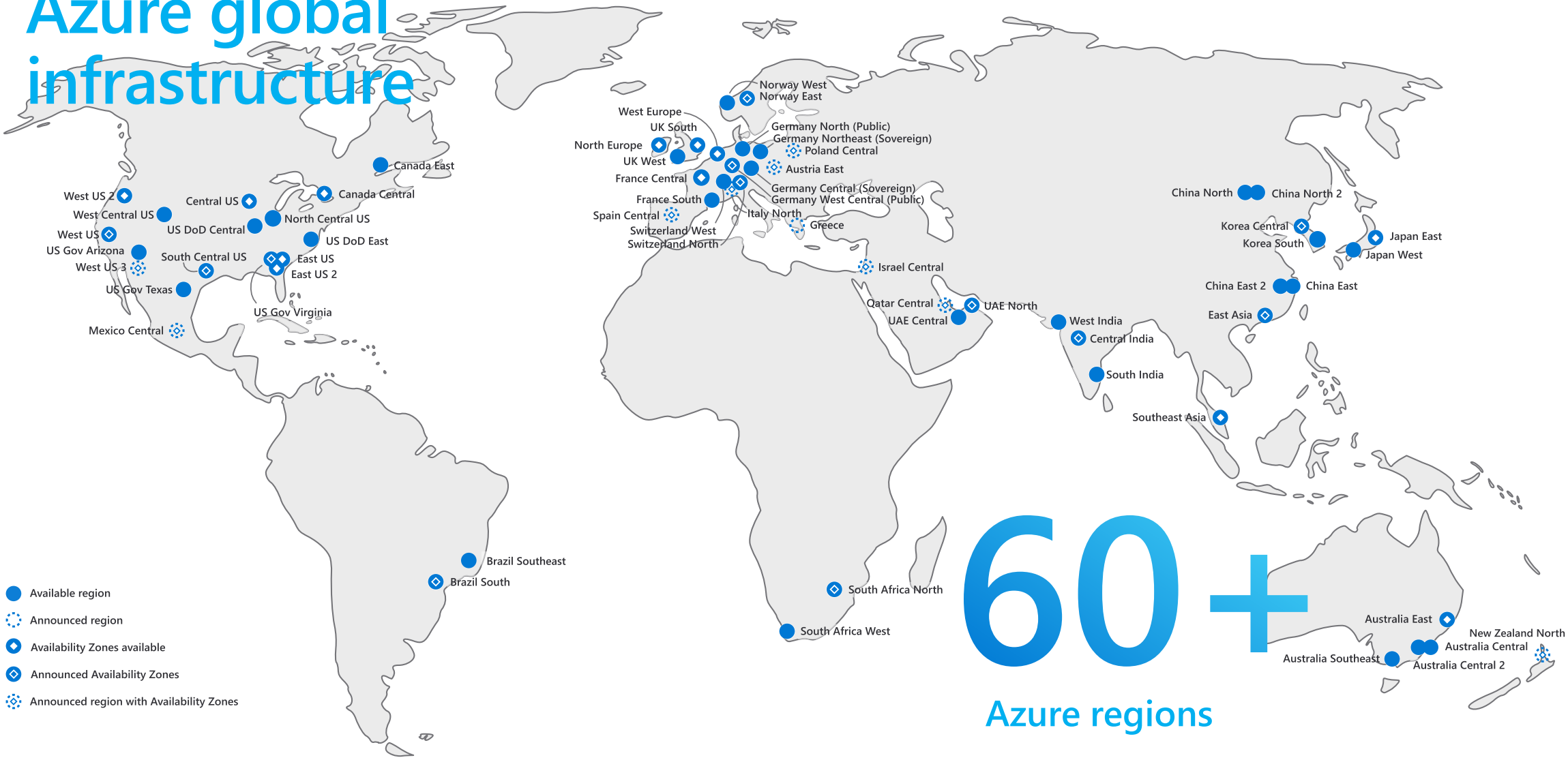
Predictive and Adaptive Failure Mitigation to Avert Production Cloud VM Interruptions

Sebastien Levy†, Randolph Yao†, Youjiang Wu†, Yingnong Dang†, Peng Huang^, Zheng Mu†, Pu Zhao*, Tarun Ramani†, Naga Govindaraju†, Xukun Li†, Qingwei Lin*, Gil Lapid Shafriri†, Murali Chintalapati†

