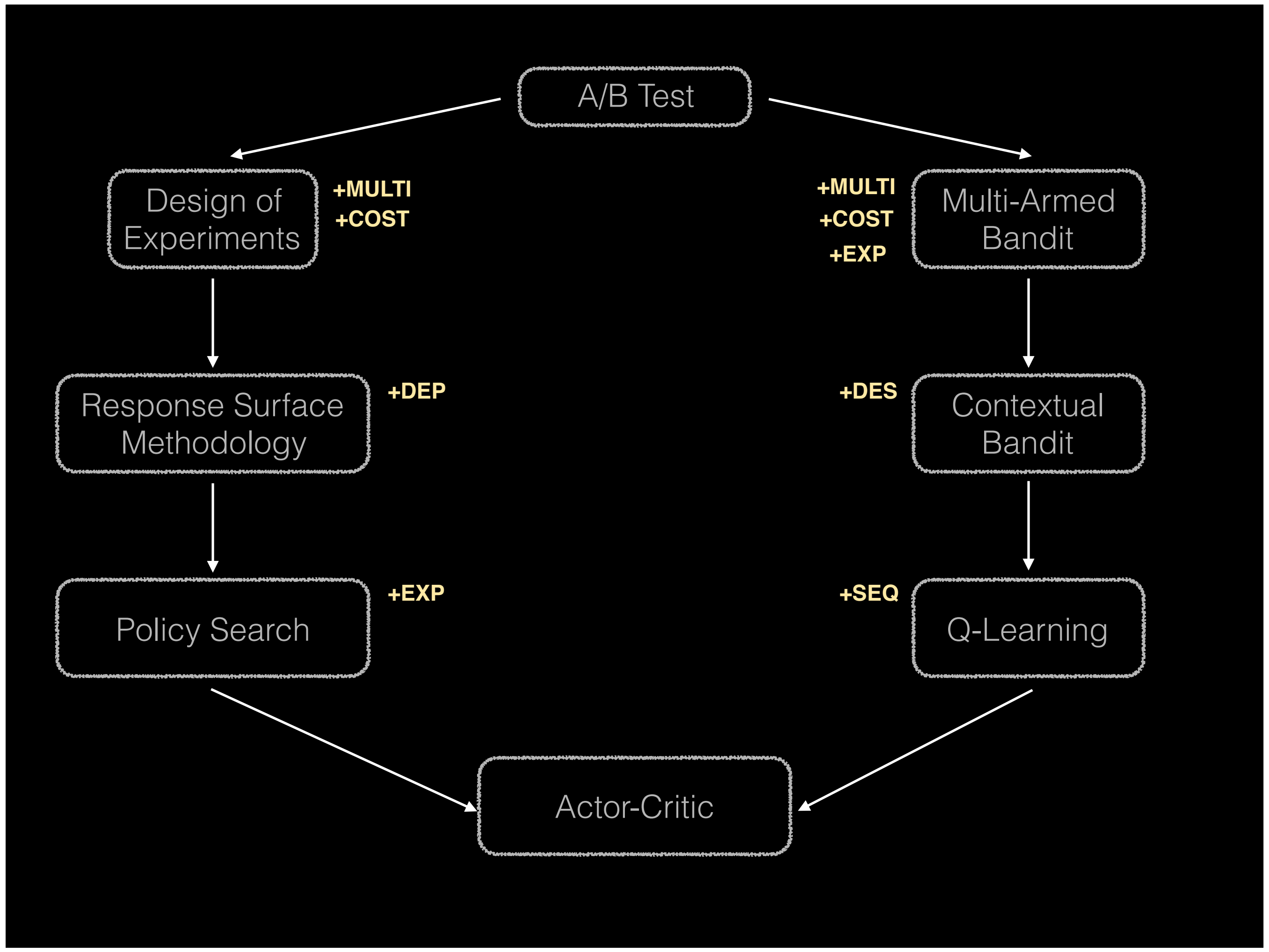
- Notice shift in mindset: “arms” now intraday decisions instead of just candidates for where to fix parameters