*† Microsoft Azure, ^ Johns Hopkins University, * Microsoft Research*

OSDI ‘20

Azure global infrastructure



60+

Azure regions

Heterogeneity

All nodes are different

Different characteristics: # cores, total memory …

Different hardware type / vendors

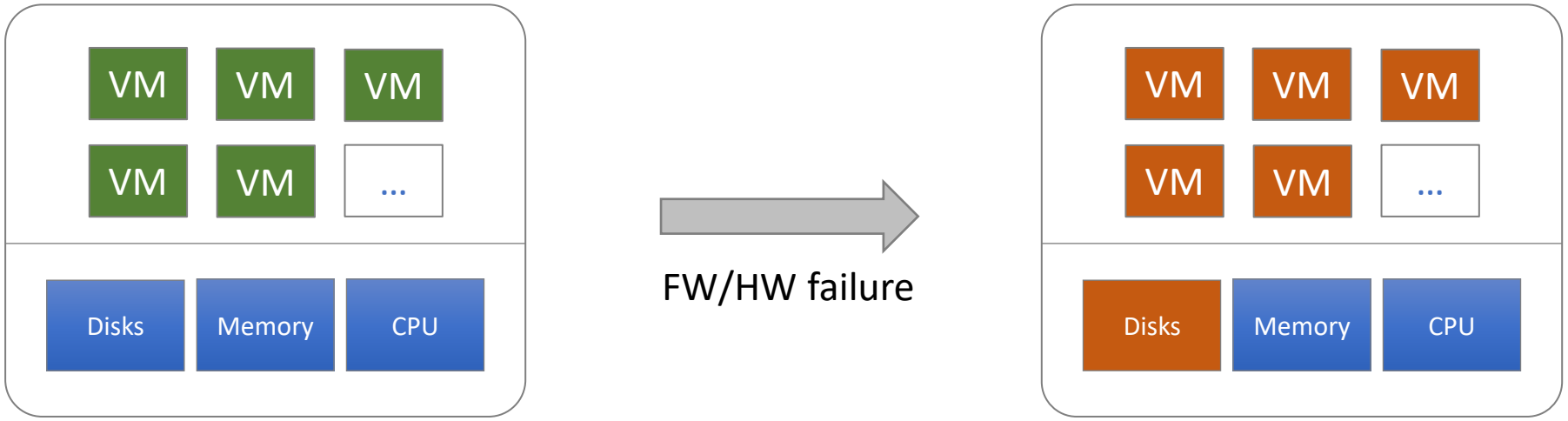
Different workload patterns

Different health history

Measuring Customer Experience

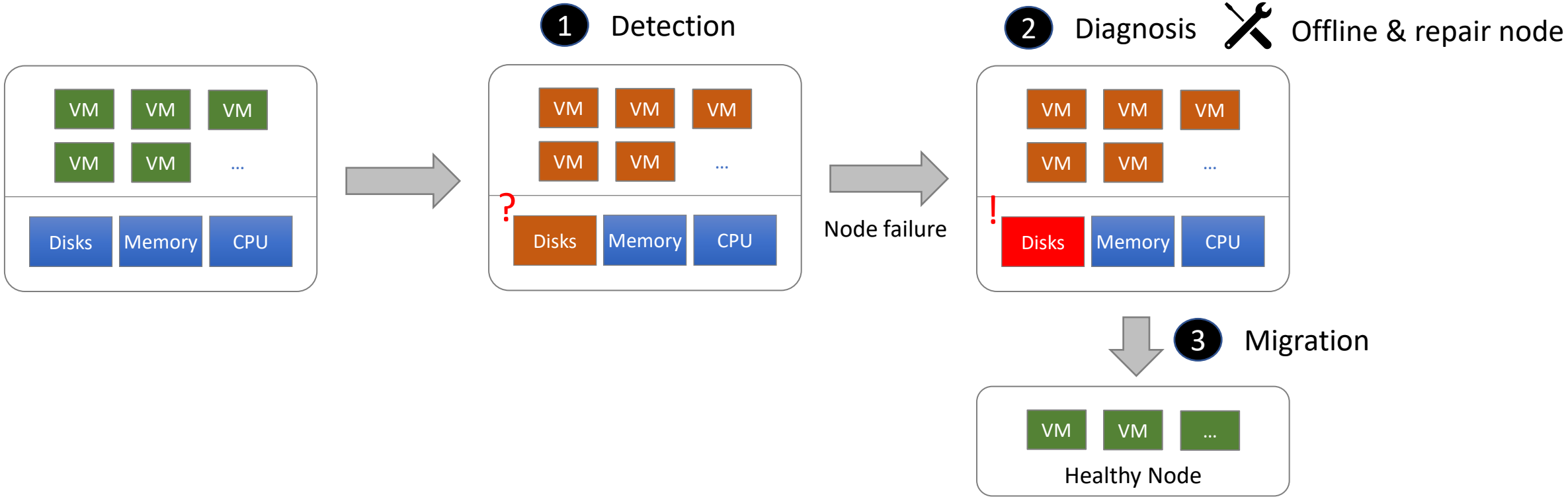
- We need to monitor VM availability: down time / up time
- But each VM interruption causes significant impact to customer
  - Disrupt user experience (ie gaming)
  - Applications take time to recover
  - Customer frustration would come in case of repeated reboots
  - Two short interruptions are more impactful than one longer one

Impact of Node Failure



- Node-level failure ➔ bad impact for every VM it contains
- We need to predict failures and take the appropriate mitigation actions

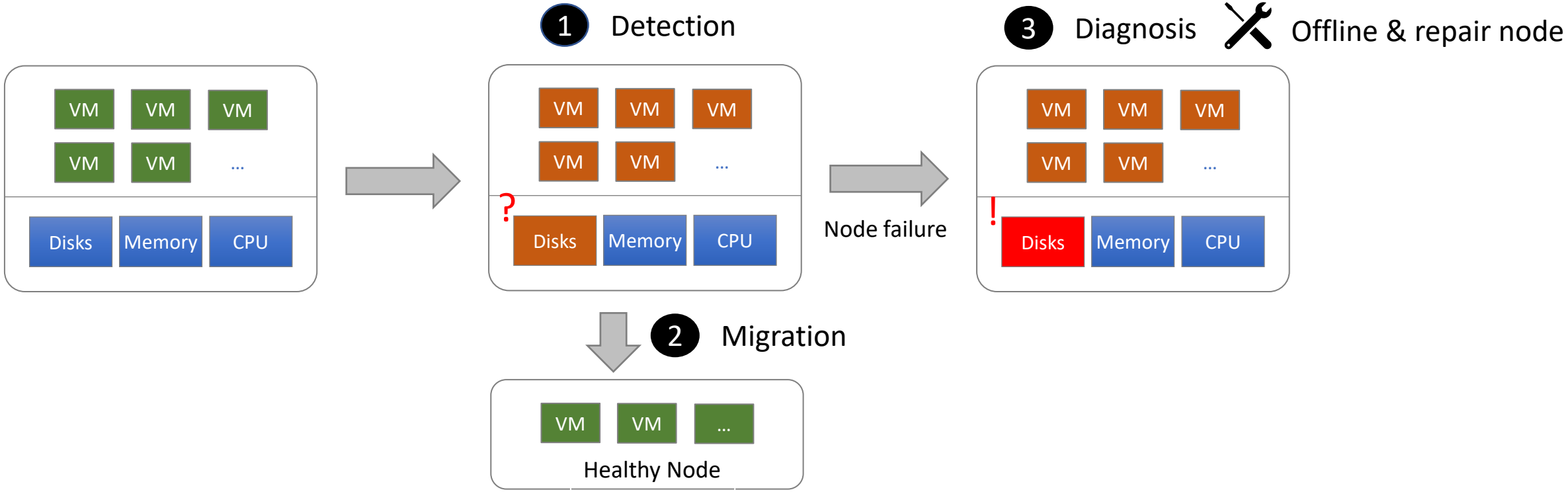
Traditional Operation Workflow



❌ **All VMs reboot**

❌ **Long VM downtime**

Attempt 1: Mitigation Before Diagnostics

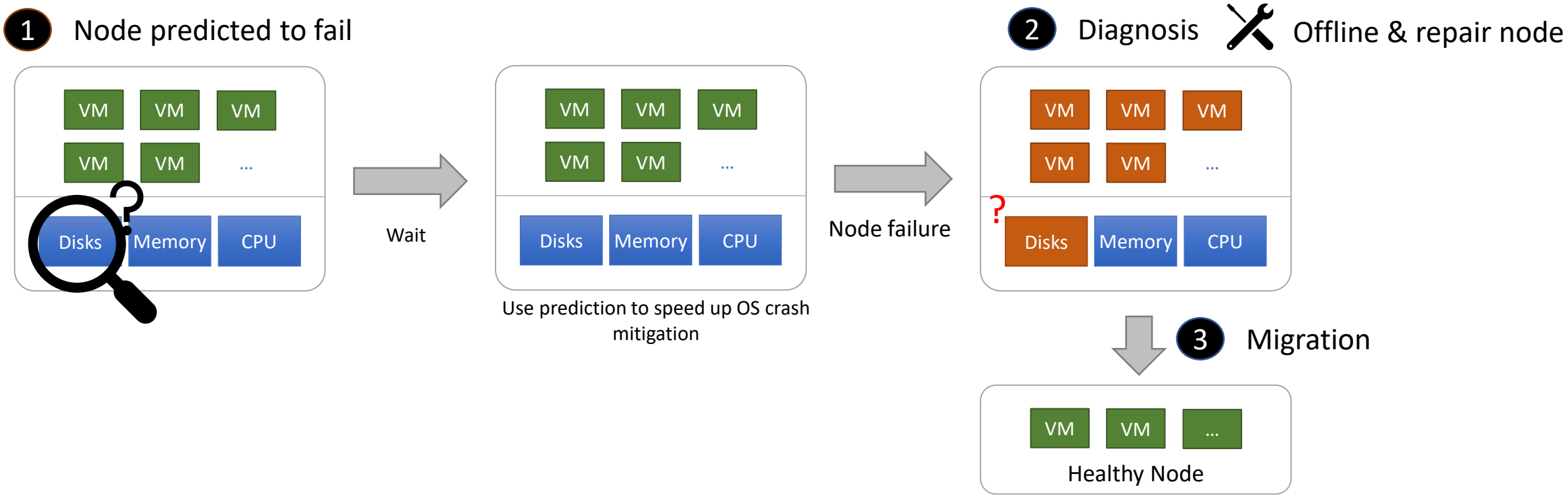


❌ **All VMs reboot**

✅ **Short VM downtime**

❌ **Mitigation can be better in some cases**

Attempt 2: Prediction of Node Failure

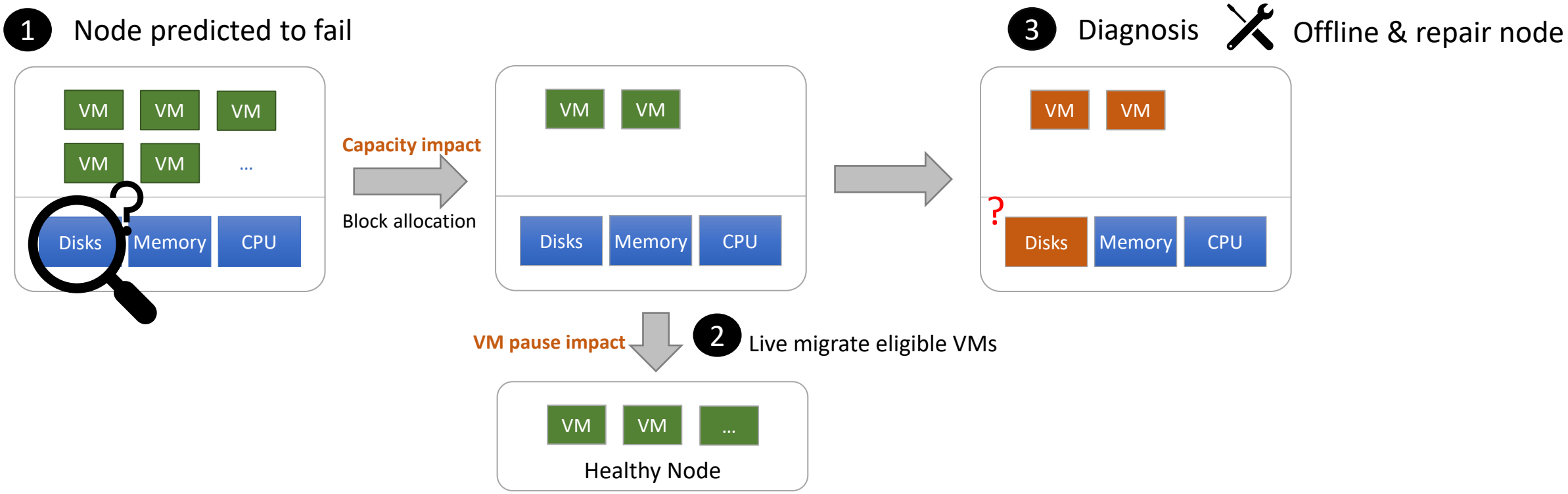


❌ **All VMs reboot**

✔ **Shorter VM downtime**

❌ **Mitigation suboptimal for wrong prediction**

Attempt 3: Prediction + Static Mitigation



✓ Some VMs did not reboot

✗ Other VMs reboot

✓ Shorter VM downtime

✗ Increased impact for false positives

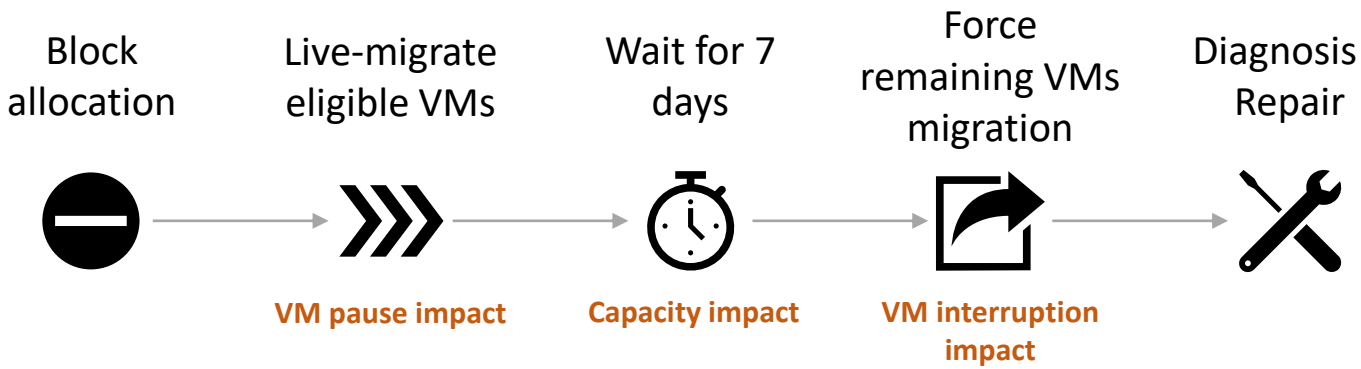
Can we do better?

Mitigation Tradeoff

- Can a mitigation be effective for all kind of scenarios?
- How to confirm a mitigation is effective?
- Even if a mitigation is effective, can we do better? Can it change in time?

Mitigation Tradeoff: No One-Size-Fits-All

How to mitigate predicted node failures: reasonable proposal



? If failure is not imminent, do we need to completely block allocation?

? How long should we wait for customers to intentionally move their VMs?

? If node has still not failed after 7 days, could it now be healthy?

How To Estimate Mitigation Efficacy?

- Heterogeneity: Multiple factors will impact the mitigation Efficacy
  - Live migration success rate depends on available capacity, hardware health, workload on the node
  - Allocation relies on a complex optimization logic and on stochastic customer demand
  - Prediction false positives are unavoidable and can depend on unobservable signals

Key Insights

“In a heterogeneous and ever-changing cloud system, the effectiveness of a mitigation action is often **probabilistic**.”