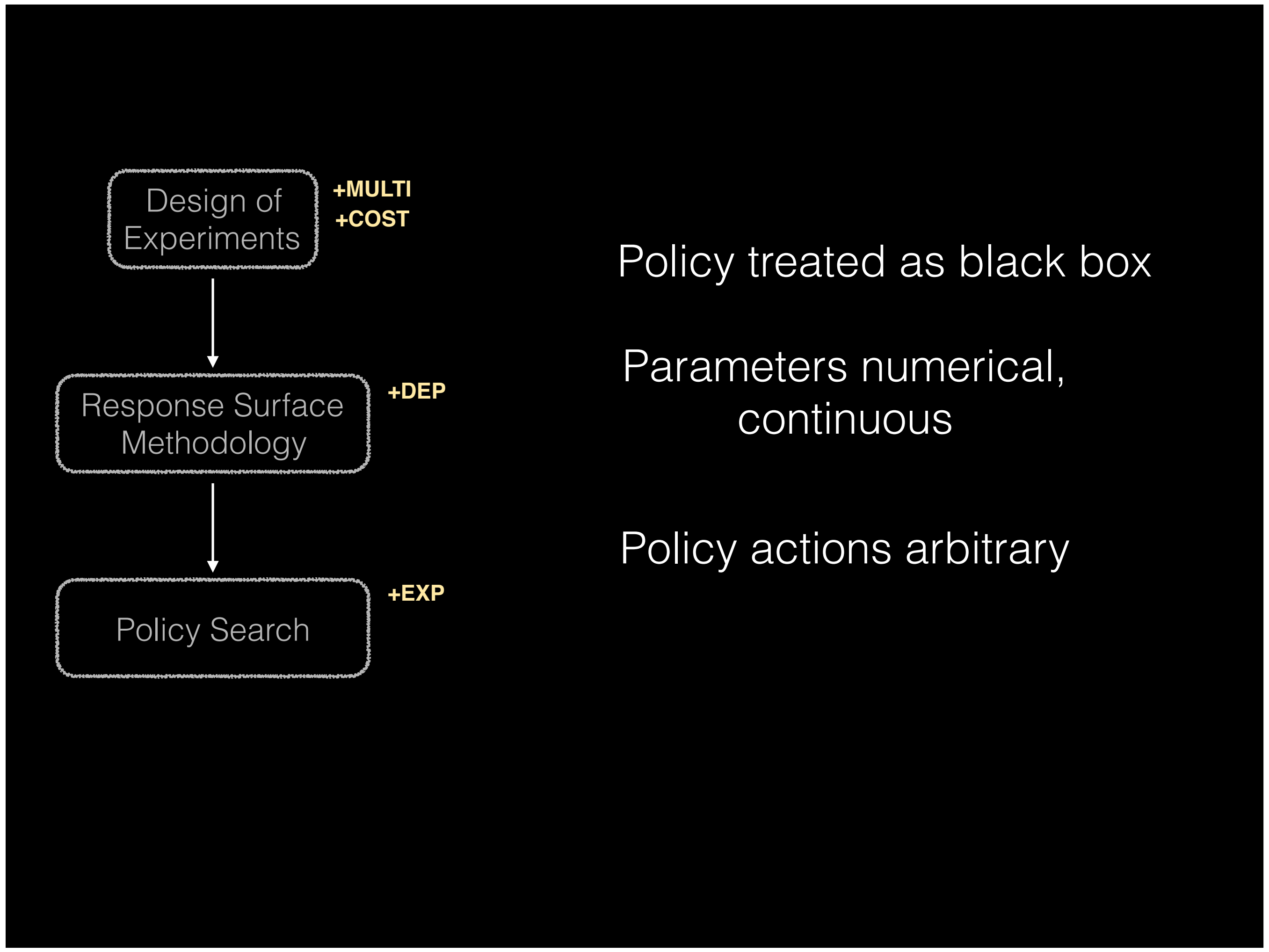
Q-Learning

- What if "arms" were buy, sell, wait?
- Consecutive arm pulls not independent
- Q(t, arm, context) = qualityMeasurement(t)
  + qualityMeasurement(t+1)
  + qualityMeasurement(t+2)
  + …
- Q function determines the policy