“To select the (near-)optimal mitigation, each possible action needs to be compared **at-scale with production workload**.”

Solution

Continuous probabilistic online minimization of customer impact

Adapt to ever-changing cloud behavior

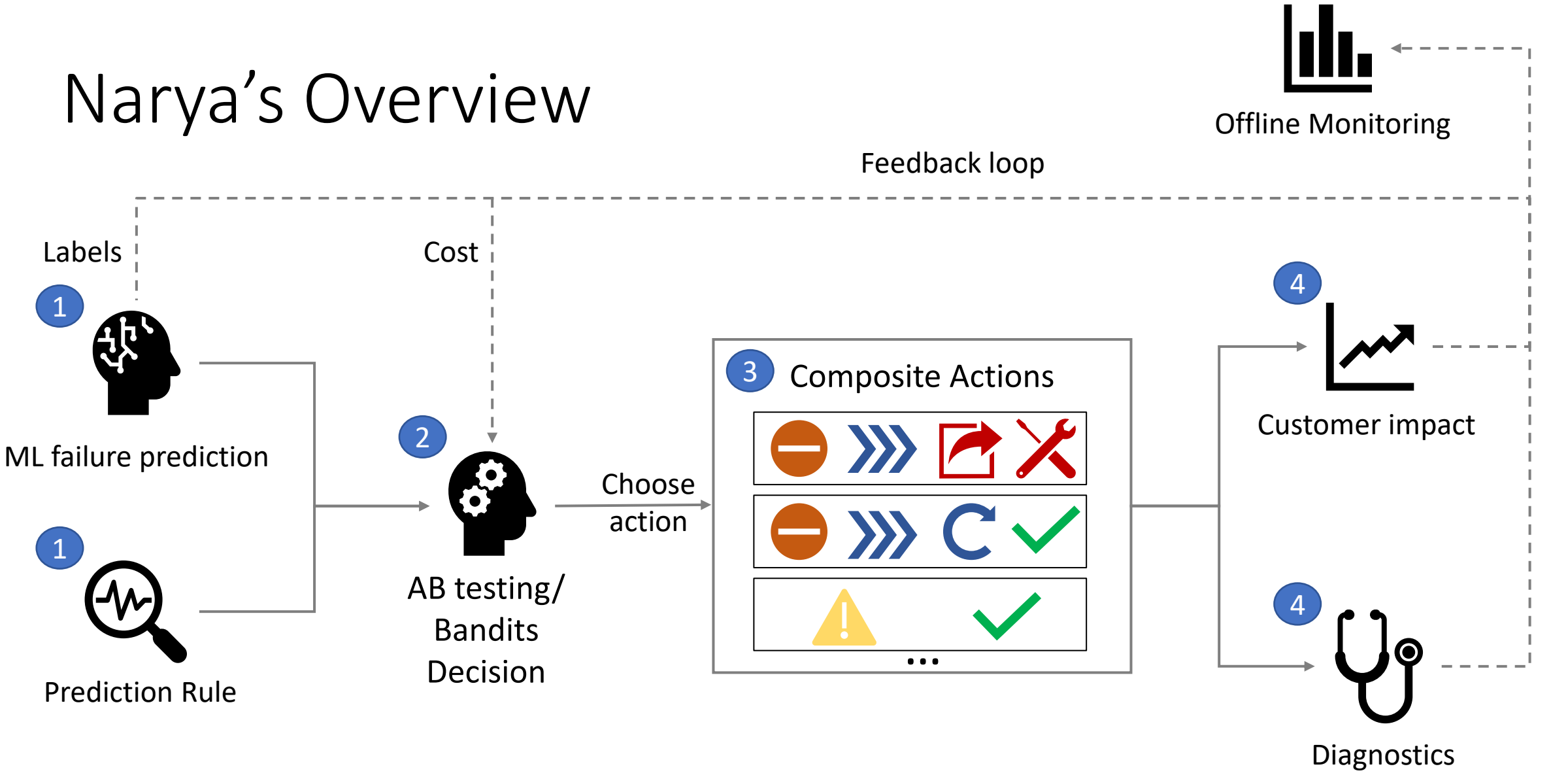
Account for node heterogeneity of Azure's fleet

Online testing of the different options

Different mitigation actions depending on the type of predicted failure

Focus on impacting failures

Narya's Overview



Main Challenges

Define	Define customer impact as a generic metric
Test	Safely test action in production
Balance	Balance exploring actions and exploiting the best so far
Adapt	Adapt our decision to system changes
Fast	Fast and scalable mitigation decision

Measuring Interruptions

- **Interruption Types:**
  - VM reboot, IO pause, VM temporarily freeze / blackout
- **Service Availability**
  - Time duration based
  - Short VM downtime DOES NOT imply short customer service downtime.
  - Each VM service downtime requires customer service to rehydrate their state
- **Interruption Count**
  - Event count based
  - Each VM interrupt impacts customer service once
  - More realistically reflects the customer pain

Defining Customer Impact

- VM interruptions are the main negative impact to customer
- We define Annual interruption rate (AIR)

$$AIR = \frac{\#\ interruptions\ in\ T}{Total\ up\ time\ in\ T} \times 365\ days\ \times 100$$

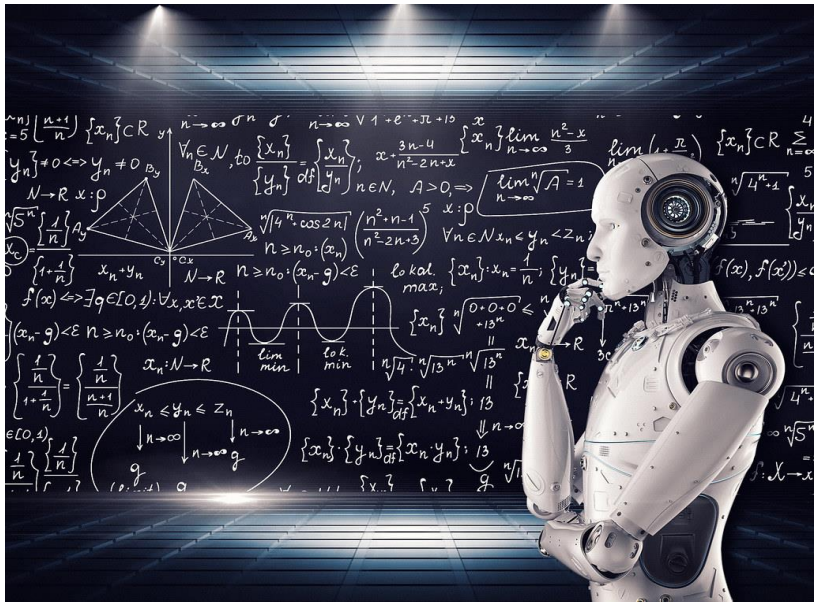
When scoping it to a contribution per node and removing the constants, we get:

Cost = # VM interruptions per node

Narya’s Prediction

Expert Rules

+



ML Prediction

Motivation To Use ML Prediction

- Expert rule
  - Setting a threshold on a single predictive signal. Ex: Disk low spare space
- Limitations of expert rule
  - High variance of prediction horizon - actual failure time also depends on factors
  - Difficult precision-recall tradeoff – strict rules result in low recall, loose rules yield low precision
- What if we look at all signals before the actual failure and use those signals to predict the failure?

ML Model Highlights

- Comprehensive list of signals that cover OS layer, driver layer and device layer observation.
- Predict server level failures that has customer impact
- Non-handcrafted features
  - Spatial attention
  - Temporal convolution

Longer Lead Time

Higher Precision

Higher Recall

Diversity of Mitigation Actions

Allocation change


Allow allocation


Avoid allocation*


Block allocation

VM migration


Live migrate VMs*
(unallocatable node only)


Service heal VMs
(unallocatable node only)

Node reboot


Soft kernel reboot*
(unallocatable node only)


Hard reboot


Offline + repair
(empty node only)


No Impact


Pause Impact


Soft capacity impact

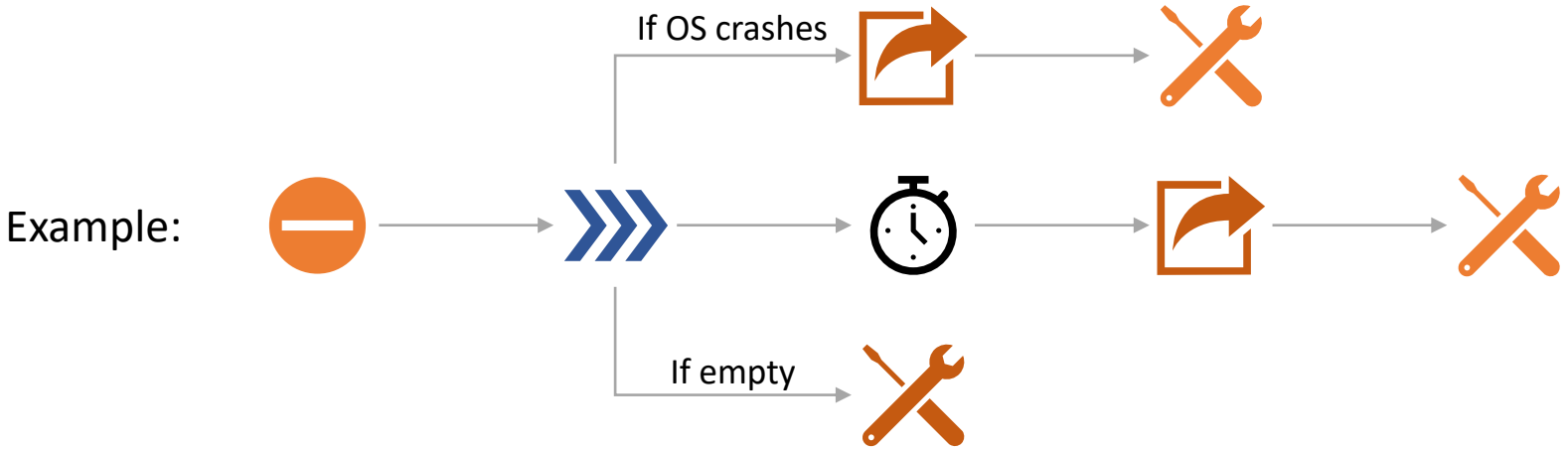

Hard capacity impact


VM reboot impact

Non-deterministic action

Narya's Mitigation Actions

- Use composite actions (sequence of actions)
  - Avoid illogical scenarios: migration without blocking allocation, repeated reboots …
  - Add safety constraint to minimize the cost of exploration
  - Easy override priority if several rules overlap
  - Simpler learning



Composite actions' duration is typically in the order of days

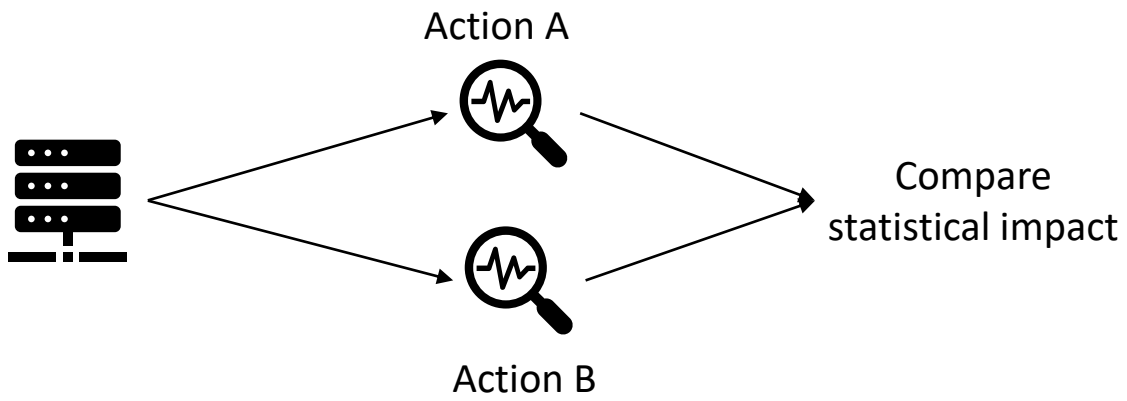
Narya's Decision Engine

- Mapping each predicted bad nodes to the best composite mitigation action to minimize customer pain

- Two algorithms:
  - **AB testing:**
    explore all different possibilities, observe the customer pain metric(s) then use the best action if it exists
  - **Multi-Armed Bandit (Thompson Sampling):**
    learn the right tradeoff between exploring new actions and exploiting the estimated best one based on a single customer pain metric