- arm pulls were independent in MAB and Contextual MAB
- Q estimate now depends on future contexts *and* your future decisions
- fitting methods can be complex; won't cover here
- Q determines whole trading strategy: Which arm has highest Q? (or highest Q + stderrQ, to include exploration)
- Ex: Smart order router has multiple destinations (arms) to send a sequence of child orders to; at any point in time we ask: "Which is best destination given context and expectations about future contexts *and* our future decisions?"



- Q-Learning models sequences of decisions (SEQ) that are not independent



Policy treated as black box

Parameters numerical, continuous

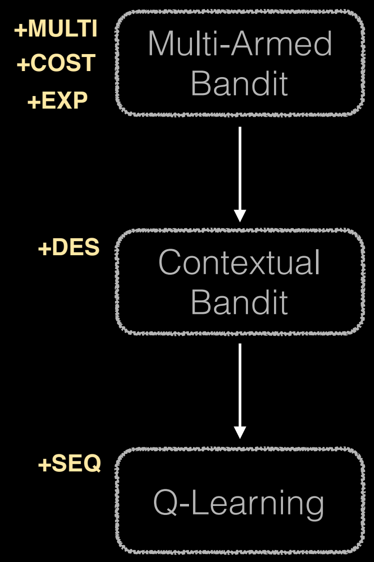
Policy actions arbitrary

- Methods optimize policy parameters
- Don’t consider what policy is doing, just look at your measurement of quality
- Very flexible: Your strategy (policy) can be designed any way you like.
- Data is used efficiently by modeling quality vs. parameters.
- Generally works only with a small number of parameters (~5).