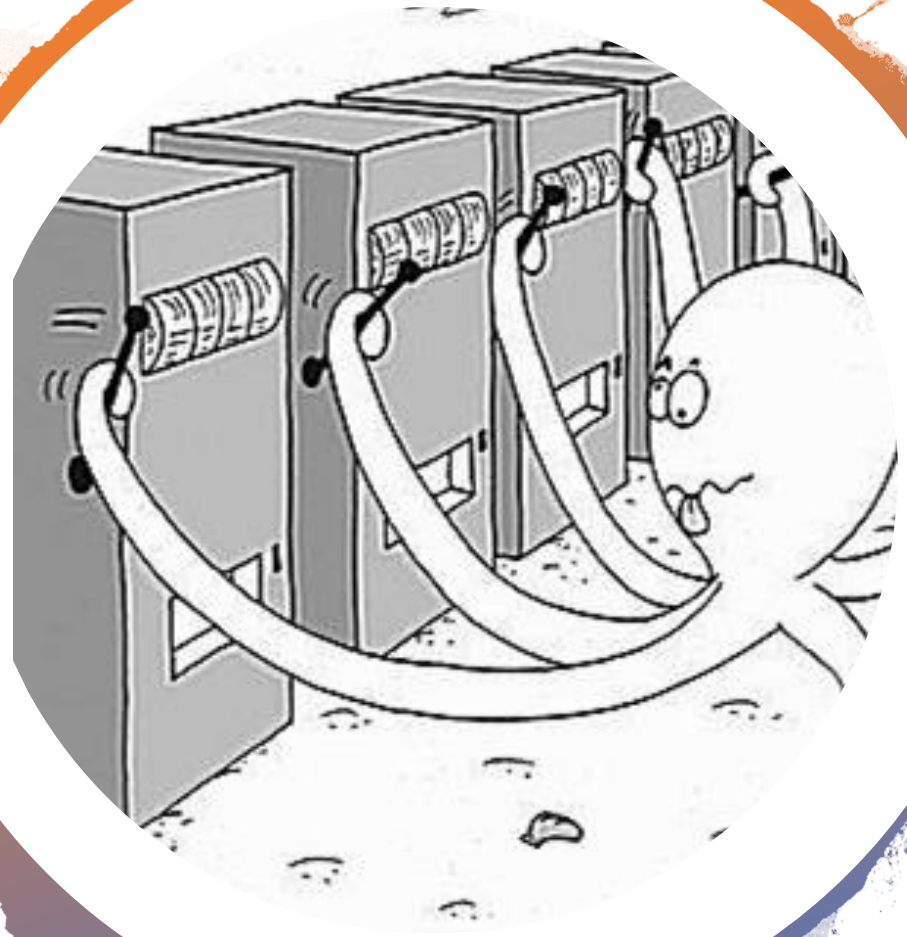
AB Testing

1. Sample action with preconfigured probabilities
2. Observe customer impact within an observation window
3. Hypothesis test between cost of all actions
4. If statistical significance, use the optimal action



Adapting AB Testing To Our Scenario

- Cost attribution: VM reboots during the observation window
  - knowing which interruptions is caused by the action is not possible
- Stickiness: same node always use the same action
  - otherwise we would violate the iid assumption
- Decision vs action: we observe the consequence of every decision
  - even if it is skipped, delayed or overridden



Bandit Motivation

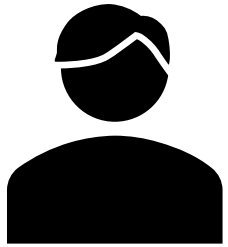
- AB testing limitations
  - Does not leverage early observation before statistical significance
  - Cannot adapt to change after statistical significance

- **Multi-armed bandit:** dynamically learn the probability based on observations
  - ✓ Leverage early observation in exploration
  - ✓ Adapt to change in exploitation

Bandit Framework

Reward = - Customer impact (Cost)

Machines = Available composite actions



Pull / Game = New node mitigation request

Each prediction rule is a different bandit experiment

Bandit Adaptation To Our Scenario

Accommodate temporal change: exponential decay

Delayed rewards

Bandit stickiness

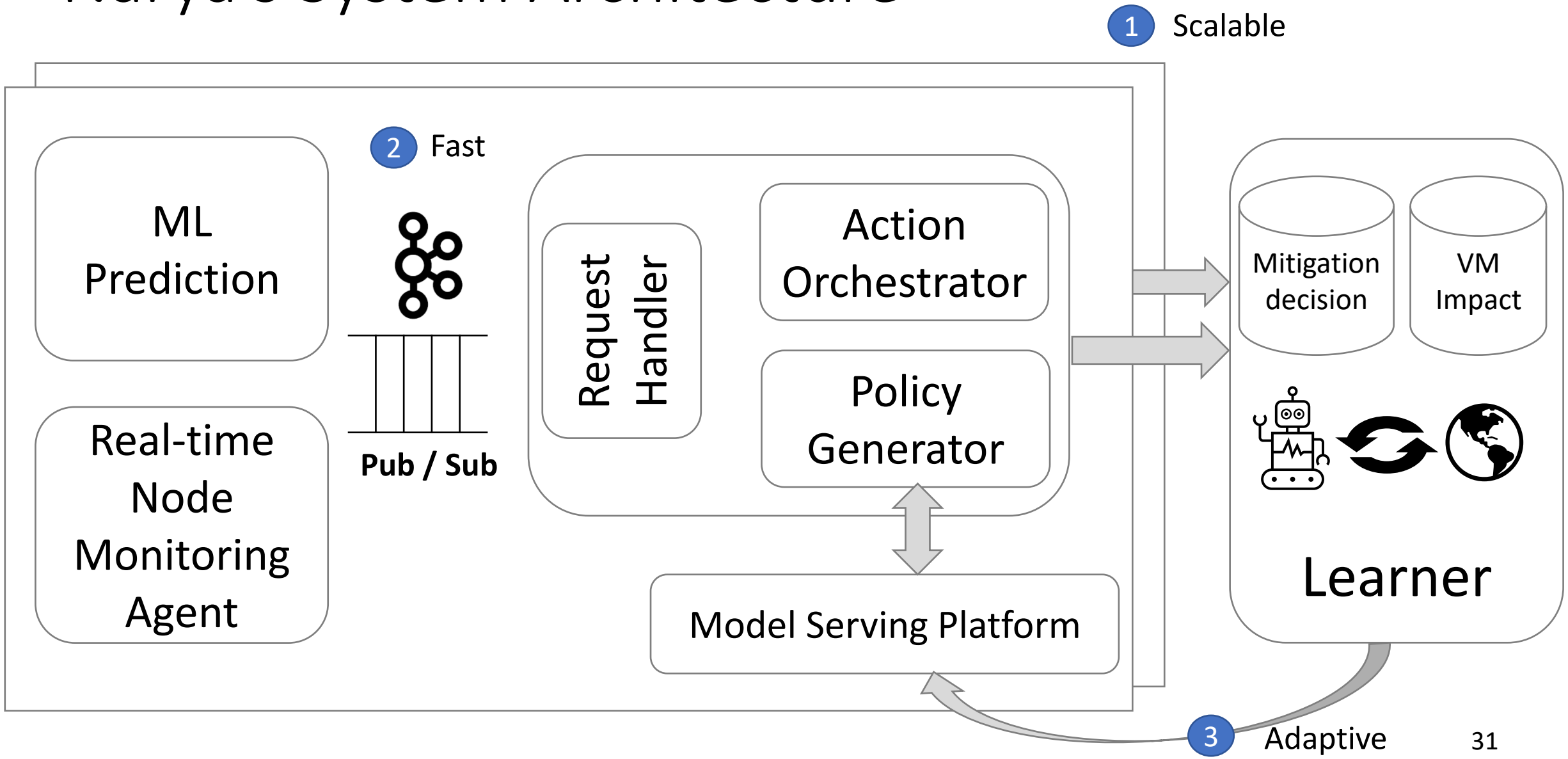
Safeguards

Ensuring Safe Exploration

- Only allow relevant composite actions
- Override logic if other more severe issues are detected

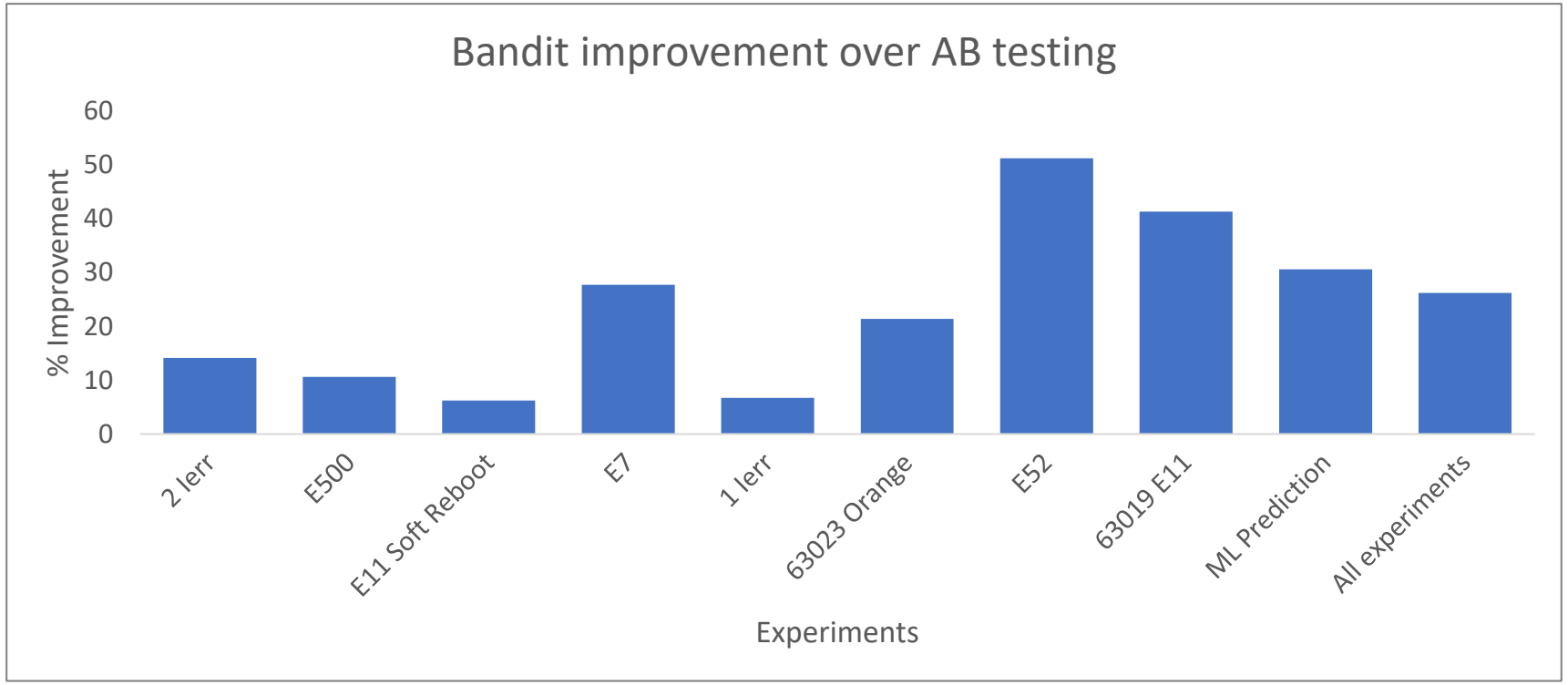
- Fallback to AB testing if there are not enough observation data
- Enforce minimum and maximum probability for each action

Narya's System Architecture



Overall Savings: 26%

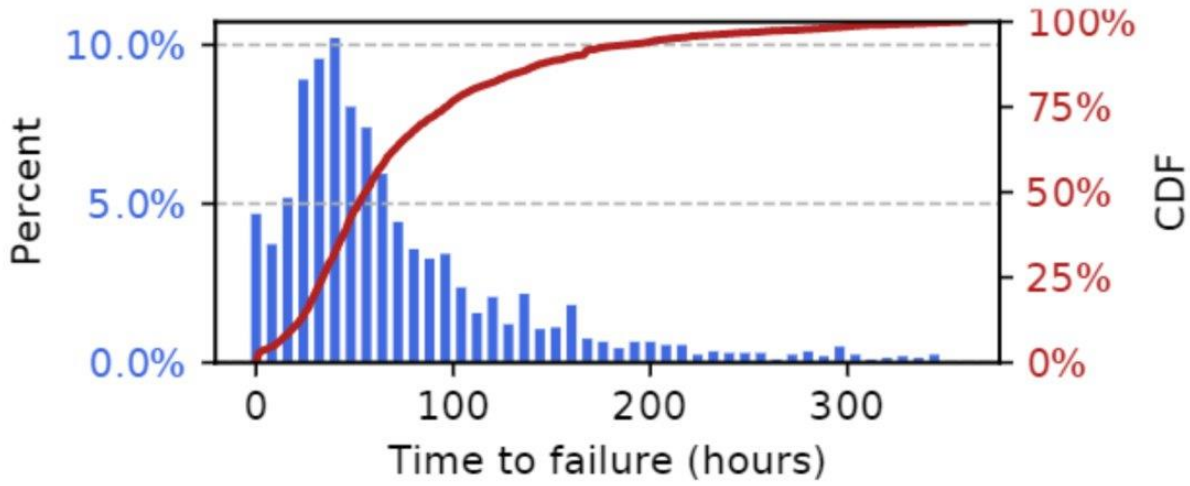
- Compare the use of AB testing/Bandit to previous static strategy
- Estimated AIR savings:
  [Observed AIR] – [Control group AIR mapped to whole fleet]