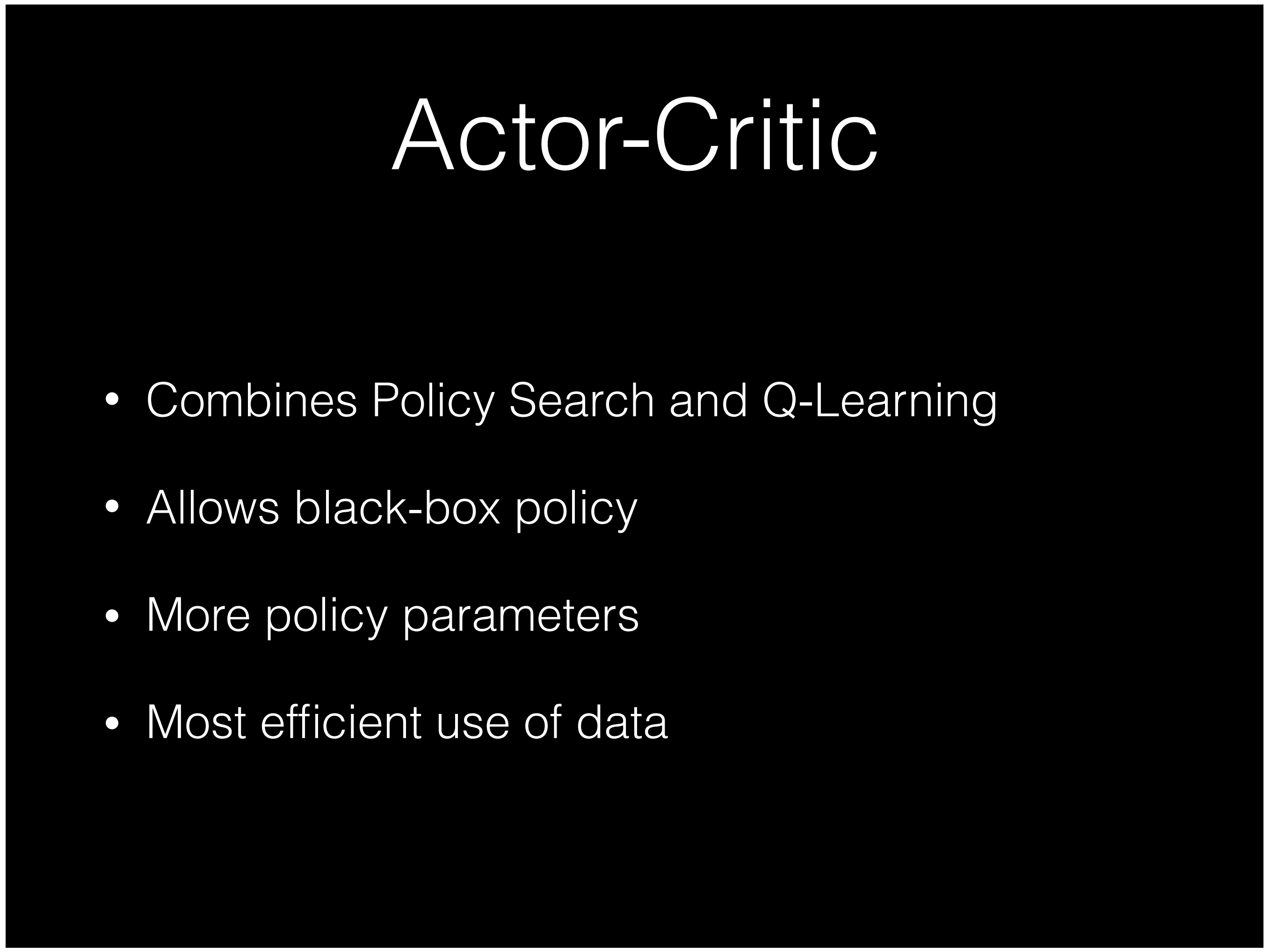
Methods look at policy
step by step

Actions are discrete

Methods analyze actions individually



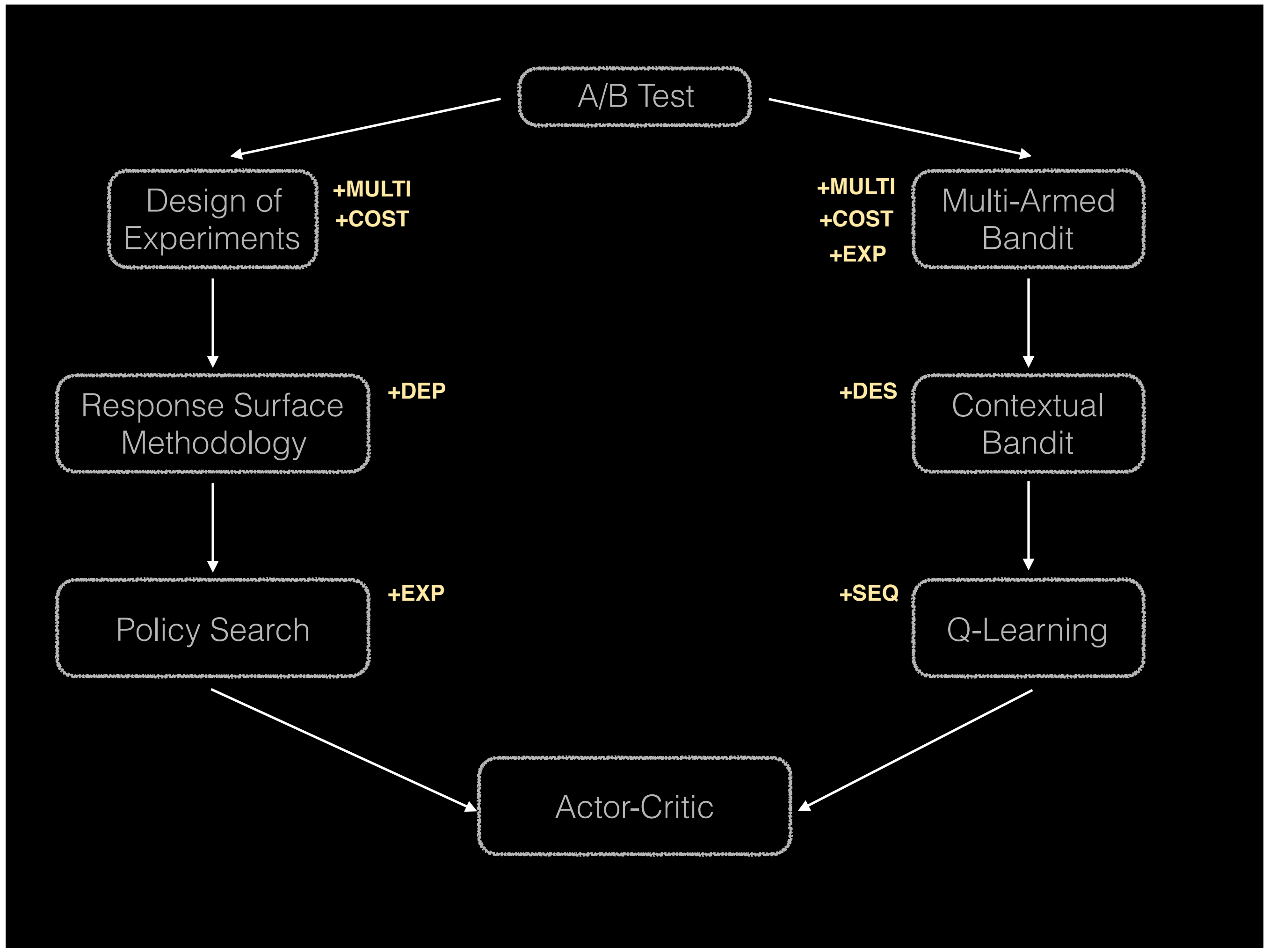
- To use Contextual Bandit or Q-Learning you need to write your strategy (policy) in a compatible way: You need a context (signals) and arms (buy, sell, hold).
- In return you get efficient use of data through models of quality vs context.