Prediction Performance

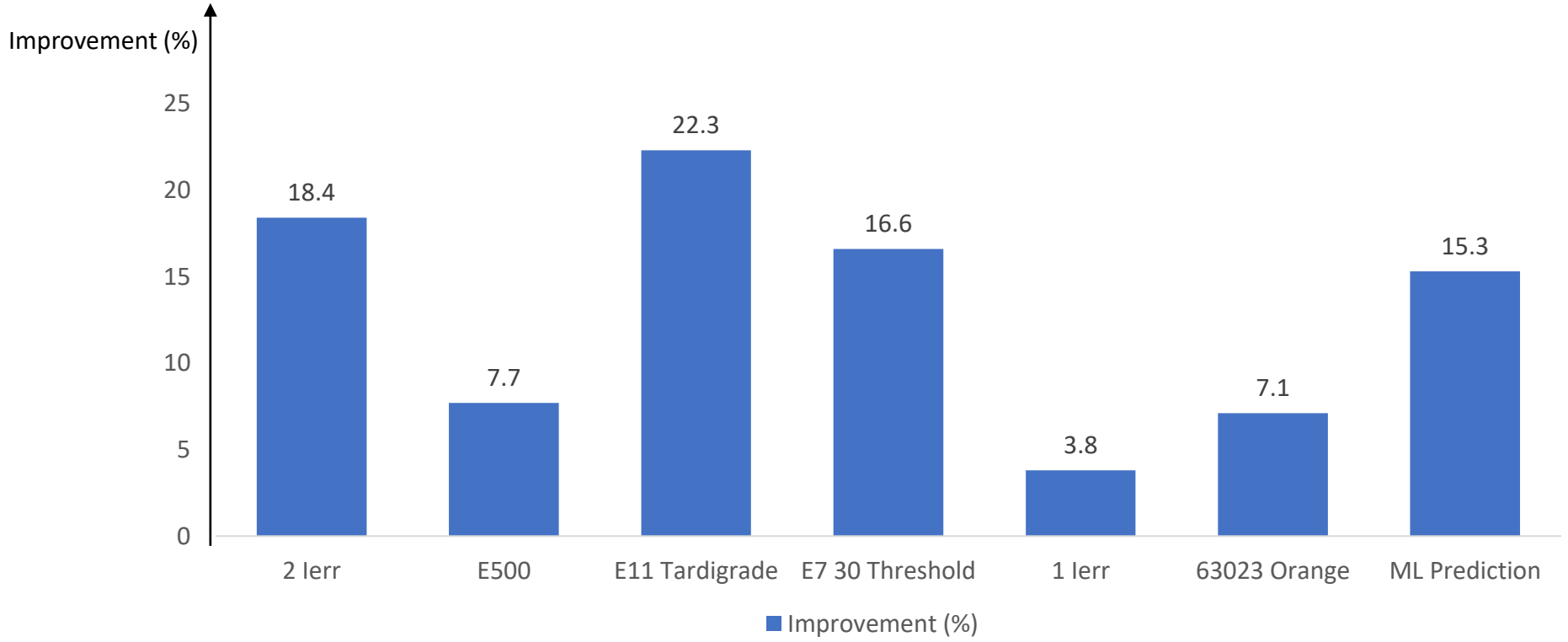
- Precision – 79.5%
- Recall – 50.7%

- ML prediction:
  Time to failure – 48 hours on average



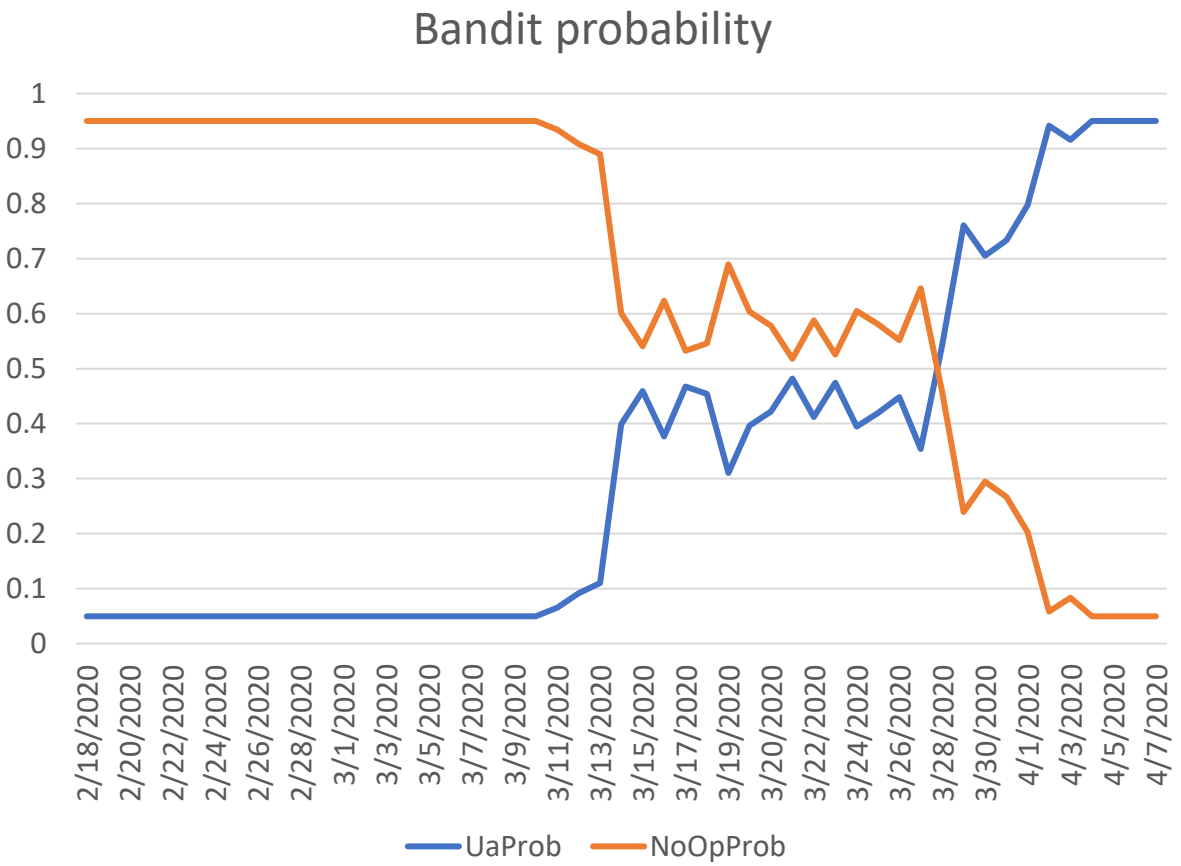
Compare AB Testing and Bandits

- Counterfactual estimation



Case Study: I/O Timeouts

- A/B testing then Bandit experiment on I/O timeouts prediction rule
- Unexpected system changes switched probability from using NoOp to using UA-LM-RH automatically
- Although change is not understood, bandit can automatically adjust



UA-LM-RH: Unallocatable + Live Migration + Reset Node Health

Operation Learnings

- Many factors influence efficiency: need for probabilistic approaches
- Decisions may go wrong: closely monitor all components behavior and use interpretable models
- Data quality is critical: watch for telemetry schema changes
- Customer impact is complex: human in the loop helps prevent from new types of impact

Summary / Takeaway Message

- Both failure prediction and proactive mitigation is critical
- No one-size-fits-all mitigation strategy, adapting different mitigation strategies in an online fashion

- Narya uses AB testing and multi-armed bandit to proactively and adaptively mitigate of predicted bad nodes
- Narya achieved 26% improvement over previous static strategy

Thank you!

- Acknowledgement
  - Haryadi Gunawi, our shepherd, and the anonymous reviewers
  - All our collaborators within Azure

- Contact us
  - selevy@microsoft.com
  - ranyao@microsoft.com
  - yow@microsoft.com
  - yidang@microsoft.com