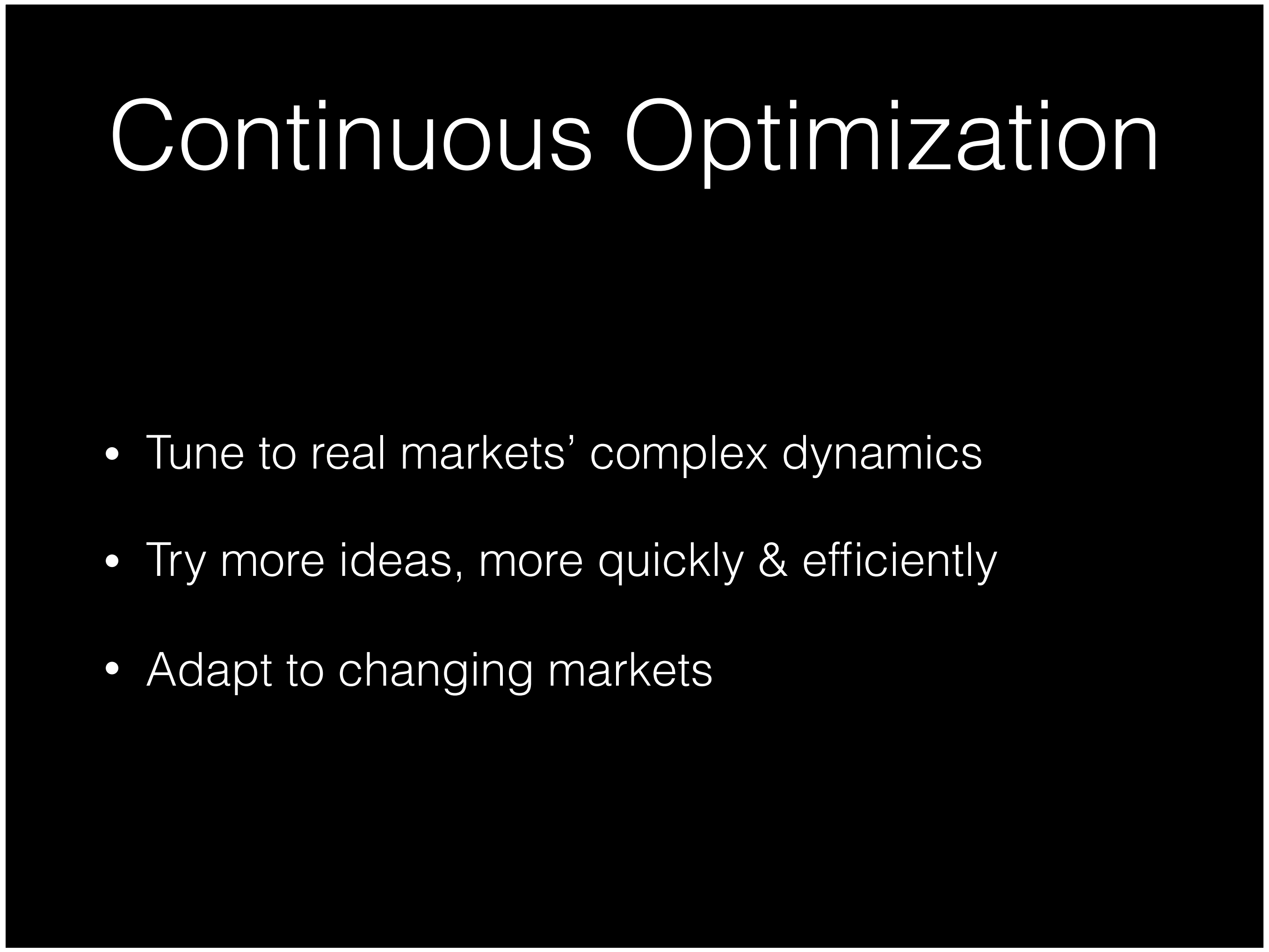
Actor-Critic

- Combines Policy Search and Q-Learning
- Allows black-box policy
- More policy parameters
- Most efficient use of data

- include here for completeness/cu; have not used
- data efficiency comes from modeling Q vs. context and policy parameters
- optimize policy parameters using model Q(arm, context, parameters) as objective instead of simpler model of Q(parameters)
- Ex (imagined, not implemented): 10 exchanges, 10,000 share buy order for MSFT; a = where should I send the order, and how many shares? 10 exchanges* 10,000 shares = 100,000 arms! (100,001, actually, b/c we might choose to do nothing)



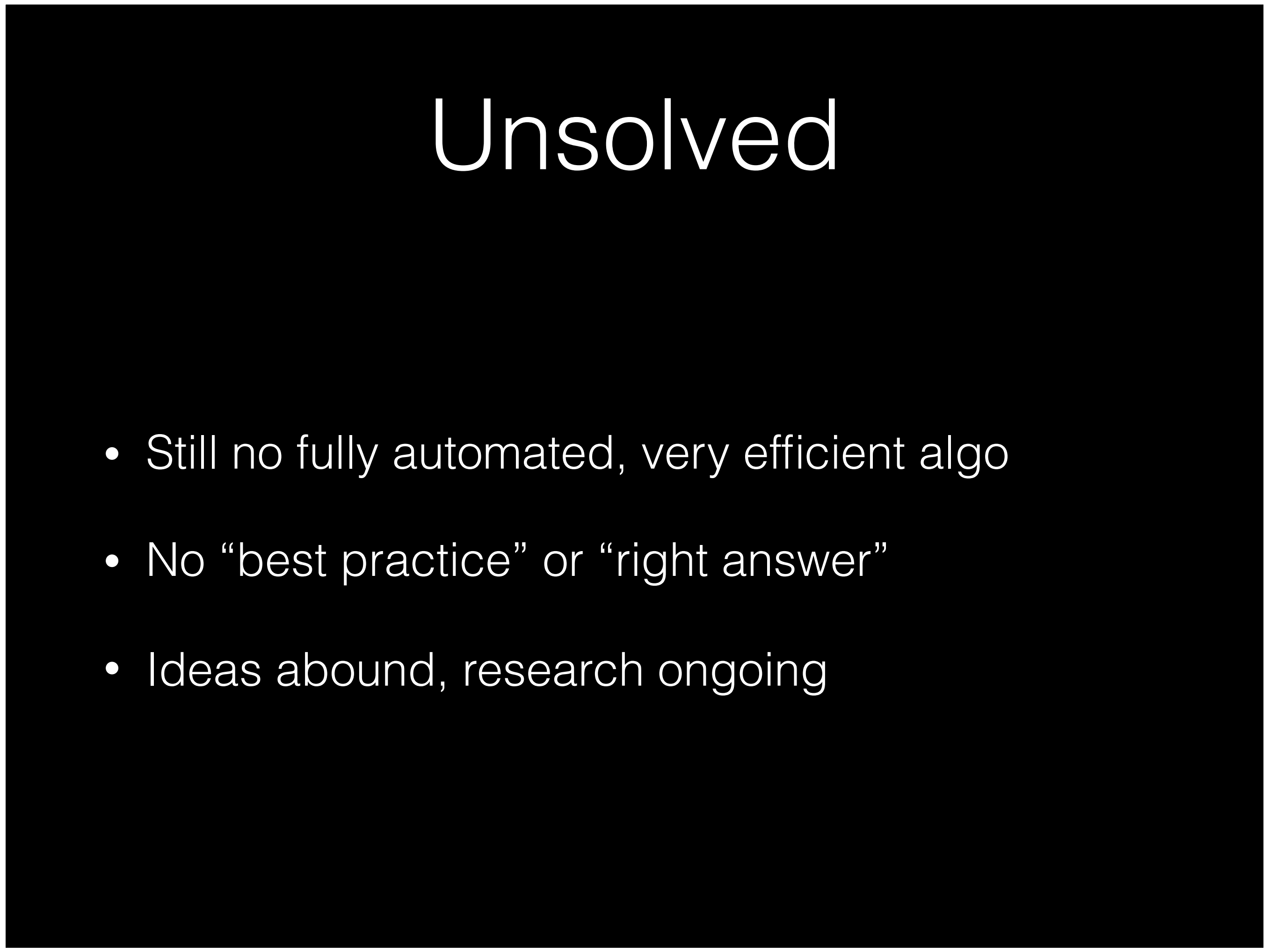
- Themes, going from top to bottom:
  - increasing number of arms/parameters
  - increasing data efficiency: build more sophisticated models of the data collected so far
  - increasing exploration efficiency



Continuous Optimization

- Tune to real markets’ complex dynamics
- Try more ideas, more quickly & efficiently
- Adapt to changing markets

Practical way to view strategy design: as a continuous, never-ending optimization



Unsolved

- Still no fully automated, very efficient algo
- No “best practice” or “right answer”
- Ideas abound, research ongoing

- Sorry!
- For some (for me), that’s part of the attraction.